UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AN INCLUSIVE END-USER DEVELOPMENT FRAMEWORK FOR TAILORABLE GAMES

FRANCO EUSÉBIO GARCIA

SUPERVISOR: VÂNIA PAULA DE ALMEIDA NERIS

São Carlos, São Paulo, Brazil

March/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AN INCLUSIVE END-USER DEVELOPMENT FRAMEWORK FOR TAILORABLE GAMES

FRANCO EUSÉBIO GARCIA

Thesis presented to the Programa de Pós-Graduação em Ciência da Computação of the Universidade Federal de São Carlos, as part of the requirements to obtain the title of Doctor of Computer Science, Concentration Area: Methodologies and Techniques of Computation.

São Carlos, São Paulo, Brazil

March/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Tese de Doutorado do candidato Franco Eusébio Garcia, realizada em 30/04/2019:

[illegible]

Profa. Dra. Vânia Paula de Almeida Neris
UFSCar

Daniel Lucrédio

Prof. Dr. Daniel Lucrédio
UFSCar

[illegible]

Profa. Dra. Renata Pontin de Mattos Fortes
USP

Profa. Dra. Maria Cecilia Calani Baranauskas
UNICAMP

Prof. Dr. Flávio Soares Corrêa da Silva
USP

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Maria Cecilia Calani Baranauskas, Flávio Soares Corrêa da Silva e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

[illegible]

Profa. Dra. Vânia Paula de Almeida Neris

This thesis is dedicated to you,
the reader.

Inclusion towards universal access is a collaborative effort.

Every one shall contribute.

*There is no "*us*" us without me, you, him, her, them.*
We need you in this endeavor.

Acknowledgements

The first acknowledgments are for my family, friends, and to you, who is reading this text. Thank you for your time and wiliness. I hope you find it useful – and that you can extend it with your own perspectives, knowledge, skills, abilities, and goals.

Then, without specific order, I would like to thank all friends, professors, professionals, authors, and participants who have helped me – directly or indirectly – over a lifetime. This research would have been impossible without the help of the people who kindly provided their times and efforts to participate in our studies. Likewise, it would had been impossible without the authors who wrote books and papers, scientists and researchers, software and hardware makers. Finally, from the people who created the games I have played over my life – as well as accessible games to expand gaming to new audiences.

I would like to thank the people from the university and the department for the aid over the years. Professors, staff, workers, students.

In particular, Vânia, who has been supervising me since 2011, as well as my friends from Laboratório de Interação Flexível e Sustentável (LIFeS) – and all the undergraduate students and professors who have supported me over the years. In the doctorate, Roberta and Gabriel allowed people to appreciate using our systems, thanks to their artistic skills and great taste. Renata helped us with the interface for the first prototype of the creation tool – of which resulted Lepi. Paula helped us in our collaborative activities. Fernanda and Angélica provided their healthcare expertise. Babette helped us with her education and accessibility expertise. Pedro who has just joined us to further develop support systems. Finally, to Carmo, who has kindly accepted to review our definitions.

I also would like to thank all the undergraduates who helped me over the graduation years. Alessandra, Diogo, Eduardo, Flávia, Gabriel, João Pedro, Leandro, Lucas, Marcos, Paulo, Pedro(s), Rafael, Renata, Roberta, Vinícius, and Vitor. As well as my friends from LIFeS. Benedito, Everton, Isaque, Kamila, Letícia, Maíra, Maílson, Marcel, Maria Carolina, Newton, Paula, Renata(s) Rogério, Tatiana, Vinícius, and Willian.

I would also like to thank all the participants who created and played games using our framework once again. Although I cannot name you, this research would had been impossible without you. In advance, I would also like to thank our future participants.

Finally, I would like to thank the funding agency who supported my studies.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

The world of games is so varied and complex that there are numerous ways of studying it. Psychology, sociology, anecdotage, pedagogy, and mathematics so divide its domain that the unity of the subject is no longer perceptible. Not only are such works as Homo Ludens by Huizinga, Jeu de l'Enfant by Jean Chateau, and Theory of Games and Economic Behavior by von Neumann and Morgenstern not aimed at the same readers, but they don't even seem to be discussing the same subject. One may finally ask to what extent semantic problems cause different and nearly incompatible researches to be viewed as at heart concerning the same specific activity. It has even been doubted that there are any common characteristics on the basis of which play may be defined in order to facilitate a cooperative approach to studying it.

— Caillois (2001)

Resumo

Seguindo o aumento de propósitos e aplicações de jogos digitais, a criação tem se popularizado nos últimos anos. Tanto indústria quanto academia têm estudado e criado ferramentas, métodos, e técnicas para que usuários finais desenvolvam seus próprios jogos. Entretanto, públicos-alvo ainda são restritos (normalmente a jovens anglófonos sem deficiências físicas, cognitivas, e emocionais). Para superar barreiras, faz-se necessário permitir que pessoas com diferentes habilidades e necessidades de interação possam contribuir com o desenvolvimento. Esta tese apresenta um *framework* para o desenvolvimento de jogos ajustáveis por usuários finais, com o objetivo de promover criação inclusiva de jogos inclusivos. O lema desta pesquisa é jogos por e para todos. A tese parte da hipótese de que se usuários finais utilizassem ferramentas de criação adequadas às suas necessidades de interação e seguissem um modelo de contribuições colaborativas para melhorar iterativamente funcionalidades de acessibilidade a serem inseridas em uma arquitetura de software capaz de modificar a interação humano-computador em tempo de uso, então eles poderiam criar jogos que satisfizessem necessidades de interação heterogêneas de possíveis jogadores. Nesta tese, explorou-se pesquisa documental (revisão sistemática para a identificação de abordagens existentes), criação e especificação (para sistemas, modelos, e arquiteturas de software necessárias), e pesquisa de campo (para entendimento de necessidades de interação e avaliação do *framework*). O *framework* compreende uma arquitetura formal de software para implementação de jogos inclusivos, um modelo de colaboração para co-criação de inclusão, e uma plataforma inclusiva para criação de jogos ajustáveis por usuários finais (Lepi). O *framework* foi usado por um público não considerado pela Literatura (pessoas adultas – algumas das quais com baixo letramento e que nunca tinham usado computadores – em acompanhamento para reabilitação de abuso de álcool e drogas) por quatro meses. Resultados mostram que os participantes conseguiram criar jogos que satisfizessem tanto às suas próprias necessidades de interação, quanto às de pessoas com necessidades diferentes (confirmando, portanto, a hipótese de pesquisa). Observou-se ainda que as atividades de criação e uso transformaram os participantes. Os benefícios foram além da co-criação de software, rumo a auto-melhorias. Desta forma, contribui-se com criação e acesso participativo e universal do cidadão brasileiro ao conhecimento. Como a arquitetura permite a inclusão contínua de novos públicos por meio da inserção de novas alternativas de uso, espera-se continuar permitindo novos públicos às práticas de criação e uso – pequenos passos dos usuários, grandes saltos rumo ao acesso universal.

Palavras-chave: Acessibilidade em Jogos. Acessibilidade. Acesso Universal. Design Participativo. Design Universal. Design de Jogos. End-User Development. Human-Centered Computing. Interação Humano-Computador. Jogos. Sistemas Colaborativos.

Abstract

Following the growing purposes and applications of digital games, the creation of this interactive media has been becoming more popular. Industry and academy alike have studied and created tools, methods, and techniques to allow end-users to develop their own games. However, intended audiences are still restricted (normally to anglophone youngsters without physical, cognitive, and emotional disabilities). To overcome barriers, it is necessary to allow people with different interaction abilities and needs to contribute on game development. This thesis presents an end-user development framework for tailorable games, with the purpose of promoting inclusive creation of inclusive games. Our lemma is games for everyone, by everyone. The thesis starts from the hypothesis that if end-users used creation tools suitable to their interaction needs and followed a collaborative work model to iteratively improve accessibility features to be inserted into a software architecture able to modify human-computer interaction at use-time, then they would be able create games satisfying heterogeneous interaction needs of possible players. In this thesis, we have explored documental research (systematic review to identify current approaches), creation and specification (for required software systems, models, and architectures), and field research (to understand interaction requirements and evaluate the framework). The framework encompasses a formal software architecture for implementing inclusive games, a collaborative work model for co-creation of inclusion, and an end-user game development platform (Lepi). Our framework was used by an audience not yet considered by the Literature (adult people – some of which with low literacy and that had never used a computer before – undergoing supervision for alcohol and drug abuse rehabilitation) over four months. Our results show that the participants were able to create games that satisfied their own interaction needs, as well as those from people with different needs than theirs (therefore, they have confirmed our research hypothesis). We have also observed that game creation and play transformed the participants. The benefits went beyond software co-creation, towards self-improvements. In this way, we have contributed with participatory and universal to creation and access of knowledge by the Brazilian citizen. Moreover, as the architecture allows continuous inclusion of new audiences via the addition of new alternatives of use, we hope that we can continue enabling more people to create and play – small steps from users, giant leaps towards universal access.

Keywords: Accessibility. Collaborative Systems. End-User Development. Game Accessibility. Game Design. Games. Human-Centered Computing. Human-Computer Interaction. Participatory Design. Universal Access. Universal Design.

List of Figures

List of Tables

List of definitions

List of axioms, lemmas, and theorems

List of listings

List of abbreviations and acronyms

2D	Bi-dimensional
3D	Tri-dimensional
AI	*Artificial Intelligence*
ACM	Association for Computing Machinery
API	Application Programming Interface
ASL	American Sign Language
BCI	*Brain-Computer Interface*
C3-I	Collaborative Co-Creation of Inclusion
C3-IDG	Collaborative Co-Creation of Inclusive Digital Games
CAPS-AD	Centro de Atenção Psicossocial – Álcool e Drogas
CBT	*Cognitive Behavioral Therapy*
CSCW	Computer-Supported Collaboration Work
CT	*Computational Thinking*
CTP	*Computational Thinking Patterns*
DD	*Data-Driven*
DDA	*Data-Driven* Architecture
DL	Digital Literacy
DLL	Dynamic Link Library
DSL	*Domain-Specific Language*

ECS	*Entity-Component System*
EDA	*Event-Driven Architecture*
EDP	*Event-Driven Programming*
ET	*Elemental Tetrad*
EUD	*End-User Development*
EUGD	*End-User Game Development*
EUP	*End-User Programming*
FPS	First-Person Shooter
GBL	Game Based Learning
GL	*Gaming Literacy*
HCC	*Human-Centered Computing*
HCI	*Human-Computer Interaction*
HDD	Hard of Hearing
HRTF	Head-Related Transfer Functions
HTML	Hypertext Markup Language
HUD	Head-Up Display
ICT	*Information and Communications Technology*
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IO	Input and Output
JSON	JavaScript Object Notation
LIBRAS	Língua Brasileira de Sinais
LISP	*List Processor*
MD	*Meta-Design*
MG	Meta-Game
ML	Media Literacy

MSc	Master of Science / Master's Degree
MVC	Model-View-Controller
NPC	Non-Playable Characters
PD	*Participatory Design*
PbD	*Programming by Demonstration*
PhD	Doctor of Philosophy / Doctorate
RIS	Real-Time Interactive System
SDK	Software Development Kit
UA-Game	*Universally Accessible Game*
UDG	*Universal Digital Game*
UD	*Universal Design*
UGE	*UA Game Engine*
UI	User interface
URL	*Uniform Resource Locator*
UUI	*Unified User Interface*
UnD	*Unified Design*
VP	*Visual Programming*
VPL	*Visual Programming Language*
VR	Virtual Reality
XML	Extensible Markup Language

Contents

CHAPTER 1

INTRODUCTION

The purposes of digital games expand beyond fun – professionals of education, training, and rehabilitation have been using games to support their practices. New types of literacy – such as *Digital Literacy*, *Media Literacy*, and *Gaming Literacy* (GL) (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012) – define *Information and Communications Technologies* (ICTs) and games ways of learning, discovering, acquiring knowledge, and culture. GL, for instance, emphasizes that playing promotes the development of abilities related to *Computational Thinking* (CT), *storytelling*, communication, and problem solving skills in different knowledge areas (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012).

GL promotes games as a new way of literacy; thus, it becomes necessary to increase the intended audience; people who can play games. Yet, Aguado-Delgado et al. (2018) found that there are, currently, no study guaranteeing universally accessible games. Although there are design guidelines and techniques (YUAN; FOLMER; HARRIS, 2011; International Game Developers Association, 2004; ELLIS et al., 2013; BARLET; SPOHN, 2012; GARCIA; NERIS, 2013b; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; GARCIA; NERIS, 2014; GARCIA, 2014; TORRENTE et al., 2015), the industry does not follow them. Rather, software developers often make assumptions regarding the abilities of their end-users (POZZI; BAGNARA, 2013). As a result, ICTs may become inaccessible to people whose abilities deviate from these assumptions. According to Mankoff, Hayes & Kasnitz (2010), “the person designing a piece of software is, in some sense, defining who is disabled with respect to that software”. This also happens in digital games – “whether people are disabled when playing the game is entirely up to the designers and developers involved”, according to a participant in a study by Westin (2017).

In a recent systematic review, Aguado-Delgado et al. (2018) found that:

None of the studied initiatives can guarantee universally accessible video games:

1. Approaches that involve particular interfaces and custom developments may lead to a lack of quality in the generated/adapted video games and the segregation of disabled players.
2. The frameworks often depend on the use of specific technologies. This dependence hinders accessibility, since it necessarily involves the use of elements that can be beyond the control of the developer (they belongs [*sic*] to third parties) and limit the available options.
3. The guidelines, techniques, strategies, etc. that facilitate the development of accessible video games are ignored by many developers (due to ignorance, lack of comprehension, impossibility of application, etc.).

What if we designed software without assuming particular ways of interacting with the system? Instead of assumptions, we could define functionalities and semantics of use – commands for interaction, *what* instead of *how*. Then, we could enable how people use the system iteratively, according to their abilities. We defined this approach for games in (GARCIA, 2014). We have been since improving it. In particular, we have also expanded the research into a new direction: towards *End-User Development* (EUD) (LIEBERMAN et al., 2006) of digital games (hereafter called *End-User Game Development* (EUGD)). In this new thesis, we enabled people, who, potentially, had never played digital games or interacted with a computer, to create games for themselves and for their peers. From each according to her/his abilities, to each according to her/his needs. The community for the individual, the individual for the community.

1.1 Problematic and Research Question

The systematic review from Aguado-Delgado et al. (2018) reveals that inclusion is an open issue in digital games. When we started this research, game creation focused a specific demographic; for instance, in education contexts, it target primarily young students Earp (2015).

More recently, we can observe efforts to promote inclusive creation by and for other audiences (for instance, in (URBANEK; GÜLDENPFENNIG; SCHREMPF, 2018; GIANNAKOPOULOS et al., 2018; YILDIZ et al., 2018)). Yet, game accessibility can become more inclusive. The 4th Grand Challenge in Computer Science Research in Brazil (CARVALHO et al., 2006) – participative and universal access to knowledge for the Brazilian citizen – describes the importance of promoting universal and participatory access to information. This importance, as stated by the Challenge, results from the necessity of enabling people to perform active roles in the creation of knowledge.

Thus, we shall not restrict ourselves to passive access. Rather, we want to go beyond, towards inclusive creation – of digital games, in this research. This motivated our research question: "how can we enable more people to create and play digital games, respecting and satisfying their own interaction needs?"

Although we may not guarantee universal access for every digital system and game, we can always make them more inclusive, accessible, and usable for everyone. We, thus, started working towards an End-User Game Development Framework For Everyone, By Everyone.

1.2 Goal

As previously stated, our goal in this research was to enable more people to create and play digital games. In this study, we focused on small scale digital games with few mechanics – for instance, similar to Atari games such as Pong and Space Invaders –, aiming for broader inclusion. We are considering inclusion as a broad term, encompassing, for instance, physical, cognitive, and emotional (dis)abilities, age, language, education and knowledge, culture, socioeconomic factors, computer literacy and skills, and even time and context of use. By this definition, inclusion encompasses the concepts of accessibility, usability, and inclusion defined in World Wide Web Consortium (2016).

1.3 Research Hypothesis

Our research hypothesis was that if end-users used creation tools suitable to their interaction needs and followed a collaborative work model to iteratively improve accessibility features to be inserted into a software architecture able to modify human-computer interaction at use-time, then they would be able create games satisfying heterogeneous interaction needs of possible players.

The hypothesis defined the three pillars of this thesis, which we grouped into an Inclusive End-User Game Development Framework[1] for Tailorable[2] Games:

1. A Run-Time Tailoring Software Architecture for Games;
2. Lepi, a game creation platform implemented exploring the architecture and generating games using the architecture;

[1] The Cambridge Advanced Learner's Dictionary & Thesaurus (2017) define the noun *framework* as: "[C] *a supporting structure around which something can be built;* [C2] *a system of rules, ideas, or beliefs that is used to plan or decide something*".

[2] Tailoring (NERIS; BARANAUSKAS, 2009; NERIS, 2010) proposes adapting systems according to their context of use. In particular, we explore tailoring in this thesis for accessibility purposes. Tailoring is a metaphor for adjusting a system to make it "tailor-made", "custom-made" according to the needs of the users (GARCIA, 2014).

3. A Collaborative Work Model for Co-Creation of Tailorable Digital Games – hereafter named Collaborative Co-Creation of Inclusive Digital Games (C3-IDG).

We define the elements of the architecture formally using set theory, to ensure their correctness[3]. From the theory, we defined the run-time tailoring algorithm, which enables software to re-redefine its user interaction at run-time, according to definitions stored in an Interaction Profile.

The algorithm was implemented in Godot Engine[4], to show that we can explore interaction redefinition in state of the practice game development middleware. Lepi was, afterwards, implemented using the algorithm, as a prototype of tailorable system to develop tailorable games. As proof of concept, Lepi provided slots (abstracting interaction alternatives) to enable end-users to create games for people who could read, listen, and/or understand Língua Brasileira de Sinais (LIBRAS). If a player could understand at least one of these alternatives, she/he could be able to play the resulting game (for instance, a blind player could play using audio alternatives).

Finally, the C3-IDG orchestrated end-user creation of tailorable games. It provided a process to help people to co-create tailorable games for themselves and for their peers. In the C3-IDG, game development promotes inclusion – we try to enable as many people to co-create and play games as possible, respecting their abilities and interaction needs.

The framework has been evaluated in ten meetings spanning four months by adults undergoing alcohol and drugs rehabilitation at a Centro de Atenção Psicossocial – Álcool e Drogas (CAPS-AD). Ten users of the service participated in game creation activities. They had different interaction needs, levels of literacy, and experience using computers and games. With their abilities, they created games using Lepi. With the help from collaborators, they could generate games suiting their own interaction needs and those of their friends – everyone could play the game from everyone else. This provided evidence to accept our research hypothesis – the three pillars enabled a heterogeneous public to create games for themselves and all their peers.

1.4 Research Approach

In this thesis and over the development of the framework, we have performed documental research, creation and specification, and observational research.

For the documental research, we have employed a systematic literature review (KITCHENHAM, 2004). We used a systematic literature review to identify approaches described in

[3] At the final version of this thesis. It is currently undergoing a review for improvements (and a possible simplification).

[4] <https://godotengine.org/>

the Literature when we started this research. The review tried to identify tools, methods, techniques, and strategies which enabled end-users to create their own games.

For defining the models, architectures, and required software systems explored over this thesis, we have performed creation and specification activities. These activities resulted in the implementation of the pillars of the framework.

Finally, we performed field research and case studies as observational research to understand interaction requirements and evaluate the framework. These methods contributed to further understand the interaction needs of particular audiences with subsets of interaction needs (low literacy, hearing disabilities), as well as refine, iterate, and evaluate the framework on subsequent creation and specification cycles.

1.5 Summary of Contributions and Results

As we will mention in , Appendix A provides a summary of the research and its results. This section lists major and minor contributions and results of this thesis.

1.5.1 Major Contributions and Results

- Architecture: multiple games may co-exist within a unique Meta-Game.
  - Composable interactions;
    - Accessibility as add-ons / extensions / plug-and-play tailorable user interfaces.
  - Semantics for input and output;
  - Commands as intents;
  - User interfaces as interaction re-mapping via run-time tailoring. We can always reshape how people perceive and control digital systems and games.
  - Heuristic / Corollary: Can we define an algorithm which runs in suitable time for the game (usually in milliseconds to a couple seconds) to enable use?

    Whenever we can, we can enable creation and play.
- Work model: inclusion can be a dynamic, iterative, incremental, collaborative, end-user powered process.
  - Communistic collaborations;
  - Mutualistic collaborations;
  - Everyone can collaborate: contributions comes from abilities and skills;

  - Interdependence in software may promote independence, once assistance become part of the system.

- Lepi: tailorable editor for tailorable games.
  - Abstracts creation with the architecture;
  - Implements the architecture itself – we can enable more people to create;
  - Makes inclusion part of the development process – slots show existing/missing alternatives of use.

- CAPS-AD: transforming lives is a major contribution.
  - Digital inclusion;
  - Self-growth.

1.5.2 Minor Contributions and Results

- Development method to design and implement games with the architecture;
- Capacitation course for professors, providing training in serious games, educational games, and using Lepi as a classroom tool for teaching;
- Source code repositories with examples and videos.

1.6 Thesis Organization / How to Read This Text

This thesis starts at this chapter (Chapter 1) and ends at Chapter 4. In the default order, we present the core results of the thesis: a synthesis of the frameworkChapter 2 and its evaluationChapter 3.

Chapter 2 explains how the framework contributes to the goals defined in Section 1.2. It defines each pillar, then discusses how they work together to promote inclusive creation of tailorable games to enable inclusive play.

Chapter 3 describes our evaluation outlined in Section 1.2. It presents a real world scenario which used to framework as a tool to assist healthcare practices, on which participants with heterogeneous interaction needs created and played their own games.

The appendices are optional. The reader might refer to them as needed. Readers who want a quick summary of this thesis can refer to Appendix A. It outlines the research approach and its results in items written in (we hope) simple language. Therefore, even if optional, it is, perhaps, a good starting point for reading this text.

Appendix B and Appendix C[5] details the architecture (its formalization and implementation), and Appendix D defines the collaborative work model (rationales, models, workflows, and opportunities for collaboration based on abilities).

Although you can follow this order, there are many other ways to read it. In many ways, this work is systemic – the whole is more than the combination of its parts. Thus, although each pillar of the framework is standalone on its own, they also complement, extend, and depend on one another.

First, there is the reductionist approach. Readers with a stronger development or programming interests may benefit from this approach. It goes from technology to people. It starts from the architecture, moving to the collaborative work model, then to the development tools to form the framework. In this order, we establish the underlying theory first, from its foundation – elements of the architecture, their properties, and relationships. To follow this order, explore the thesis in the order:

1. Appendix B;
2. Appendix C;
3. Appendix D;
4. Chapter 2;
5. Chapter 3.

Next, there is the holistic approach. Readers with a stronger design or collaborative work or Human-Computer Interaction (HCI) interests may benefit from it. It goes from people to technology. It synthesizes the work, trying to bring apparently different concepts together. To follow this order, explore the thesis reading:

1. Chapter 2;
2. Appendix D;
3. Chapter 3;
4. Appendix B;
5. Appendix C.

There is also an applicable approach. Readers who would rather want to apply this research into practice rather analyzing its intricacies might benefit from it. Readers who want to implement software using the architecture should read:

[5] New findings regarding the architecture described in Appendix B, based on Computation Theory.

1. Appendix B;
2. Appendix C;
3. Chapter 2;
4. Appendix D;
5. Chapter 3.

Readers who want to implement a variation of our collaborative work model should follow:

1. Appendix D;
2. Chapter 2.

Readers who want to use Lepi as a standalone tool may benefit from:

1. Chapter 2;
2. Appendix D.

Readers who want to use the framework without much reading can consult:

1. Chapter 2.

There is also the approach for practical programmers. Instead of text, readers may skip directly to the algorithms and source code listings – or the source code repositories directly. For the latter, search this text with the string `francogarcia` – the results will provide Uniform Resource Locator (URLs) for the repositories.

If you are looking for ideas for your dissertation or thesis (or want to contribute with advancing this research), refer to Section 4.3. Finally, you can choose a page or section at random, or consult the Table of Contents, find something interesting, and read it.

Chapter 2

A Framework for Tailorable Games: Towards Inclusive End-User Development of Inclusive Games

2.1 Introduction

In the digital games domain, there are efforts to enable end-users to develop their own games. Game modifications (modding) and game making popularized the practice, allowing end-users (modders) to modify games to their interests and creativity. From the academic perspective, End-User Development (EUD) promotes end-users to non-professional software developers, aiming to empower them.

Educators propose new types of literacy – such as Digital Literacy, Media Literacy, and Gaming Literacy (GL) (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012) –, considering that Information and Communication Technologies (ICTs) and games can develop abilities related to computational thinking (CT), storytelling, communication, and problem solving (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012).

From the industry side, EUD for game development (hereafter called End-User Game Development – EUGD) is an important practice as well. When end-users improve their favorite games and share their modifications, they can learn and self-express, benefit other players, studios and publishers, and extend the overall quality and shelve life of the original product – potentially increasing the lifetime, popularity and sales (GEE; TRAN, 2015; SCACCHI, 2011; El-Nasr; SMITH, 2006; POOR, 2014; POSTIGO, 2008; NIEBORG; VAN, 2008).

Like other digital devices and media, games can enhance the life of people.

Learning, working, entertainment, health, communication (and other areas) can benefit from games (SCHELL, 2008; CAILLOIS, 2001; CHEUNG, 2006; MADER; NATKIN; LEVIEUX, 2012; EARP, 2015; GEE; TRAN, 2015; GAMES, 2010; HAREL; PAPERT, 1991; El-Nasr; SMITH, 2006; SALEN, 2007; HUIZINGA, 2016). In a world where virtual environments are starting to merge with real one, being unable to interact with ICTs may hinder opportunities. In this context, universal access becomes a key aspect to promote digital and social inclusion. In the ideal scenario, this promotion should support use and creation alike – technology should suit people's needs to enable them to interact and develop ICTs.

Considering the emergency of digital world into the real one, we should strive to promote inclusion[1]. Ideally, we should empower people to create as well as use digital systems, regardless of (dis)abilities. Game design and programming are complex disciplines requiring specialized knowledge. Schell (2008), for instance, mentions that game designers benefit from a very wide range of skills and knowledge. A limited, non-exhaustive list may include disciplines such as animation, anthropology, architecture, art, communication, economy, engineering, mathematics, writing, and psychology. Moreover, the Elemental Tetrad proposed by Schell (2008) divides game design into an interconnected composition of four elements – mechanics, aesthetics, story, and technology. According to Schell (2008), technology are "any materials and interactions that make your game possible"; aesthetics define "how your game looks, sounds, smells, tastes, and feels"; story unfolds the "sequence of events that unfolds in your game", and mechanics provide "procedures and rules of your game".

Although it may seem as an obstacle at first, should we adopt a different perspective; the variety of required skills suggests that virtually everyone may contribute to game design activities based on their own strengths. If we focus on abilities, we can enable people to co-create digital games.

However, for playing, although human abilities, knowledge, culture, and soft skills vary, in a game accessibility context, the game industry intended audience is, mostly, homogeneous (International Game Developers Association, 2004; YUAN; FOLMER; HARRIS, 2011; GARCIA, 2014). In this text, hereafter, we use the expression "average user" to refer to users belonging to a normal distribution of all users (FISCHER, 2001; NERIS, 2010). For game playing, average users for games are usually alphabetized people, without disabilities (GARCIA, 2014).

For game creation, the public seems even narrower. In a systematic review from Earp (2015) considering an educational context, only 6 out of 494 studies (1.21%) performed activities with people at least 23 years old. In contrast, 351 studies (71.05%)

[1] In this chapter, we consider inclusion as a broad term, as defined in World Wide Web Consortium (2016).

targeted a younger audience – children, youngsters, and teenagers. Earp (2015) further notices that inclusion in most studies explore "game making as a strategy for addressing the underrepresentation of girls and women in computing". Age and gender are, still, only two variables when we consider diversity.

In this chapter, we explore a different approach. In this chapter, we are considering very small scale, simple and slow paced digital games, with the intent of achieving broader inclusion for creation and play. Thus, we consider games focused on a few game mechanics (for instance, clones of Atari games such as Pong and Space Invaders), as a re-imagination of digital game development from its start, focusing on accessibility to enable more audiences to create and to play. Our goal is enabling inclusive development of tailorable games[2], with the lemma of "games by everyone, for everyone". The lemma does not mean that everyone will become able to create and play every game. Rather, it means that games can always become more inclusive, accessible, usable, and fun for more people, and that we can always enable new audiences to co-create them.

For broader inclusion, a possible way to enable more people to play is to enable more people to create. Creation, thus, predates playing, considering diversity and inclusion from the start. Playing and creating make two fronts:

***Game playing* (*use*)** The process of playing a game. For the player, there should be her/his user interface (UI); the UI that suits her/his own interaction needs. For the end-user acting as player, what truly matters is having suitable input and output (IO) features enabling her/him to play. This means that we can explore multiple accessible UIs in a very same game. We can provide people with choice. They can interact with the standard game interface (as it happens now), use a pre-defined version suitable for their abilities, or even combine existing interaction alternatives to create their own.

***Game creation* (*development*)** The process of creating or modifying a game, including design and programming activities. In this case, accessibility would allow for new audiences to make their own games. For broader inclusion, approaches (tools, methods, techniques, and practices) should not be (exclusively) programming-centered. Like game playing, we can provide multiple interaction alternatives to enable creation. For the end-user acting as creator, the approach should suit her/his needs to enable her/him to make games.

To the best of our knowledge, game accessibility currently guides researchers and professional developers to create more inclusive content to players. For smaller, simpler

[2] Tailoring a metaphor for the activities of tailors' (NERIS; BARANAUSKAS, 2009; NERIS, 2010; GARCIA, 2014). They adjust materials for the best fit. We follow this same idea here – tailorable games are games that provide alternatives for perceiving and command a game with the same rules.

games we can try to reach universal access (for instance, with Universally Accessible Games – UA-Games) (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011). For complex projects requiring fast response times, tight time constraints, and fast actions (for instance, games from big publishers – AAA games –, or even smaller fast paced action games), we should aim for inclusion. Regardless, we can always improve accessibility.

However, we do not have to think "one-size fits all" solutions for inclusion. Rather, we can support "one *core* fits most" *as well* as multiple accessible versions at the same time.

Different abilities lead to different ways of perceiving, processing, and providing information. We should acknowledge this fact; if the interaction needs of one person is different from another, we should not force them to play equally. Rather, we should respect the abilities, and promote the best experience to suit the abilities of each person. The core (logic and rules) of the game can be the same; the way to play it can be completely different. This way, we can provide options. If a person wants to play the core version, she/he can. If she/he wants to play an accessible version, she may. If she/he is able combine existing interaction alternatives to create her/his own custom tailored version, she/he might as well.

As proof of concept, we have chosen storytelling based games for our initial efforts. They were easier to explain to people who had never played games (or used a computer) before, focused on content creation, and allowed people with different skills to contribute and co-create. In the future, we aim to expand it, including new audiences first, then moving towards new mechanics, genres, and EUGD strategies over time.

Towards this goal, we have defined a framework to enable more individuals to create games – for themselves and for others. Regardless of physical, cognitive, emotional abilities, and technical knowledge, we aim to promote inclusion from design to use. Our hypothesis is that if end-users used creation tools suitable to their interaction needs and followed a collaborative work model to iteratively improve accessibility features to be inserted into a software architecture able to modify human-computer interaction at use-time, then they would be able create games satisfying heterogeneous interaction needs of possible players.

To verify the hypothesis, we have defined a framework with pillars: a collaborative work model, a run-time tailoring software architecture, and a game creation platform.

The work model describes a process for co-creation of tailorable digital games, on which people support each other, improving the interaction quality and existing alternatives iteratively. Following the model, an inclusive game results from successive iterations defining (adding or improving) interaction alternatives that, combined, results into broader inclusion. People who were, at a previous step, unable to interact may, upon inclusion, become contributors who may include new people, forming cycles of improvements (we will present more details in Section 2.6[3]). In this perspective, inclusion is constructed iteratively from abilities, skills, and collaboration.

The architecture describes a model which decouples logic from interaction. With the architecture, we attach interaction to games, (re-)defining arbitrary input and output (IO) interactions during use time, enabling end-users to select/switch IO devices and mappings to match their own needs whilst a system is running. In practice, the architecture enables us to explore any input device to provide commands, as well as any output device to convey information. If we can map game data to these devices, we can enable more people to play (we will discuss our approach in Section 2.4[4].).

Our principle is that software is indefinitely extensible at run-time within the limits of what is computationally possible; therefore, conventional ("one core fits most"), accessible, and even custom defined user interfaces can co-exist in the same digital system – and we can alternate among then at any time. As the logic is unique, we can, therefore, build user interfaces to "fit" or "adjust" interaction features to the needs of the user. As a tailor adjusts material to the best fit of her/his customers, we can tailor game interaction with our run-time tailoring algorithm.

Finally, the game creation platform is an EUGD tool, acting as proof of concept for inclusive end-user creation of tailorable games (Section 2.7). As a proof of concept, it currently supports text, audio, and sign language to convey the content of adventure, point-and-click and visual novel genres, as well as mouse, keyboard, and gamepads to provide input to it. Other IO media and devices (including assistive technologies) could be added for broader interaction needs. As interaction is composed, currently audiences are always included; whenever we define new interaction alternatives, we can enable more people to create and to play.

In its current state, our framework supported the hypothesis in a first evaluation scenario, on which people undergoing alcohol and drugs rehabilitation created games[5] (Section 2.8). These people had different interaction needs (including literacy and proficiency with ICTs). With the framework, they could create alternatives to satisfy needs of their peers, enabling them to play. In the remaining of this chapter, we describe our framework.

[3] For a detailed version, the reader may refer to Appendix D.

[4] For a detailed version, the reader may refer to Appendix B.

[5] We followed research ethics protocols throughout the entire processes – Certificado de Apresentação de Apreciação Ética from Plataforma Brasil: CAAE: 89477018.5.0000.5504. As we will mention in Section 2.8, although we have a second audience (including people with hearing and cognitive disabilities), we were not able not perform activities with them yet.

2.2 Related Work

In this chapter, we explore a tailoring approach to inclusion (KAHLER et al., 2000; PIPEK; KAHLER, 2006; NERIS, 2010). Tailoring proposes designing applications according to their context of use.

2.2.1 Game Accessibility

Digital games, in particular, are challenging applications for accessibility, both due to their complex, real-time, interactive nature, and to abilities required for interaction (GARCIA, 2014; YUAN; FOLMER; HARRIS, 2011). There exists accessible games for specific audiences (for instance, audio games for visual disabilities), as well as games aiming to include a broader public (for UA-Games) (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011) [6]. However, game accessibility is an exception instead of a rule – most games provide only basic accessibility features (for instance, subtitles).

The literature provides guidelines, strategies, and design processes to improve accessibility in games (for example, (CHAKRABORTY et al., 2017; DARIN; ANDRADE; SÁNCHEZ, 2018; DESURVIRE; WIBERG, 2015; PEREIRA et al., 2018; WESTIN et al., 2018; MALINVERNI et al., 2017; BERNARDO et al., 2016; URBANEK; GÜLDENPFENNIG; SCHREMPF, 2018; YILDIZ et al., 2018; MANGIRON; ZHANG, 2016; de Borba Campos; OLIVEIRA, 2016; CANO; FERNÁNDEZ-MANJÓN; GARCÍA-TEJEDOR, 2018; FORTES et al., 2017; BARLET; SPOHN, 2012; International Game Developers Association, 2004; YUAN; FOLMER; HARRIS, 2011; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; GARCIA, 2011; GARCIA; NERIS, 2013b), Game Accessibility Guidelines[7], and Accessible Player Experiences[8]). In particular, the Unified Design is a structured, participatory, user-centered, interactive process for designing games (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011). With the Unified Design, designers, domain experts (in accessibility and usability, for instance), and end-user collaborate to improve a game.

However, developers do not follow the game accessibility literature in practice (PORTER; KIENTZ, 2013; PORTER, 2014); reasons include costs and efforts to development, lack of support from game programming middleware, or disinformation (Aguado-Delgado et al., 2018; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; PORTER; KIENTZ, 2013; PORTER, 2014; YUAN; FOLMER; HARRIS, 2011; FORTES et al., 2017). Games rarely support assistive technologies; when they do, interaction quality is might

[6] For complex games, UA-Games can be unfeasible in practice.
[7] <http://gameaccessibilityguidelines.com/>
[8] <https://accessible.games/accessible-player-experiences/>

be poor (Aguado-Delgado et al., 2018; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; PORTER; KIENTZ, 2013)[9].

In this chapter, the Interaction Model from Yuan, Folmer & Harris (2011) is particularly useful. Yuan, Folmer & Harris (2011) defined game playing as a finite state machine, where players cycle between receiving stimuli, determining a response, and providing input – from the start of a playing session until its end. They have provided a summary of accessibility strategies for each state of the which were the basis to define our engine (UGE) (GARCIA, 2014) that, upon simplification, resulted into the architecture of our framework.

2.2.2 End-User Development and End-User Game Development

According to Lieberman et al. (2006), EUD:

> *EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact.*

According to Fischer et al. (2004), Meta-Design (MD):

> *Meta-design characterizes objectives, techniques, and processes for creating new media and environments that allow "owners of problems" (or end-users) to act as designers. A fundamental objective of meta-design is to create socio-technical environments that empower users to engage actively in the continuous development of systems rather than being restricted to the use of existing systems.*

In the context of EUD and MD, another approach to promote game accessibility would be enabling people to create their own accessible games. Towards this goal, we performed a systematic literature review (KITCHENHAM, 2004). For the purposes of this chapter, two review questions are important:

1. What are the approaches (strategies, tools, methods, techniques, and practices) which support end-users to create digital games?
2. What are the challenges and opportunities to improve practices?

We provided the search string (`"End-User Development" OR "EUD" OR "End-User Programming" OR "EUP" OR "Modding") AND ("Digital Game" OR "Game")`, to four academic databases: Association for Computing Machinery (ACM) Digital Library, Institute of Electrical and Electronics Engineers (IEEE) Xplore, Science Direct, and Scopus. The review was last updated at April 2016. We did restrict results from starting date.

[9] The Microsoft's Xbox Adaptive Controller (<https://news.xbox.com/en-us/2018/05/16/xbox-adaptive-controller/>) might help to improve assistive technology support in games.

We defined two lists with criteria for inclusion and exclusion of studies. For the most part, they were mutually exclusive; however, there was the possibility that a study was either unavailable or we could not access it. Thus, we rejected papers with at least one exclusion criterium.

Inclusion criteria:

SRI1 Study provides a report of a challenge related to EUGD;

SRI2 Study describes an approach to promote EUGD;

SRI3 Study describes the process of development employed by an end-user to create a game;

SRI4 Study describes an activity on which an end-user implemented or modified a game through programming;

SRI5 Study describes an activity on which and end-user created or modified a game without programming (she/he worked on aesthetics or story).

SRI6 Study describes a game design activity performed by end-users;

SRI7 Study describes how a end-user perceives the development of a game (for instance, how she/he thinks, acts, or wishes).

Exclusion criteria:

SRE1 Study does not describe any approach, challenge, or activity related to EUGD;

SRE2 Study is related to game development; however, it is does not promote EUD.

SRE3 Study is related to EUD; however, not to game development;

SRE4 Study is not related to digital games;

SRE5 Study is unavailable for access.

We did not exclude surveys from the results. The search returned 582 potential studies, 40 of which were duplicates. Of the remaining 542, 420 were rejected after we read the title, abstract, and meta-data. From the remaining 122 studies, we selected 63 studies matching our inclusion criteria:

- 23 studies described challenges related to EUGD;
- 41 studies described related approaches (32 describing EUD approaches, 9 using commercial games to promote the practice of modding);

- 15 studies described end-user creation processes;
- 15 studies described activities on which modification explored programming;
- 6 studies described activities on which end-user modification did not require programming (for instance, the focus was on aesthetics and/or storytelling);
- 1 study related how an end-user approached game creation.

Considering the selected studies, we could answer the review questions as follows:

1. Although exists multiple approaches that enable end-users to create their games[10], most of them contribute to the development of game technology. A common goal is reducing programming barriers to enable creation. For instance, Visual Programming (VP) was a common approach in selected studies – often used together with Event-Driven Architectures (EDA) and Event-Driven Programming (EDP), commonly present in games programmed by professionals (MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014).
2. For challenges and opportunities, we have noticed that few studies focused on the creation of new game mechanics and that current audiences for approaches were similar. For the latter, students were the most common audience for approaches (both as intended audience and for evaluation). In particular, the most common audience was composed by young anglophone students without disabilities.

More particularly, we could not identify studies that addressed accessibility towards inclusion of people with disabilities as game creators.

Game Creation and EUD

For studies involving programming from an EUD perspective, program creation or modification was the main goal of approaches. Visual Programming (VP) was the most common subcategory, explored by (AHMADI; JAZAYERI; REPENNING,

[10] For instance, AgentSheets and AgentCubes (IOANNIDOU; REPENNING; WEBB, 2009; REPENNING; IOANNIDOU, 2006a; IOANNIDOU; REPENNING; WEBB, 2008); Alice <http://www.alice.org/> (COOPER; DANN; PAUSCH, 2000); Game Salad <https://gamesalad.com/>; Game Studio <http://3dgamestudio.com/>; Game-Editor <http://game-editor.com/>; GameMaker <https://www.yoyogames.com/gamemaker/>; Gamestar Mechanic <http://gamestarmechanic.com/> (GAMES, 2010); Kodu <https://www.kodugamelab.com/> (MACLAURIN, 2009) and Project Spark <http://www.xbox.com/en-US/games/>; Panda 3D <http://www.panda3d.org/>; Phogram <http://phrogram.com/>; PyGame <http://www.pygame.org/>; RPG Maker <https://www.rpgmakerweb.com/>; Scratch <https://scratch.mit.edu> (RESNICK et al., 2009b); Sploder <http://www.sploder.com/>; ToonTalk <http://toontalk.com/>.

2012a; AHMADI; JAZAYERI, 2014; de Leeuw et al., 2007; FERREIRA et al., 2012; IOANNIDOU; REPENNING; WEBB, 2008; IOANNIDOU; REPENNING; WEBB, 2009; JAZAYERI; AHMADI, 2011; KOH et al., 2014; MARCHIORI et al., 2011; MARCHIORI et al., 2012; MOTA; FARIA; de Souza, 2012; MOUMOUTZIS et al., 2014; PERRONE; CLARK; REPENNING, 1996; REPENNING; AMBACH, 1997; REPENNING; IOANNIDOU, 2006a; REPENNING, 2006; REPENNING, 2011a; REPENNING, 2011b; REPENNING, 2013; TONON; BAECKER, 2010; van Herk; VERHAEGH; FONTIJN, 2009). Programming by Demonstration (PdB) was the second, used by (CYPHER; SMITH, 1995; MCDANIEL; MYERS, 1997; MCDANIEL; MYERS, 1999; REA; IGARASHI; YOUNG, 2014; WOLBER, 1996; YOON; KIM, 2012). Scripting was the third, explored in (CARBONARO et al., 2008; KAUHANEN; BIDDLE, 2007; TONON; BAECKER, 2010; van Herk; VERHAEGH; FONTIJN, 2009). Finally, a single study used macros: (van Herk; VERHAEGH; FONTIJN, 2009). Parameterization was explored by (BURLESON et al., 2009; CHILTON et al., 2009; REPENNING; IOANNIDOU, 2008; YOON; KIM, 2012).

From an ET and EUD perspective, most studies (23[11]) aided on the creation of the required technology for games. The minority of the studies focused on the remaining elements: aesthetics (9 studies[12]), story (4 studies[13]), and mechanics (8 studies[14]). For game design, this is a limitation when we consider the statement of (SCHELL, 2008) that no element is more important than another.

Furthermore, we identified three main purposes for the listed tools – game creation; learning game design; and content creation. For learning design, approaches explore the principles of Constructionism (HAREL; PAPERT, 1991; KAFAI; BURKE, 2015): game creation is explored as means of learning programming. Content creation either explored modding, or was the goal of games such as Super Mario Maker[15] and Little Big Planet[16].

[11] Technology: (IOANNIDOU; REPENNING; WEBB, 2009; REPENNING; IOANNIDOU, 2006a; AHMADI; JAZAYERI, 2014; BASAWAPATNA; REPENNING; KOH, 2015; PENA, 2011; FERREIRA et al., 2012; REPENNING, 2013; REPENNING, 2011a; MOTA; FARIA; de Souza, 2012; KOH et al., 2014; JAZAYERI; AHMADI, 2011; AHMADI; JAZAYERI; REPENNING, 2012a; van Herk; VERHAEGH; FONTIJN, 2009; BURLESON et al., 2009; TONON; BAECKER, 2010; REPENNING, 2011b; LAGERSTRÖM et al., 2014; AHMADI; JAZAYERI; REPENNING, 2012b; BERLAND et al., 2011; REPENNING; AMBACH, 1997; KESER et al., 2010; IOANNIDOU; REPENNING; WEBB, 2008; PERRONE; CLARK; REPENNING, 1996)

[12] Aesthetics: (YOON; KIM, 2012; IOANNIDOU; REPENNING; WEBB, 2009; REPENNING; IOANNIDOU, 2006a; AHMADI; JAZAYERI, 2014; JAZAYERI; AHMADI, 2011; REPENNING; IOANNIDOU, 2008; AHMADI; JAZAYERI; REPENNING, 2012a; WOLBER, 1996; IOANNIDOU; REPENNING; WEBB, 2008)

[13] Story: (de Leeuw et al., 2007; MARCHIORI et al., 2012; MARCHIORI et al., 2011; CARBONARO et al., 2008)

[14] Mechanics: (REPENNING, 2006; MCDANIEL; MYERS, 1997; MCDANIEL; MYERS, 1999; CYPHER; SMITH, 1995; SMITH; CYPHER; TESLER, 2001; REA; IGARASHI; YOUNG, 2014; SMITH; GRAHAM, 2010; REPENNING; AMBACH, 1997)

[15] <http://supermariomaker.nintendo.com/>

[16] <http://littlebigplanet.playstation.com/>

Resnick & Silverman (2005) and Burke & Kafai (2014) explore a house as a metaphor to define four desirable criteria for defining EUGD tools:

Low floors Tools should be intuitive and accessible for new users.

High ceilings Tools should allow the creation of complex applications.

Wide walls Tools should allow the creation of a wide range of applications.

New windows Tools should promote developing in communities: open "new windows" to encourage discussions, help, collaboration, and learning among users.

Audiences for EUGD

Although the "low floors" criteria mentioned accessibility, no study explored accessibility for inclusion of people with disabilities. Rather, the criteria seems focused on ease of learning and use to enable new users without disabilities to create their games.

Instead (or besides) describing approaches related to EUGD, 25 studies focused on aspects related to end-user game creation. These studies explored two main contexts: 8 studies[17] focused game making as means of acquiring GL. The remaining 17 studies[18] focused on specifics of the development, including learning and motivations regarding the practice of EUGD. Most GL studies have students as their intended audience; the exception is Moumoutzis et al. (2014), which provided a framework for teachers, acknowledging the importance of ensuring they knew GL before they taught their studies.

Learning EUGD

End-users are not, necessarily, professional artists, designers, writers, and/or programmers – specially at once. Besides, they do not always have a tutor or professor to teach them. In this context, it is important to recognize common difficulties and strategies to overcome them. Studies relating to learning EUGD affirm that children learn with playing (PETRE; BLACKWELL, 2007) and collaborating (UZUNBOYLU; BAYTAK; LAND, 2010), can communication via game mechanics (MCARTHUR; TEATHER, 2015), and acquiring Computational Thinking (CT) when playing and creating (BASAWAPATNA et al., 2014). Ease of use, install, and sharing are important to game creation tools

[17] GL: (MOUMOUTZIS et al., 2014; CHILTON et al., 2009; MCARTHUR; TEATHER, 2015; LIN; CHIOU, 2010; El-Nasr; SMITH, 2006; MORAIS; GOMES; PERES, 2012; HAYES; GEE, 2010).

[18] Development process: (AHMADI; JAZAYERI; LANDONI, 2012; BASAWAPATNA et al., 2014; CHEUNG, 2011; COMUNELLO; MULARGIA, 2015; DENNER; WERNER; ORTIZ, 2012; HAYES, 2008; HONG; CHEN, 2014; MCARTHUR; TEATHER, 2015; NIEBORG; VAN, 2008; OWENS, 2011; PETRE; BLACKWELL, 2007; POOR, 2014; POSTIGO, 2008; RESNICK et al., 2009a; ROBERTSON, 2012; SOTAMAA, 2010; UZUNBOYLU; BAYTAK; LAND, 2010).

for children (PETRE; BLACKWELL, 2007). Furthermore, middle school students have difficulty to understand condition and repetition commands, the use of variables and their scope, and how to modularize their program (DENNER; WERNER; ORTIZ, 2012), and traditional programming languages are hard for children (RESNICK et al., 2009a).

Basawapatna et al. (2014) defined a *gentle slope* model of Computational Thinking (CT) acquisition: the *Consume-Create Spectrum*. Initially, the model proposes that children should consume existing technology to get familiar with them before creating their own. Thus, they start learning passively, through using, observing, and interacting with that currently exists. Next, they should start authoring their own technology, with increasing levels of complexity: animations, interactive simulations, participatory simulations, construction kits simulations, pattern based authoring, EUD, and, finally, traditional programming. The model defines a process for creating a simulation: children should start with a question, then develop a model to explain it; next, they should express their model computationally, run it, visualize results, and, if necessary, review their model.

Ahmadi, Jazayeri & Landoni (2012) defended the development of problem solving skills, employing a storytelling method for game creation. In their method, the end-user define a story (which describes the desired gameplay) for her/his game. Next, they should determine objects and their behaviors. Story creation starts with the definition of nouns and verbs to convert the story into a program. Nouns become scene objects if they perform any role on it. Verbs become behaviors to program – if possible, using a Computational Thinking Pattern (CTP).

Motivations for EUGD

Motivations for game modding included sense of control over the game, self-expression, self-satisfaction, "cultural remixing", artistic expression, cultural expression, and to improve balance and historical authenticity (POOR, 2014; SOTAMAA, 2010; OWENS, 2011; CHEUNG, 2011).

In general, collaboration and community are important elements for EUGD (UZUNBOYLU; BAYTAK; LAND, 2010; POOR, 2014; COMUNELLO; MULARGIA, 2015; HONG; CHEN, 2014). However, collaboration and community are not always positive, potentially causing competition between modders, social pressure, maintenance of reputation, and excess of work (HONG; CHEN, 2014; NIEBORG; VAN, 2008).

Demography

Students were the main intended audience for studies involving active end-users' participation; 29 studies involved students on their activities[19]. Children and adolescents

[19] Students with participation of students: (UZUNBOYLU; BAYTAK; LAND, 2010; IOANNIDOU; REPENNING; WEBB, 2009; AHMADI; JAZAYERI, 2014; PETRE; BLACKWELL, 2007; BASAWAPATNA;

of *elementary* and *high* schools in the United States of America were a common audience. For EUGD applied to education, Earp (2015) found similar results – as mentioned in Section 2.1, most studies focus on younger people.

Following students, modders were the second most common group, with 5 studies[20]. Communities of modders were more heterogeneous; for instance, (POOR, 2014) participants average age was 31 years old.

Three studies performed gender comparison for EUGD practices, stating that girls create games as good – or better – than boys (DENNER; WERNER; ORTIZ, 2012; ROBERTSON, 2012; HAYES, 2008). Hayes (2008) notices that interests are different between genders – boy usually added cheats to the game; girls, however, showed more interest on creating new characters, clothes, and items for customization.

2.2.3 Professional Game Development: Game Engine Architectures

To support broader inclusion and to enable more people to create and play digital games, we also focused our attention on architectures for game development middleware – including Application Programming Interfaces (APIs), Software Development Kits (SDKs), and game engines. In particular, we wanted flexible architectures for design and implementation of inclusive games.

In our architecture, we explore Entity-Component Systems (ECS), Event-Driven Architectures (EDA), and Data-Driven Architectures (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014). Although there are variations defining how to implement these architectures, we are more concerned with their definitions and flexibility in this chapter. In fact, we will provide our definitions in Subsection 2.5.1. The reasoning if to provide more flexibility to developers who wish to implement their own variations.

ECS decomposes games into components (groupings of data to abstract a single responsibility) and entities (collections of components). An ECS is a more extreme version of the Decorator pattern (GAMMA et al., 1994); in an ECS, components describe what an entity is able to do. When we attach a component to an entity, it becomes able to perform an activity related to the component; when we remove the component from

REPENNING; KOH, 2015; KAUHANEN; BIDDLE, 2007; DENNER; WERNER; ORTIZ, 2012; REPENNING, 2013; MORAIS; GOMES; PERES, 2012; MOTA; FARIA; de Souza, 2012; KOH et al., 2014; JAZAYERI; AHMADI, 2011; AHMADI; JAZAYERI; REPENNING, 2012a; HAYES, 2008; MCDANIEL; MYERS, 1999; CARBONARO et al., 2008; El-Nasr; SMITH, 2006; ROBERTSON, 2012; REPENNING, 2011b; LIN; CHIOU, 2010; HAYES; GEE, 2010; SMITH; CYPHER; TESLER, 2001; REA; IGARASHI; YOUNG, 2014; BERLAND et al., 2011; MCARTHUR; TEATHER, 2015; MOUMOUTZIS et al., 2014; KESER et al., 2010; IOANNIDOU; REPENNING; WEBB, 2008; PERRONE; CLARK; REPENNING, 1996).

[20] Studies involving modders: (POOR, 2014; AHMADI; JAZAYERI; LANDONI, 2012; POSTIGO, 2008; PERRONE; CLARK; REPENNING, 1996; SOTAMAA, 2010).

it, it loses the ability.

EDA allow to decouple dependencies relationships from an event (point of interest) and its processing (the event handler or event listener) (GAMMA et al., 1994). Event handlers process events once they happen (they are triggered). This allows applications to change their execution flow. Events and event handlers form many-to-many relationships. An event can have multiple event handlers, and an event handler may listen to many events.

Data-Driven Architectures allow to change the execution flow of a digital game from an external data source (for instance, a file or a database). Game engines often explore these architectures to increase their extensibility – instead of hard coding values into variables, values can be fetched from the resource and loaded at run-time. These are particularly useful when combined with Factory patterns (GAMMA et al., 1994; MCSHAFFRY; GRAHAM, 2012; GARCIA; NERIS, 2014) to construct game entities.

From game engines, we also explore the concept of input (re-)mapping (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014). This technique allows players to customize their bindings to provide input to a game. In our architecture, we use this strategy for full input (re-)mapping – both for bindings and for devices. We extended this same idea for output.

2.2.4 Computer-Supported Collaboration Work and EUD for Accessibility

As collaboration is an important element in EUGD, we researched the Computer-Supported Collaboration Work for references of people with disabilities co-creating accessibility solutions. Although they are often recipients of assistance – many times unfairly (BENNETT; BRADY; BRANHAM, 2018) –, we found success stories in (SHIRAISHI et al., 2017; BUEHLER et al., 2015; BENNETT et al., 2016; HURST; TOBIAS, 2011; PIPER et al., 2006; BENNETT; BRADY; BRANHAM, 2018). In these cases, people co-created hardware and assistive technologies, (BUEHLER et al., 2015; BENNETT et al., 2016; HURST; TOBIAS, 2011), communication for people with hearing disabilities (SHIRAISHI et al., 2017), and participatory creation of a game for social therapy for Asperger's Syndrome (PIPER et al., 2006)[21].

In particular, we found that we should consider inclusion as relations between individuals and communities, and as independence and interdependence (BENNETT; BRADY; BRANHAM, 2018; LIU; DING; GU, 2016a). Liu, Ding & Gu (2016a) highlights "communistic interaction", defined as "from each according to their abilities, to each according to their needs". This is supported by Bennett, Brady & Branham (2018), who

[21] The game was implemented by the researchers, not by the participants (end-users).

says that 'access is not only a solution to a disability-related barrier; it is a way of being together and helping one another'.

Bennett, Brady & Branham (2018), in particular, affirm that interdependence helps to notice connections between people and things. Interdependence highlights that people can receive and provide access at the same time. People with disabilities can be active contributors. In this work, we describe our efforts towards this goal – in digital games.

2.3 Towards Inclusive Creation of Tailorable Games

Although there are approaches enabling end-users to create their own games, they intended audiences are still limited. We should bring diversity and inclusion to the house metaphor of Resnick & Silverman (2005) and Burke & Kafai (2014) (Section 2.2.2). To promote games inclusive creation and use, we should aim to create opportunities for collaboration, removing entry barriers in related activities.

The literature currently addresses people without disabilities acting as non-professional programmers. Thus, the main barriers for the audience were technical – to enable more people, we should reduce development complexity. Yet, although technology is a fundamental part of digital games, technical barriers are only part of whole. When we consider further diversity, a non-exhaustive enumeration of barriers related to game creation and play could include:

1. Technical barriers:
    - Programming knowledge;
    - Game design knowledge;
    - Academic knowledge (mathematics, physics...);
    - ICTs proficiency;
    - Language knowledge (especially for non-English speakers).
2. Physical and sensory barriers;
3. Cognitive barriers;
4. Emotional barriers;
5. Cultural barriers;
6. Required software/hardware barriers;
7. Socioeconomic barriers.

We can, thus, infer that overcoming barriers is a multi-domain endeavor, requiring knowledge and skills in multiple disciplines beyond Computer Science and Game Design. For a single person, it may be impossible to overcome all, for every conceivable scenario. However, as in traditional EUGD, communities could be a key factor for success.

We argue that effective communication and participation could leverage individual abilities and knowledge to collective ones. In other words, we could explore strengths of individual members to, together, provide interaction alternatives benefiting the community.

Although the size of community may vary (friends, families, schools, hospitals, Internet forums, organizations), the principle is the same. Every single member has her/his own interaction needs. Interests, abilities, experiences, knowledge, and interaction needs of each member will vary. If we embraced these needs, people could help each other. Instead of exclusion by particular skills, people could provide their abilities to help themselves and their peers – inclusion as a collaborative effort.

People co-design a game, and provide interaction alternatives aimed to fulfill accessibility gaps in the system. Instead of "able to play" and "unable to play", people are either "included" or yet "to be included" into a game. A goal of the community is, thus, to alternatives to enable inclusion – inviting people from "to be included" to the "included" one. Inclusion stops being sole responsibility of game developers. Instead, developers could further provide modification tools letting end-users (players) to define their own customization for interaction features. Analogously to traditional modding, it could result into "accessibility modding" towards inclusion.

From our point of view, inclusion has a cascading effect, as included people may, subsequently, help others to create and play. In this sense, people are empowered by their abilities. We should match their abilities to opportunities to contribute. In a way, this is similar to building jigsaw puzzles. People fit pieces to fill gaps, forming a bigger picture in the process. Gaps are the currently missing interaction alternatives that could enable someone to play. The pieces are software (and/or hardware, and/or media) constructs to provide interaction alternatives[22]. More importantly, this means that, with suitable approaches, everyone could contribute. People with disabilities could become contributors, as they can share knowledge, expertise, and their skills to propose – and to create – enhancements to the game. They could become empowered, active creators.

To achieve this perspective, it is necessary to enable end-users to contribute towards accessibility. First, digital game creation should expand beyond creating software; technology is only a part of digital game. Combining the four elements (mechanics, aesthetics, story, and technology) of the ET with the extensive disciplines that can con-

[22] As they are software, people can always create new pieces to fit gaps.

tribute to game design, we can expand game creation approaches to suit abilities of creators, focusing on abilities and their goals. Thus, people with heterogeneous interaction needs can work together to co-create multimodal content for stories, art, mechanics – we should change questions such as "can we play the game?" into "how can I make the game work for us?"

2.4 A Framework Towards Tailorable Game Creation: End-User Collaboration to Co-Create and Play Digital Games

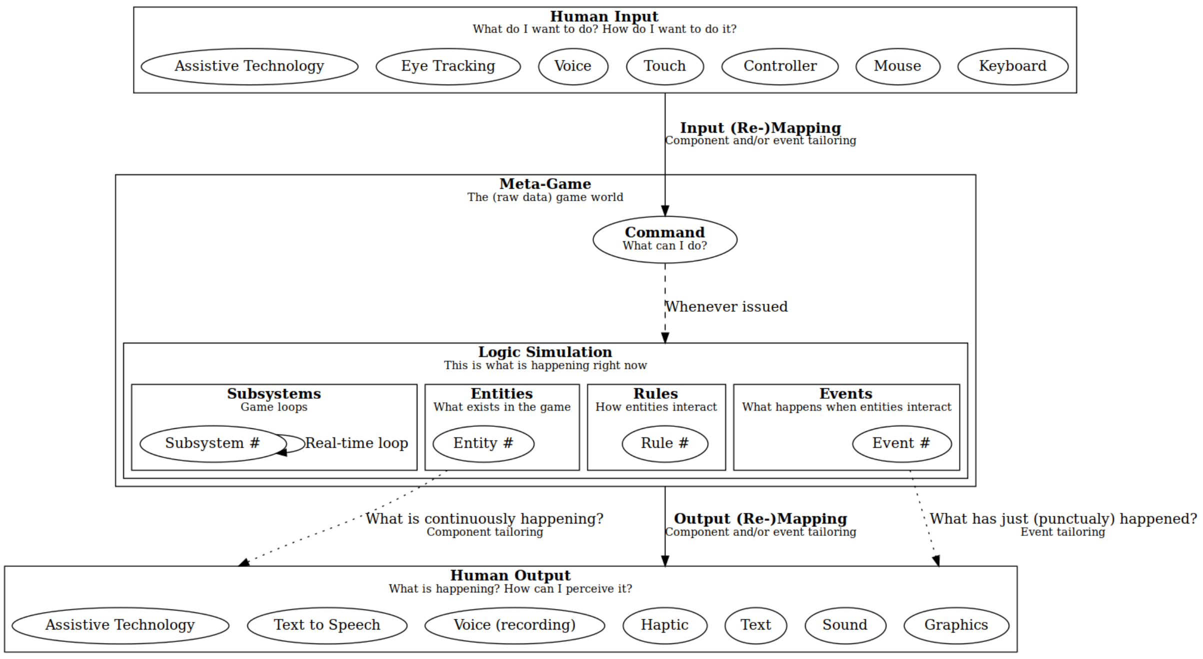
Due to multi domain barriers – a task requiring technological and human efforts from people with multiple backgrounds –, we knew that there would not be a single solution – a single tool, a single approach, a "silver bullet" – to more people to create digital games[23]. Rather, to support the diversity, we needed a combination of approaches, a collection of practices and alternatives. The Cambridge Advanced Learner's Dictionary & Thesaurus (2017) define the noun framework as "[C] *a supporting structure around which something can be built;* [C2] *a system of rules, ideas, or beliefs that is used to plan or decide something*". To support diversity, thus, we needed a framework on its broader term – a structure composed of tools, techniques, methods, strategies, guidelines, practices, and systems used both to design as well to implement software.

To define our framework, we benefited from the EUGD (Section 2.2) and game accessibility literature, which describes what we could offer in resulting games. Previously ((GARCIA, 2014; GARCIA; NERIS, 2013a; GARCIA; NERIS, 2014).), we had defined a game engine based on game accessibility guidelines, the study from Yuan, Folmer & Harris (2011) and using UA-Games as references. Our result was a way to implement games abstractly, without any reference to human-related IO. The engine allowed us to implement Meta-Games without human interaction, allowing us to define multiple interactive Games to suit the needs of players at use-time. During use-time, the Meta-Game could become multiple human-playable Games with re-attachable IO features. With this possibility, we could re-introduce players to suitable version of the same logical implementation base game (Meta-Game) tailored to their own interaction needs.

To refine our previous solution, we extracted the tailoring features of the engine, simplifying its complexity. We tried to define an architecture with few elements, high flexibility, easy to implement operations, and providing maximum reuse of existing features. Our goal was to enable to design and implement digital games for adaptation; we implement for no one, to include everyone. As a result, the architecture enabled us to

[23] Point in case, visual programming languages are, by definition, inaccessible to blind users; written programming is inaccessible to illiterate people; spoken programming is inaccessible for mute people. . .

Figure 1 – Run-time tailoring: full interaction (re-)mapping.



Source – Created by the author.

Note – As every IO element can change, the task of the developer is to map what is happening into the game simulation (the Meta-Game) to the abilities of the player. If there exists, at least, one possible mapping via any existing IO technology, we can enable play. Interaction, is, thus, composed to the Meta-Game. Furthermore, as IO is decoupled, we can always introduce aesthetic features to make the game more fun – aesthetics are not part of the Meta-Game; therefore, they can be inserted at will. Similarly, we can opt to introduce only subsets of interaction features to avoid stimuli overload. This way, we can define custom tailored accessible versions of a game per public, or enable a user to define her/his own from available alternatives.

implement games without any references to human-related IO, defining (and re-defining whenever needed) particular interactions during use-time (run-time) (Figure 1). It was an interactive Meta-Game (MG) without user – we abstracted the user from the game, to become able to provide an accessible game to suit her/his needs[24]. The next step was defining tools to create games implementing the architecture. To achieve creation inclusion, we explored the architecture to implement the creation tool. This resulted into the second pillar of the framework – an inclusive game creation platform for tailorable games.

Our goal with the platform was enabling people who were currently unable to play to both *create and play*. The platform aimed to support their abilities and play to their strengths. In particular, end-users with disabilities and/or low literacy may never had interacted with computer and digital games before. Therefore, we needed to guide and support creation activities beyond tools. Moreover, we acknowledged that, although

[24] Once again, this does not mean that every game can become universally accessible. It does mean, however, that every game can become more accessible.

no single person would possess skills to make games for everyone, a community could work together to achieve greater inclusion. This defined the third pillar of the framework – a collaborative work model to scaffold game development practices, aiming to promote inclusive creation of inclusive games.

The collaborative work model explores communication and collaboration as strategies to enable people with, potentially, heterogeneous interaction needs to create and play games. In special, it defines transient roles according to what someone is doing at a given time. Besides providing more opportunities for collaborating, the roles promote a transfer of knowledge (or abilities) from people to people, aiming to allow creators to surpass barriers[25]. Degrees of assistance may differ on purpose and need. For instance, the creator might need temporary support, due to anxiety or depression, or permanent aid, due to cognitive or motor disabilities. For these cases, one person may help another to allow the creator to overcome barriers.

As with the ET for Game Design, pillars of the framework support each other to empower people to make their own games, sharing their creations with their friends, enabling people to play. Our rationale is that interaction requirements come from people, emerging from the diversity. The diversity can also work together, as communities, to provide solutions to enable creation and use. Thus, we can support end-user created inclusion if we enable and support this workflow.

The architecture fully decouples logic from interaction. As a result, interaction can be arbitrarily introduced to (or removed from) digital games, at use-time. In particular, developers can consider interaction as extensions – add-ons – to the logical system, aiming to enable use and (re-)define interaction when people are using the system. This means that a user can choose how to interact with a system based on how she/he combines existing interaction alternatives to command and perceive a game.

As digital games are software, interaction alternatives (as well as their required software and hardware interfaces) have to be implemented. Traditionally, this implementation is a task for the developers; with EUD, we enable more non-developers to contribute. The second requirement is, thus, enabling end-users to create game content *as well* as interaction features. With game creation (or modification) tools, end-users may provide new ways to interact with the logical system – both to command it (with interaction alternatives for input) and to perceive it (with interaction alternatives for output).

[25] In parallel to the framework, we worked with non-programmers in educational and healthcare contexts, exploring games in therapeutic and rehabilitation practices (RODRIGUES et al., 2014; RODRIGUES et al., 2015). A related result was a visual language to help end-users design their own therapeutic games (GARCIA; RODRIGUES; NERIS, 2016). In these studies, we observed that game creation can contribute to learning and rehabilitation. Moreover, domain expert from these areas (professors and healthcare professional for the last example) may assist others playing games.

The third requirement is benefiting from the diversity – combining EUD to communistic interactions to define communities for "accessibility modding". As communities, every person able to create can, potentially, contribute to improve a project based on her/his own abilities, skills, and knowledge. Multiple people can contribute; one person may contribute multiple times. Improvements add to each other; a contribution may enable someone new to play and create, as well as enhancing the overall playing experience and interaction quality. This means that inclusion towards accessibility and usability can be an iterative, mutualistic[26], communistic, and collaborative process.

Alone, each pillar defines activities to allow the desired outcomes. As a whole, the framework contributes to the three levels of tailoring (customization, integration, and extension) proposed by Mørch (1997). At use level, players can customize and integrate a MG to create their Game without coding – they select available interaction alternatives to define how they shall interact with the software. With the collaborative work model and the tools, they can reach the extension level.

Considering the definition of EUD and MD (from Subsection 2.2.2), the workflow defined by the framework enables end-users to act as non-professional designers and developers to collaborate iteratively towards inclusion. Together, the pillars of the framework enable the desired workflow. The community provides interaction alternatives (as add-ons) exploring EUD practices (in tools) to practice "accessibility modding" towards improving inclusion. The MG allows for exploring MD and EUD to tailor user interaction, transforming an abstract system into potentially accessible Games to a player. If the game is not yet accessible, the community may work together (according the abilities of its members) to provide new interaction alternatives aiming to include the player. Once a player is included, she/he can, potentially, contribute in the future to include new players – defining iterative cycles of inclusion.

We describe each pillar in the next sections.

2.5 The Architecture: Tailoring the Interaction to Individual Needs in Use- and Run-Time

We formalized a software architecture as a pillar of the framework[27]. For greater potential of inclusion, the architecture decouples interaction from logic. This way, our only limitations for inclusion are current available technologies, and what is possible to implement software-wise.

As discussed in Section 2.4, we previously defined a game engine to support run-

[26] As defined in Biology, mutualism defines a relationship in which organisms benefit from the existence of each other.

[27] Appendix B further details the architecture

time IO tailoring of digital games (GARCIA; NERIS, 2014; GARCIA, 2014). We defined the engine based on the Interaction Model and high and low-level strategies for game accessibility described by Yuan, Folmer & Harris (2011). We tried to support as many strategies at engine level as possible. As a result, we ended with fully re-configurable game interaction, which we can explore for accessibility purposes. In particular, EDA and EDP, we can introduce input automation features to games with ease.

As the (re-)definition of IO interactions at run-time was more useful than the engine itself, we decided to generalize the solution to self-contained definitions and a corresponding architecture. This way, we could implement it into existing game creation platforms to reduce development efforts – and to benefit from exists communities. For brevity, in this section we provide an overview of the architecture.

2.5.1 Run-Time Tailorability for Games: High-Level Concepts

For greater flexibility, we needed an architecture able to provide run-time tailoring to software. For simplicity and a more generic solution, we tried to create the architecture using features provided by mainstream programming languages. The reasoning was that, with a generic solution with simple programming constructs, we would be able to explore the architecture within existing game development middleware – including APIs, SDKs, and game engines. With these requirements, we combined Entity–Component Systems (ECS), Event Driven architectures (EDA), and Data-Driven (DD) architectures to define our (MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014). These are also architectures commonly used by the industry and can be defined in software, if the underlying middleware do not provide them.

Entity-Component Systems (ECS) provided the concepts of entity and component. Event Driven architectures provided events and event handlers. Data-Driven architectures allowed us to explore external data resources to change run-time behaviors of systems. To ensure the desired separation between logic and implementation, components and event handlers were split into three mutually exclusive groups: logic, input, and output. From these conceptions, we proposed the following definitions[28]:

Component. A component is a well-defined data set that has a single, well-defined purpose for a digital game.

Components are divided in three disjoint groups according to their purposes: input, output, and logic.

Entity. An entity is a finite set (collection) of components.

Event. An event is any happening of interest in a digital game.

[28] These are a simplification of the original concepts defined by Garcia (2014).

Event Handler. A procedure that process events.

Event Handlers are divided in three disjoint groups according to their purposes: input, output, and logic.

Subsystem. A subsystem process a finite and arbitrary subset of all the existing components according to rules defined for the game.

Abstract [Component, Entity, Event Handler, Subsystem]. An element without references to user related (human) IO. It is, therefore, logic only.

Concrete [Component, Entity, Event Handler, Subsystem]. An element which may have references to user related (human) IO. Besides logic, it can have IO.

Rule. A rule is a relation among game entities which results into an event.

Combinations of rules can represent mechanics, small parts of a bigger mechanics, or other functionalities for systems. Their first goal is to provide an event to avoid implicit information that should be conveyed to players. Their second goal is to enable developers to decouple how to handle interaction the implementation. This way, rules semantically marked the code to, at a later time, enable finer IO processing as necessary (we describe some possibilities later, in Subsection 2.5.5).

Agent. An agent is a concrete entity with, at least, one input component to become able of interacting with the system.

Agents can be human or Artificial Intelligence (AI) controlled. At the one hand, this means that we can change how controls an entity at any time – this can provide automation for player who need assistance to provide input. At the other hand, this means that the same algorithms that can control Non-Playable Characters (NPCs) can provide partial automation to help the player.

Command. A command is an event triggered with the purpose of providing input to enable interaction between an agent and the game.

Agents use command to interact with the game; this decouples how the players interact with the game. Commands, thus, define what players are able to do.

Meta-Game (MG). An MG is a template to create games with the same elements and rules. It is defined from a combination of events, commands, rules, and abstract elements (components, subsystems, entities, and event handlers). A game simulation emerges from the combination of the previous elements. As it is abstract, it does not have references to human-related IO. Communication between entities occur through commands.

Thus, an MG is a system without human-related interaction. The system is already implemented; the data structures (entities and components) and important moments (events) to convey data are there, as well as the ways to interact with it (commands). An MG is like a template to create clones of a very same game. As it is free of user-related IO, it can be specialized to allow any desired interactions. Such specialization occurs without effecting how the logic is processed.

The MG can be described by a set of abstract components (C_{Meta}), a set of events (V), a set of rules (R), a set of abstract subsystems (S_{Meta}), a set of commands (A), a set of abstract entities (E_{Meta}), and a set of abstract event handlers (H_{Meta}) as follows:

$$MG = \{C_{Meta}, V_{Meta}, R_{Meta}, S_{Meta}, A_{Meta}, E_{Meta}, H_{Meta}\} \tag{2.1}$$

The MG abstracts human-interaction with commands. As we design and implement MGs for no one, we do not exclude any audience by design. Rather, we design and implement software *for* adaptation.

Interaction Profile. An external resource which stores data describing how to convert abstract into concrete elements – that is, how to (potentially) introduce arbitrary IO to a Meta-Game.

Game. A Game is the result of transforming abstract elements from a Meta-Game into concrete ones, generating a software artifact with human-related IO. Therefore, it is, potentially, a specialization of the MG considering interaction needs of a user.

A Game is defined as an extension of an MG. It explores an Interaction profile to introduce concrete elements (components, entities, subsystems, and event handlers) to its base MG. A Game is a possible interactive system resulting from adding features to enable people to interact with an MG. This way, interaction becomes similar to a plug-in (add-on or extension) to define how a player should command and perceive her/his Game.

A Game can be described by a set of concrete components (C_{Game}), a set of events (V_{Game}), a set of rules (R_{Game}), a set of concrete subsystems (S_{Game}), a set of commands (A_{Game}), a set of concrete entities (E_{Game}), and a set of concrete event handlers (H_{Game}), and an Interaction Profile (P_{Game}) as follows:

$$G = \{C_{Game}, V_{Game}, R_{Game}, S_{Game}, A_{Game}, E_{Game}, H_{Game}, P_{Game}\} \tag{2.2}$$

For convenience, we can consider C_{Game}, S_{Game}, and H_{Game} as all existing components, subsystems, and event handlers.

> The Interaction Profile describes what human-related features will be present in the game. All elements sets are supersets of the corresponding MG's ones – this means that a Game extends the MG to provide interaction features to enable use. The idea is that we can attach IO elements to transform abstract elements into interactive ones. Whenever we detach the IO elements, we revert the Game back to an MG – and are able to completely (re-)define it once again.

The difference between abstract and concrete elements is of particular importance for this chapter; they refer, respectively, to absence or (possible) presence of human-related IO. Hereafter, whenever we use the term "abstract" before an element (component, event handler, or subsystem), it implies that the element must not have any human-related IO. Analogously, the term "concrete" implies that the element may have human-related IO. Considering the groups, abstract elements are contained in the logic group; concrete elements belongs to either the logic, output, and input groups.

Abstract elements define semantics – what entities are able to do, how they react to commands and rules, what happens as a result from a rule. This means that, although there is an interactive game[29] in MG, it is not playable by people. Rather, people interact with Games. Concrete elements determine arbitrary mechanisms to convert semantics into sensory information – how a user sees, hears, feels, touches, and smells the system, as well as provide input to it – and/or receive commands from input devices.

In the architecture, components are composed/aggregated to entities, and event handlers are registered to events. As a result, they can be inserted or removed arbitrarily – or never be used at all. This makes accessibility features into plug-and-play solutions. Developers can create their traditional games and define optimum presets for specific audiences. Players may choose to use the ones that they wish and ignore the others. The Interaction Profile stores these decisions; they can also store presets for audiences. This way, every player may define her/his own game based on her/his abilities, needs, and preferences.

Logic elements define the raw data model of the system that, when processed, defines the simulation. Logic components are the parameters for the simulation, the settings on which the logic subsystems and logic event handlers operate. For instance, Position components abstract data about position; Kinematic components abstract velocities and accelerations; Collision components abstract collision shapes, areas and related data[30]. A spaceship able to exist in the world, move, and collide would be logically

[29] Hereafter, we will use the term "Game", with upper case initial letter, to distinguish regular games from our definition presented in this section.

[30] Suffixing components with "able" is a helpful way to name (and think about) components. Once we attach a component to an entity, the latter becomes able to do whatever the component abstracts. For instance, the previous components could be renamed to "Positionable", "Collidable", "Kinematic-able".

defined by these three components. Logic subsystems and event handlers define how to process this data to define the game simulation.

Output elements provide meaning to the raw data existing in logic elements. They define how the player should perceive the content to understand it using output devices. Images, sounds, text, and haptic stimuli are possible implementations to convey data as sensory information – as well as other assistive technologies, once we implement support for them. For the spaceship example, we can sensory perceive its existence in a game world after we add components to the entity (or register event handlers for events related to the spaceship). Once we attach an image to the spaceship entity, it is graphically represented into a screen as a Game. Likewise, once we register an event handler to an event marking its movement, it reproduces sounds in speakers whenever it moves.

Input elements enable controlling the raw data. They map how the player commands entities to control the simulation with input devices. Traditional controllers (such as mice, keyboard, and controllers), assistive technology, and even algorithms can be used for command, once we implement support for them and add suitable input elements. Similarly to output elements, once we add input elements to entities or register them to events, we can arbitrarily control a Game entity.

2.5.2 Introductory Example: A Clone of Access Invaders Implemented with the Architecture

We will explore a reference implementation to illustrate the architecture before discussing it. As the architecture is for implementation rather than design, we will adopt the game design of Access Invaders (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009) to create a clone of it using the architecture[31]. Figure 2 illustrated the Unified Design for the game.

Access Invaders' Meta-Game

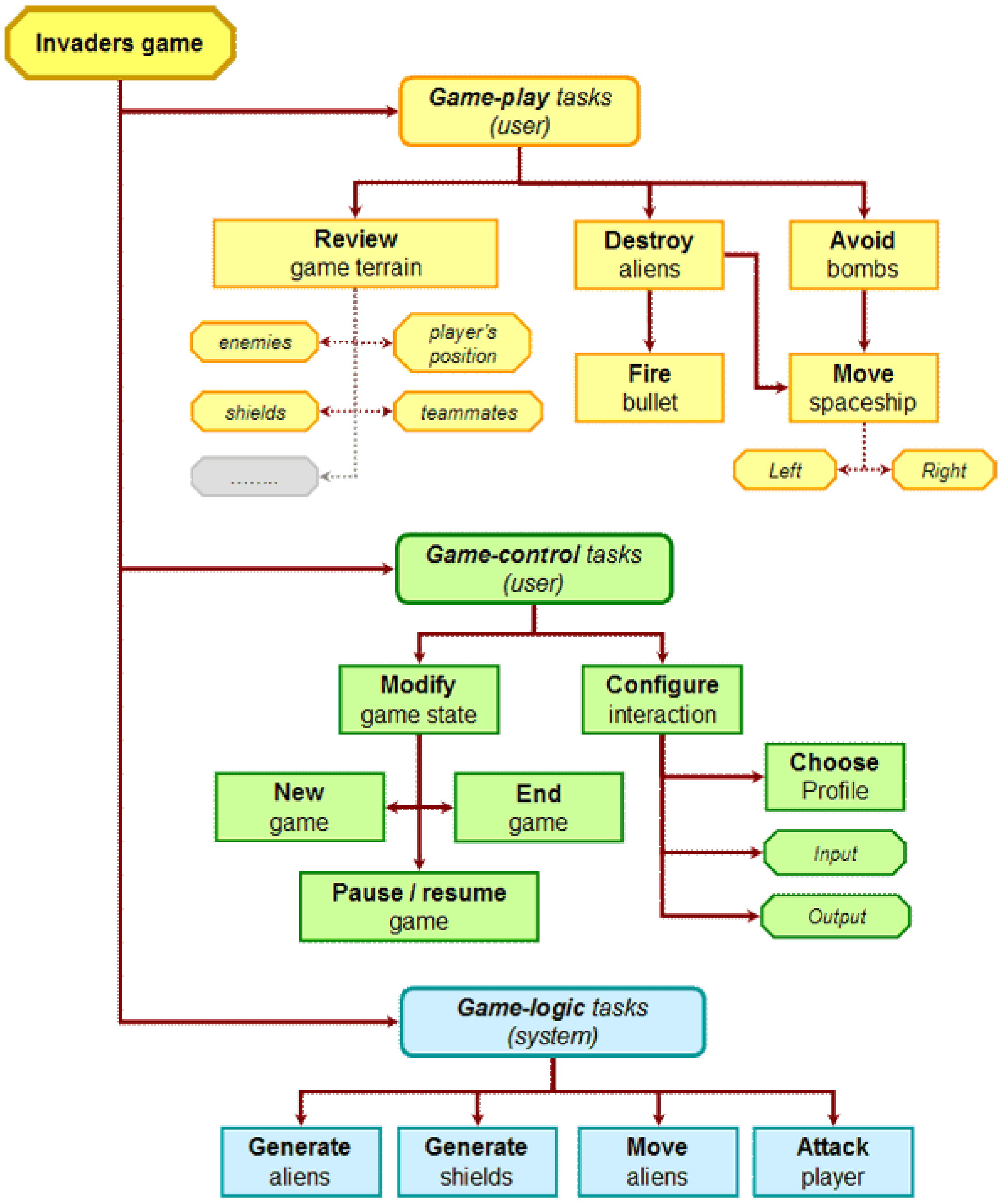
After designing the game using the Unified Design, we can map tasks into elements of the architecture to plan the implementation. The first step to plan a project using the architecture is determining the and events of the game. Analyzing Figure 2, we can identify the following entities:

E1 *Aliens*;

E2 *Bombs*;

[31] Available at <https://gitlab.com/francogarcia/GamesRunTimeTailorability>.

Figure 2 – Unified Design for Access Invaders.



Source – Extracted from (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007).

E3 *Bullets*;

E4 *Spaceship*;

E5 *Shields*;

E6 *Terrain*.

Similarly, the events (explicit and inferred) in Figure 2 are:

V1 *Alien Moved*;

V2 *Alien Spawned* (inferred from “Generate aliens”);

V3 *Alien Destroyed*;

V4 *Alien Hit* (inferred from “Destroy aliens”);

V5 *Alien Fired*;

V6 *Spaceship Moved*;

V7 *Spaceship Spawned*;

V8 *Spaceship Destroyed*;

V9 *Spaceship Hit*;

V10 *Spaceship Fired*;

V11 *Bullet Spawned*;

V12 *Bullet Destroyed*;

V13 *Shied Spawned*;

V14 *Shield Destroyed*;

V15 *Shield Hit*.

Next, we analyze the role of each entity to define their components. To identify components, we should determine *what* entities are able to do in the game (for instance, what they can do, react to, suffer from, and interact with). Events provide one way to think about their abilities. From the list of events, we can infer creation and destruction (based on health and damage), movement (based on position and velocity), and collision (based on position and shape) are logic functionalities for the system. In the architecture, these requirements become logic components (to store the data), and logic subsystems and/or event handlers (to process the data).

Events further reveal input for the system. In the architecture, input becomes commands. In this case, *aliens* and *spaceship* are able to fire projectiles (*bombs* and *bullets*). Similarly, the *spaceship* may move to the right and to the left, revealing a bi-dimensional control scheme. Those, are, thus, the commands for the Meta-Game. In the architecture, we abstract those as input components (to store data related to commands, if needed), and input subsystems and/or event handlers (to abstract an input as an event).

From these inferences,

E1 *Aliens* are logical compositions of Position, Collision, and Health components;

E2 *Bombs* are logical compositions of Position, Collision, and Damage components;

E3 *Bullets* are logically similar to *Bombs*;

E4 *Spaceship* is logically similar to *Aliens*;

E5 *Shields* are compositions of Position, Collision, and Health components;

E6 *Terrain* may be purely aesthetic in simple implementations (for instance, it acts a background). In this case, it does not affect the simulation; therefore, it is not part of the Meta-Game, as it does not affect the game logic[32].

After entities are defined, we can think about rules. Rules are abstract interactions among entities. This means that we should consider how each entity interact with each other entity in the game world. In Access Invaders, we can list, for instance, the following rules:

R1 Create *Alien* $\mapsto$ *Alien Spawned*;

R2 *Alien* Collision *Bullet* $\mapsto$ *Alien Hit*;

R3 *Alien* Collision *Bullet* $\mapsto$ *Alien Destroyed*;

R4 *Alien* Command Fire *Bomb* $\mapsto$ *Alien Fired*;

R5 *Alien* Command Move *Alien* $\mapsto$ *Alien Moved*;

R6 Create *Spaceship* $\mapsto$ *Spaceship Spawned*;

R7 *Spaceship* Collision *Bomb* $\mapsto$ *Spaceship Hit*;

R8 *Spaceship* Collision *Bomb* $\mapsto$ *Spaceship Destroyed*;

R9 *Spaceship* Command Fire *Bullet* $\mapsto$ *Spaceship Fired*;

R10 *Spaceship* Command Move *Spaceship* $\mapsto$ *Spaceship Moved*;

R11 *Shield* Collision *Bomb* $\mapsto$ *Shield Hit*;

R12 *Shield* Collision *Bomb* $\mapsto$ *Shield Destroyed*.

[32] This does not imply that aesthetics are not important; rather, they do not contribute to game logic.

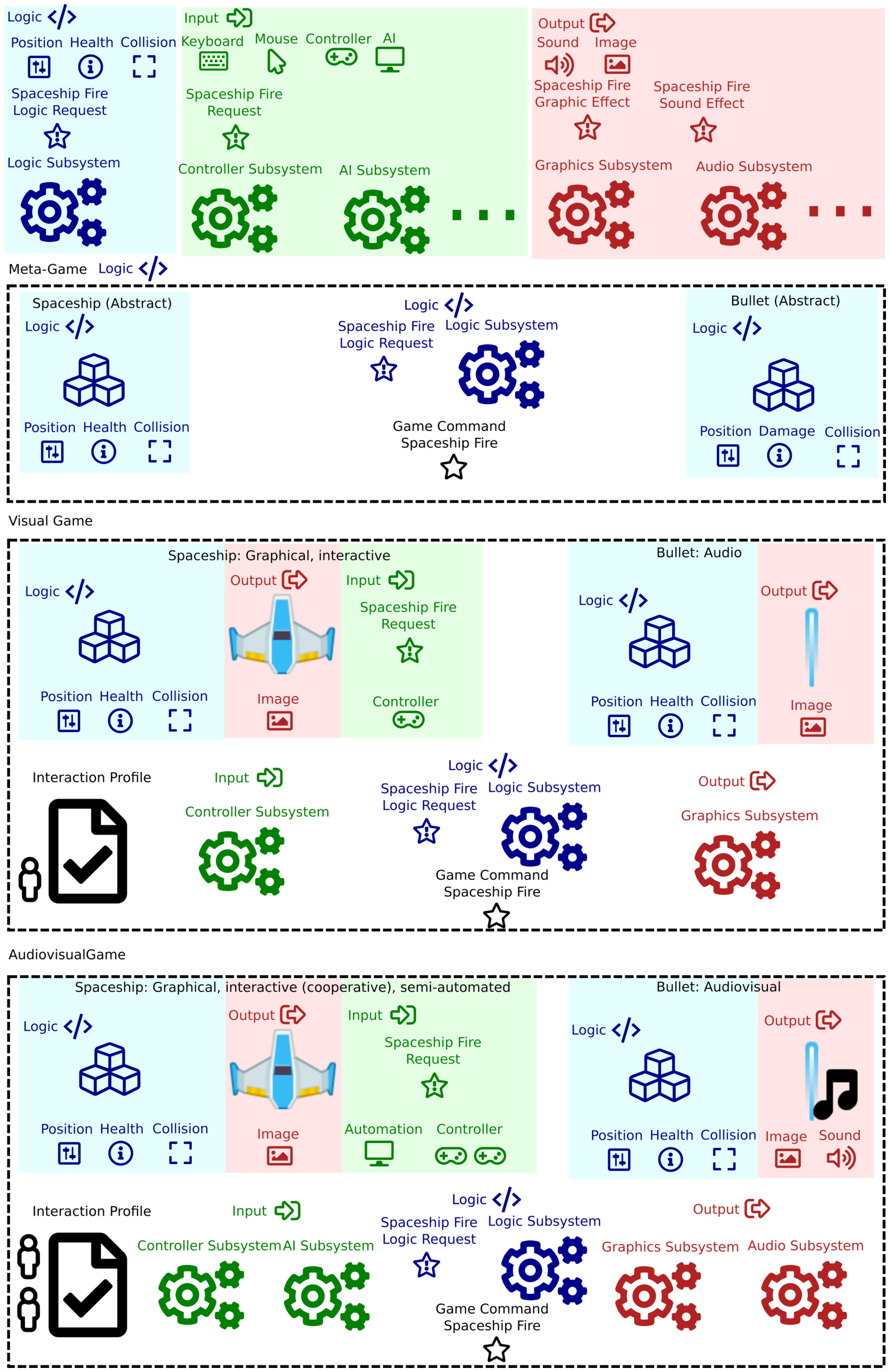
Finally, we can define subsystems and event handlers to process components and implement rules. For the given example, we need to calculate the position of *bullets* and *bombs* on each game update. Similarly, we need to track and treat collision detection and energy. After implementing the remaining rules, components, subsystems, and event handlers, we conclude the Meta-Game. The result is a system without a player. In practice, it is a non-interaction game that no one can play, as there is no human-related IO. Therefore, to create interactive Games, we need to attach user related IO to a Meta-Game: games are for humans, no for machines.

Access Invaders' Games

Once we have the Meta-Game, we can create Games to the Players (this subsection will be expanded in Subsection 2.5.4). As the Meta-Game have not made any assumption about *how* the interaction should happen, we are free to define any IO scheme at is currently allowed by existing hardware, and that is computationally possible by current implementation techniques. Therefore, we are only limited to what is currently possible to do with computers, and our abilities to translate data into information and human input into Meta-Game commands.

The strategy is to tailor the elements of the Meta-Game to suit the interaction needs of a player, (re-)mapping interactions according to her/his abilities and/or preferences (as suggested in Figure 1). For input, we should provide input elements to enable her/him to control her/his Game. For output, we should materialize raw data from components into suitable representations to convey what is happening to the player. For immediate feedback and implicit events, we can introduce event handlers to convey information whenever something important has just happened in the simulation.

Figure 3 – Tailoring elements from a Meta-Game to create two different Games.

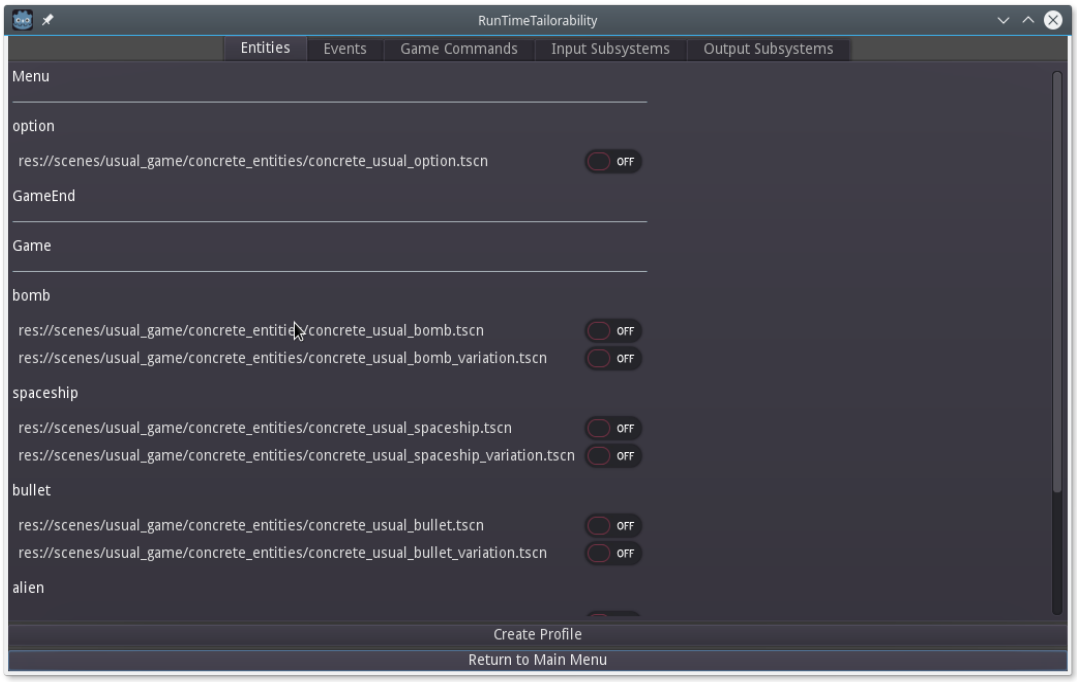

Logic
Position Health Collision
Spaceship Fire Logic Request
Logic Subsystem
Input
Keyboard Mouse Controller AI
Spaceship Fire Request
Controller Subsystem
AI Subsystem
Output
Sound Image
Spaceship Fire Graphic Effect
Spaceship Fire Sound Effect
Graphics Subsystem
Audio Subsystem
Meta-Game Logic
Spaceship (Abstract)
Game Command Spaceship Fire
Bullet (Abstract)
Position Damage Collision
Visual Game
Spaceship: Graphical, interactive
Image
Controller
Bullet: Audio
Interaction Profile
AudiovisualGame
Spaceship: Graphical, interactive (cooperative), semi-automated
Automation
Bullet: Audiovisual

Source – Created by the author.

Note – Components define the behaviors of entities. Entities with output components are represented continuously. Entities with input components receive commands. As we can attach more than one input component to an entity, we can provide partial automated assistance to players. Another possibility is enabling multiple players to control a same entity, for situations when automation is not possible.

Figure 4 – Interaction profiles creating different Games from the same Meta-Game in a same play session.

(a) Interaction profile creator.



RunTimeTailorability

Entities | Events | Game Commands | Input Subsystems | Output Subsystems

Menu

option

res://scenes/usual_game/concrete_entities/concrete_usual_option.tscn OFF

GameEnd

Game

bomb

res://scenes/usual_game/concrete_entities/concrete_usual_bomb.tscn OFF

res://scenes/usual_game/concrete_entities/concrete_usual_bomb_variation.tscn OFF

spaceship

res://scenes/usual_game/concrete_entities/concrete_usual_spaceship.tscn OFF

res://scenes/usual_game/concrete_entities/concrete_usual_spaceship_variation.tscn OFF

bullet

res://scenes/usual_game/concrete_entities/concrete_usual_bullet.tscn OFF

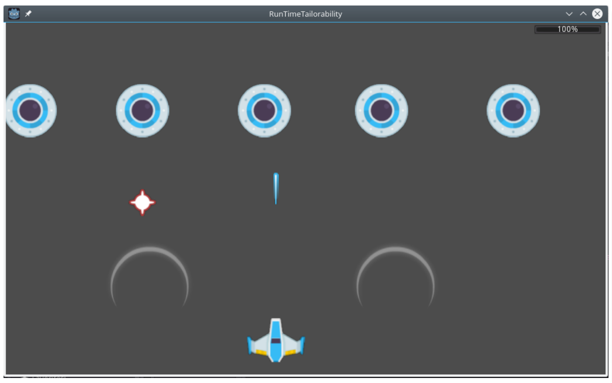
res://scenes/usual_game/concrete_entities/concrete_usual_bullet_variation.tscn OFF

alien

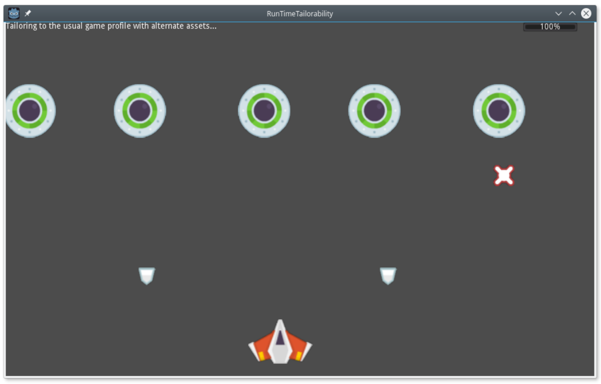
Create Profile

Return to Main Menu

(b) Default version.

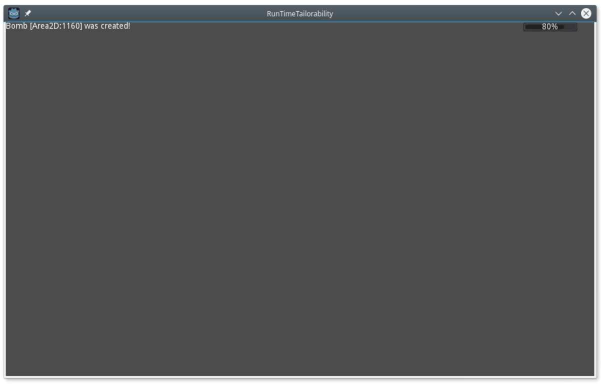


(c) Alternative art (with controller input).



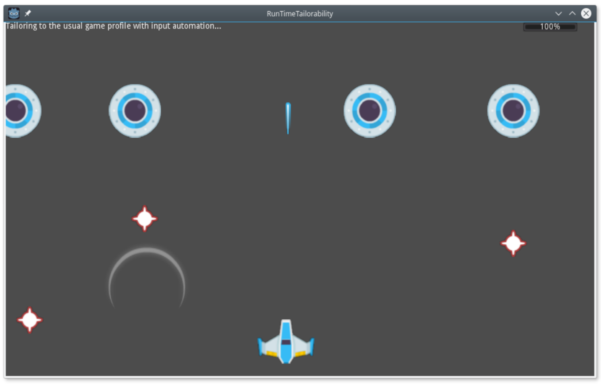
Tailoring to the usual game profile with alternate assets...

(d) Audio only (with text descriptions).



Bomb [Area2D:1160] was created!

(e) Input automation (auto-fire, manual movement).



Tailoring to the usual game profile with input automation...

Source – Created by the author.

Note – As we are composing interaction, we can fully change IO for the game. Although the images are similar in this example, we could define something completely different. If we can implement it with the elements of the architecture, we are able to introduce them at run-time.

As interaction is composed, we can define a potentially infinite number of different Games sharing the same rules. However, each game can be played different. Changes could be as simple as alternate images from graphical components to convey the content slightly different, or as sophisticated as defining a voice-controlled audio only game or simplified graphics with semi-automated input. It only depends on IO elements that we imagine, implement, and introduce at run-time exploring an Interaction Profile.

Figure 3 and Figure 4 illustrate the idea for the clone of Access Invaders. The Meta-Game could become a graphics-only Game (with graphical components, subsystems, and event handlers to represent content as images) controlled with a game controller. It could become an audio-only Game (audio components, subsystems, and event handlers to represent content with audio) controlled with a keyboard and partial automation. It could become an audiovisual game on which two or more players control the same Spaceship to help each other, sharing part of the input. It could explore assistive technologies for IO.

Provided that we can implement the elements, design interfaces to transform data into information to convey meaning with output devices, map input devices to commands, we can define new ways to play[33]. Once we combine elements and attach them to the Meta-Game, it results into a Game. Furthermore, we can combine any existing interaction alternative to define custom Games. New interaction alternatives may enable people to play, as well as provide more options to improve the experience for those who could already play.

2.5.3 Games for Machines: Implementing Meta-Games, Simulations Without (Human) Players

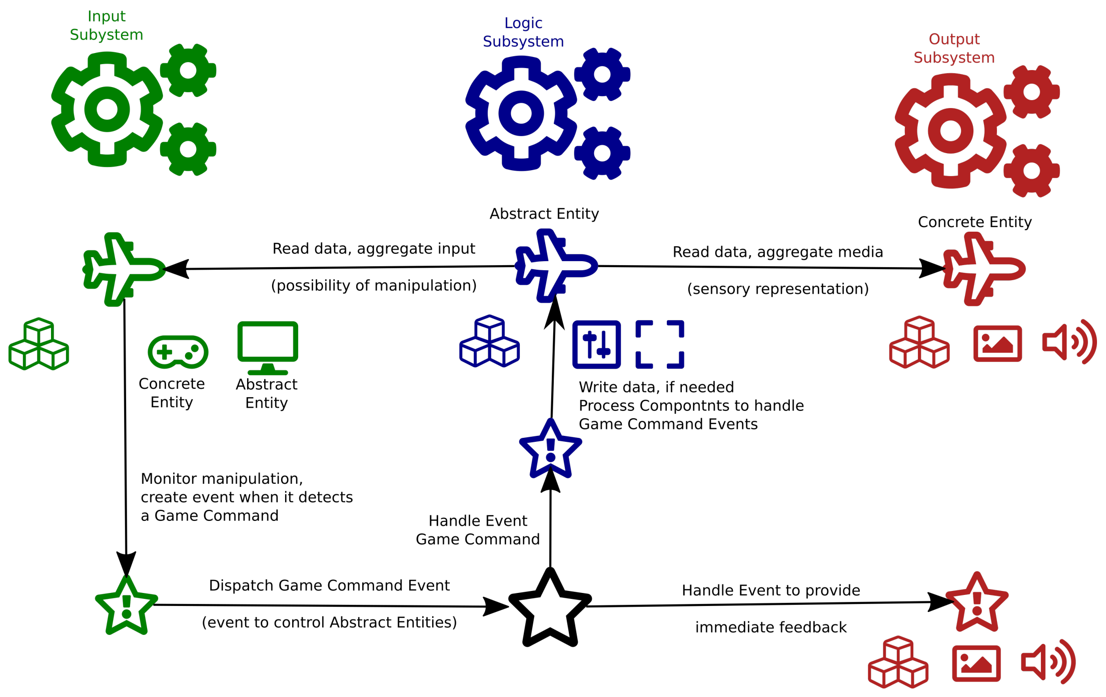
As a MG has no human players (it is abstract), it explores programming mechanisms to abstract input as input semantics, allowing logical constructs to process and react to them accordingly[34]. Event-driven architectures provide a way to implement such mechanisms.

A command, implement as an event, is able to simulate input, decoupling system logic from human IO handling. The MG not poll nor wait updates of input devices (in fact, the MG has no concept of input device), because it not a physical level interaction that changes its logical state. Instead, an MG reacts to intents expressed from commands. An MG is a purely logical construct, on which logic data fully simulates a game world.

[33] It is important to note that every new accessible version may require significant development efforts (for design, implementation, and evaluation).

[34] We omit output from the MG discussion, as it is irrelevant if there are no humans. A machine is not concerned if it emits light, makes sounds, vibrates, or smells. For a system, the current state is determined from its internal data. A more functional way of understanding is considering that, every step of the game loop, the previous internal state is an input to the current one.

Figure 5 – Data flow of the architecture.



Source – Created by the author.

Note – An MG defines the entire game simulation from abstract elements. Game commands simulate input, abstracting user interaction (and enabling the creation of AI Agents). Human-related IO become unnecessary, allowing interaction customization at use-time.

There are no images, sounds, haptic stimuli, buttons, sticks, touches. There are only numerical and categorical variables (for instance, positions, states, mensurable attributes stored as logic components in entities), and relations (expressed as rules implemented by subsystems) between them to abstract the system.

Figure 5 illustrates the data flow for the architecture. The game loop exists as one or more logic subsystems, which processes logic data from components according to rules. In the figure, the center (logic) is always running. The right side (output) exists only in Games – they define a player should perceive each entity and event. The left side (input) may exist in an MG in a form of AI agents. In Games, they will map physical-level input from a player to the game.

Relations among components define how variables change over each iteration of the game loop and due to commands (abstracting interactions). As an MG never processes input directly, commands provides an intent for interaction (for instance, instead of pressing the button “A” to make an entity jump, an MG handles a “jump” command) (Figure 5, center). Once a command is issued, the MG reacts to it. A logic subsystem translates commands to data variations over components. Provided the variations are valid, other subsystems perform their computations as usual.

As commands are events, machines can play the game implementing algorithms.

If an algorithm reads logic components to gather data, it can analyze the current situation (state of the system) to determine its play. As a command is an event, that means the algorithm dispatches an event to provide its intent – how it plans to act on an abstract game world. AI agents are, thus, similar to complex EUD macros. AI agents can define abstract entities that are able to play the MG[35].

2.5.4 The Game to the Player: Converting a Meta-Game to Human-Playable Games

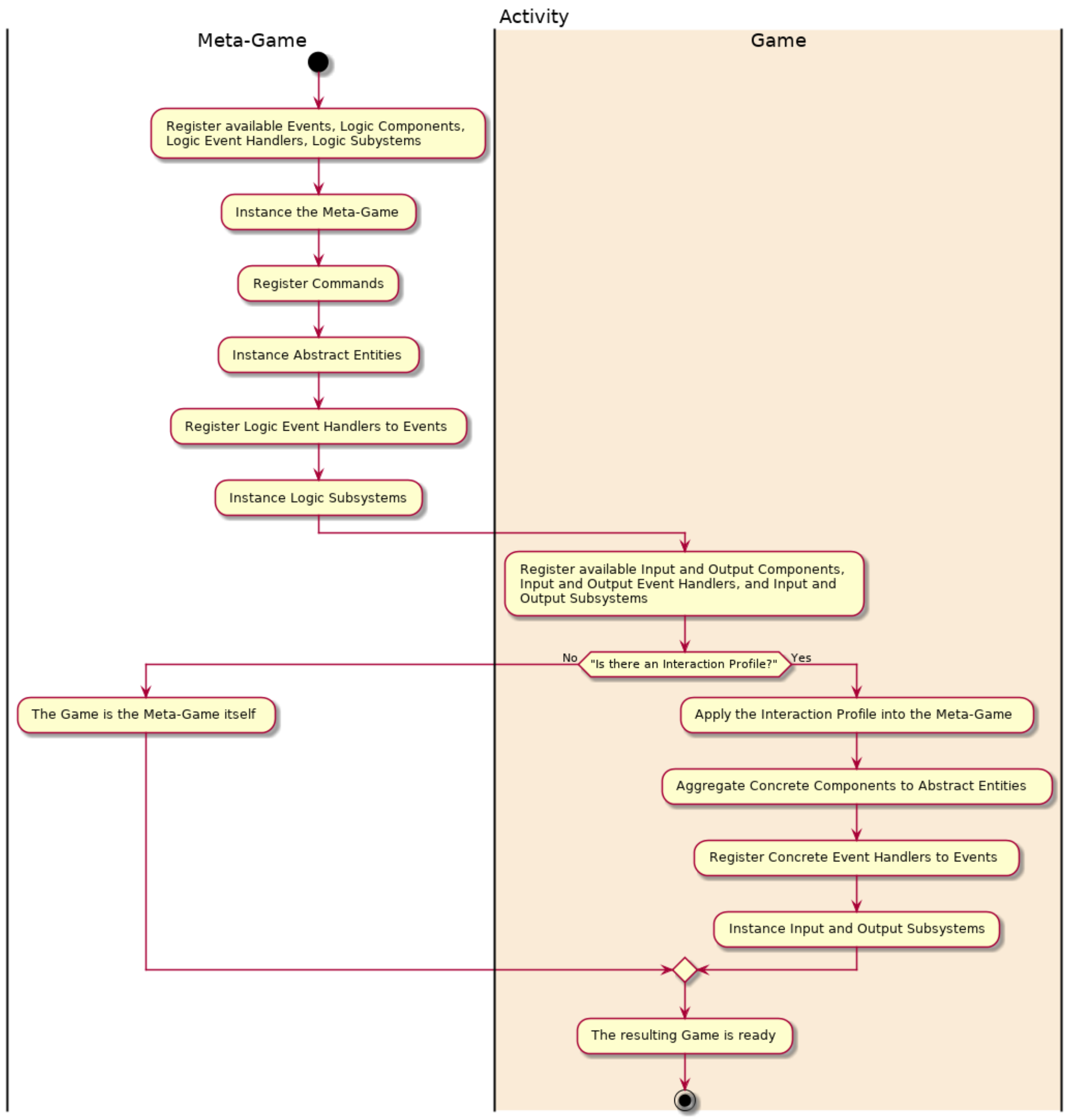
The main usefulness of an MG is abstracting interactive features despite its lack of human-related IO. This enables us to attach any IO features that we want (both for enabling interaction, and for aesthetics) to construct Games. In special, this means that we can tailor the IO to suit interaction needs of people. We represent how to apply needs in an Interaction Profile.

Figure 6 provides a high-level activity diagram of the run-time tailoring algorithm, complementing the previous referenced Figure 5, which suggested the data flow for the architecture. We divide components and event handlers according to their (mutually exclusive) roles: input, output, or logic (non IO). An MG uses logic components and event handlers to define game rules and gameplay. Instead of explicit user inputs, game commands handle events to simulate the input. During run-time, an external resource (a text file named as interaction profile) describes which IO components and events should be added to the game. The game engine parses this resource to instance the game's interaction – transforming the MG into a playable game. This means that, as the IO is generated at run-time, it is possible to redefine the game interaction by switching the active interaction profile.

As the MG fully defines the logic (rules, commands for interaction, internal state management) over non-IO data (stored in components), we can create arbitrary ways to define the game aesthetics (how the game is presented sensorialy) for output, and physical-level interactions for input. Interaction is the result from composition/aggregation of IO components, subsystems, and/or event handlers to convert abstract elements into concrete ones. Consequently, an MG becomes a possible Game. The composability is a direct result from the inner architectures – ECS for entities, EDA for events (Figure 7 and Figure 8). As a result, we can create multiple IO versions from the very same MG, which behave exactly the same way logically, but can appear, sound, smell, taste, and touch completely different.

[35] AI Agents provide a useful side effect – as they can provide input, they can be used for playing automation. For motor disabilities, this means that AI Agents can automate part of the input, potentially helping Players to play the game. Similarly, they can help players with visual and cognitive disabilities to play.

Figure 6 – Activity diagram to convert Meta-Games into Games.

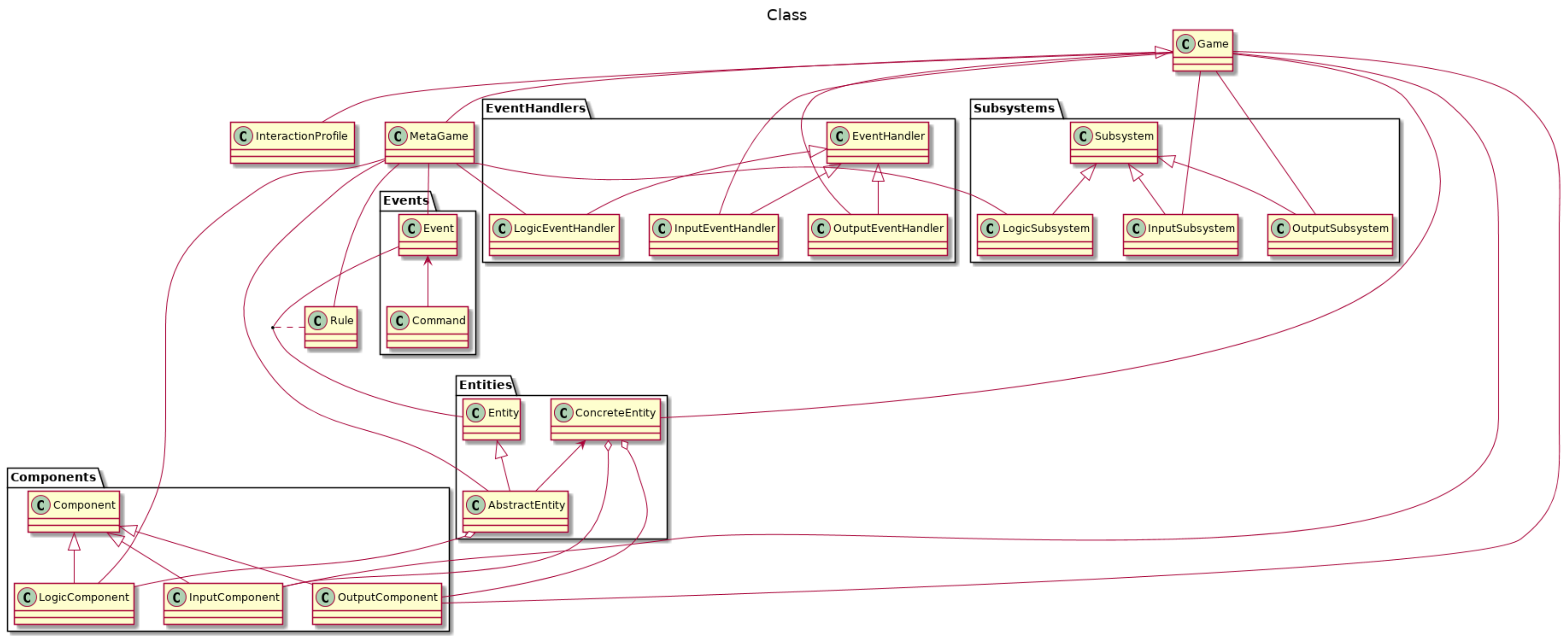

Activity
Meta-Game
Game
Register available Events, Logic Components, Logic Event Handlers, Logic Subystems
Instance the Meta-Game
Register Commands
Instance Abstract Entities
Register Logic Event Handlers to Events
Instance Logic Subsystems
Register available Input and Output Components, Input and Output Event Handlers, and Input and Output Subsystems
No
"Is there an Interaction Profile?"
Yes
The Game is the Meta-Game itself
Apply the Interaction Profile into the Meta-Game
Aggregate Concrete Components to Abstract Entities
Register Concrete Event Handlers to Events
Instance Input and Output Subsystems
The resulting Game is ready

Source – Created by the author.

As we aggregate interaction to the MG to define Games, we can combine any previously combine interaction alternative with any new ones that we create. This way, every new alternative may benefit enable new people to play, as well as provide more options to all included users. The composability enables mix-and-matching alternatives to define custom tailored Games to suit more granular needs[36]. For instance, we can combine alternatives defined for hearing disabilities and motor disabilities to enable someone with both disabilities.

36 Perhaps, with enough features, we could approach individual levels of accessibility (that is, supporting the abilities of whoever was using the game) towards universal access; at worst, we would be able to include as many audiences as possible.

Figure 7 – A possible class diagram to describe the architecture.



Source – Created by the author.

This interaction composability of the architecture allows to tailor the system in use-time to the needs of the player – from the general to the particular cases. A player can consult available options, combine a subset of them, define a Game, and play to evaluate its suitability (Figure 4). At any time, she/he can change combinations and resume playing – as the MG is the same, the simulation can continue as normal. The new Game, however, can be completely different from the previous, both for appearance and playing features. The way a player perceives and control a Game depends exclusively on the chosen concrete elements attached to tailor the MG.

For examples of real-time prototypes implementing the architecture, the reader may refer to two high fidelity tech demos[37]. One is a clone of Atari Inc's Pong®[38]; the other is a clone inspired in Access Invaders (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009) (which is a clone of Taito Corporation's Space Invaders)[39]. Both games support run-time (re-)mapping of IO provided by the architecture. In the games, we explore an external data resource (a text file) as an Interaction Profile to represent desired concrete elements to build a Game from the MG. Figure 4 shows different Games derived from the same MG, in a same play session at different moments; attached IO-elements during run-time define the interaction. The active Interaction Profile may redefine entire IO features with the game running[40].

For input, the MG is only concerned with commands. For the logic, it shall not matter the origin of the command. Whenever it happens, the MG processes it. If it is valid, the internal state changes. Therefore, it does not matter whether a command request comes from a button press, a keyboard key press, a gamepad button press, an automated command from a system, a voice command, or an assistive technology. Provided that we can map a physical level interaction to the command, we can use the device as controller. The MG knows about the command, how to process it, and what to do when it is valid. Whenever it happens, it handles it. Developers are free to choose and implement bindings for whatever input devices (and mappings) they want – from traditional, to assistive technologies, to computer code (to provide automation to help players with motor disabilities, for instance). The only requirement is dispatching the command then the player uses her/his selected bindings.

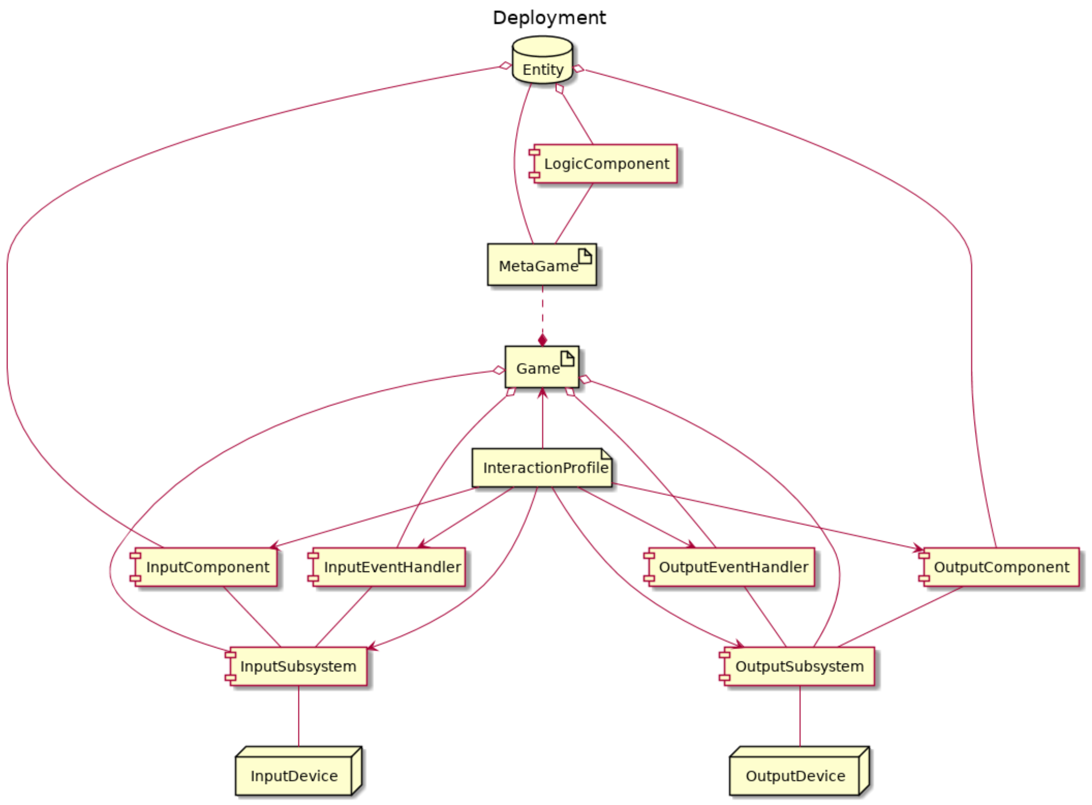
[37] They are demonstrations for the architecture, not the most inclusive games as possible. To make them truly inclusive, we would need to improve the non-graphical versions, allow the use of more input devices and assistive technology, and perform accessibility and usability evaluations. These are demonstrations for the flexibility of the architecture, and to show our approach to inclusion with tailorable games. We break the rule of not changing the MG at Section 2.5.5, to describe how we can further explore the architecture for accessibility.

[38] Available at <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong>.

[39] Available at <https://gitlab.com/francogarcia/GamesRunTimeTailorability>. We have chosen Access Invaders as an example because it is an example of UA-Game.

[40] Including UI elements for Head-Up Displays (HUDs); we can define alternatives and introduce them arbitrarily.

Figure 8 – Deployment diagram illustrating the run-time tailoring.



Source – Created by the author.

Note – An interaction profile defines what input and output components should be aggregated to transform abstract entities into concrete ones. Input and output subsystems mediate the communication between software and hardware (IO devices). Similarly, the profile describes what input and output event handlers should be subscribed to handle events when they occur.

Similarly, the MG handles the internal state with its logic components. As mentioned in Subsection 2.5.1, logic components hold plain-old data, with primitive types (integers and real numbers, Boolean values, categorical values or strings) or data structures of primitive types. They do not store presentation data; they hold data to define and simulate the game world. Therefore, developers are free to define concrete components to represent them for any senses, devices, and technologies they want. The representation of an entity might be a tangible object, an image, sounds, vibrations; a combination of those, perhaps. IO elements can read logic ones without modifying them. This ensures that the MG works without external intervention, granting Games with freedom for representing it as desired.

Figure 3 illustrates possible combinations of components. Logic elements are always present in any generated Game, as they define the MG. Input and output elements can be freely combined to define the interaction. As we attach IO behaviors to entities at run-time, we can toggle them at any time. For input, as commands are decoupled from interaction, multiple people may share a same entity to play together and help each other (like co-pilots). Likewise, we can implement automation features to assist players. For output, components represent an entity continuously.

2.5.5 Elements of the Architecture For Game Accessibility

The architecture defines few elements. As a result, if we can define an accessibility feature exploring a combination of components and event handlers, then we are able introduce it at run-time to assist a player. Plug-and-play accessibility. This applies to strategies from Yuan, Folmer & Harris (2011), for implementation of the Unified Design from Grammenos, Savidis & Stephanidis (2009), Grammenos, Savidis & Stephanidis (2011), Grammenos, Savidis & Stephanidis (2007), as well as any other accessibility guidelines and recommendations, as well as our own problem solving abilities. We can define any number of concrete elements to implement the feature, as well as reading any data from abstract elements. The only rule is that concrete elements cannot modify abstract ones; otherwise, specific IO interactions would change the game logic[41].

In particular, the definition of Rule states that relations among entities result in events. For instance, considering e as entities, R as a rule, and v as event, a binary rule[42] can be defined as:

$$e_i \, R \, e_j \mapsto v_k$$

In the expression, R could be a mechanic – for instance, `fire`, `walk`, `talk`. Each of these mechanics results into an event (`EntityFired`, `EntityWalked`, `EntityTalked`) – for player interaction, they can also have a companion command (`FireAt`, `WalkTo`, `TalkTo`). The same reasoning applies to implicit behaviors between entities – for instance, `collision` and `EntityCollided`, `enter area` and `AreaEntered`, `leave area` and `AreaLeft`.

In conventional games, these events are usually marked by sound effects; when even when they are not, the player can see them. In accessible games, they are more important. As events mark relations, the MG semantically defines interactions between entities with a mechanism that enables developers to decouple implementation. For accessibility purposes, this allows us to provide instant stimuli as soon as the event happens. As multiple event handlers may process an event, this enables developers to:

- Provide immediate sensory feedback (visual cues, sonification, haptic cues);
- Provide long term feedback (for instance, with a pair of begin/end events);

41 Actually, we can circumvent this rule for single player games, as we mention at the end of this section.
42 Developers can define other arities as needed.

- Assist automation – a subroutine may listen to events to support the player;
- Automatically toggle automation;
- Arbitrarily execute code (for instance, to provide an additional user interface to help the player);
- Provide data to assistive technologies (for instance, text-to-speech);
- Change the execution flow (slow down the game, increase response time, introduce new content, pause the simulation to let the player think);
- Inspect the MG to provide further information to players.

These marks with events are useful to introduce accessibility features at the moment they are needed. The help developers to analyze important moments to introduce accessibility features – compose accessibility as IO –, and help players with visual disabilities to perceive what is currently happening. Vision allows for scanning the environment, providing an overview of what is happening into the game world. These are implicit cues that players with visual disabilities cannot perceive. Semantic events helps these players to perceive them via other sensory stimuli.

As commands are events, they share these same properties, with one additional benefit: subroutines may dispatch commands to assist the user during play. Input elements may continuously analyze the MG to help the user, or to act on her/his behalf to support him. These partial automation are usually already part of a game project as implementation for AI – for instance, pathfinding, movement, and aiming can help a player in an action game[43]. In the same way that they can control AI-agents, they can help players with disabilities to play – the AI calculates the aid and/or execute the command to help the player. We can also think of commands as a low level API to define higher-level accessibility features – or macros, compounding commands in sequence. For instance, we can fetch a list of nearby entities and define a higher level `WalkToEntity` command based on the `WalkTo` one, and provide this into a pair of input and output interfaces. When we compose these interfaces to the game, we provide the new command (thus, players who do not want it will not have it). This can aid players with motor, vision, or cognitive disabilities to navigate in the game world. The same reasoning applies to other commands – in particular, cheating strategies in conventional games can act as accessibility features.

[43] Another approach is enabling cooperative play, on which multiple players share the same entity to help one another to play.

Mutable Meta-Game: Every Rule has its Exception

Finally, there is the rule of never changing the MG. If we want to provide multiplayer support for players with different (dis)abilities with multiple tailorable versions, we should not modify the MG. If the MG is the same, the authoritative server for the logic is the same, and, therefore, every Game runs equally – they can be tailored client-side to each player. With client-side tailoring, every player can have his/her version with IO features suiting her/his interaction needs. Every version can be completely different from each other as well. The communication protocol (commands) are the same for everyone.

For single player purposes, however, we can modify the MG if needed – parameterize it, change defaults – to suit the needs of single player games (or single player mode).

Can we transform an action game into a turn based one? If it is computationally possible, and we can decompose in the elements of the architecture, we can implement and introduce a solution as at run-time. For instance, if time is an important constraint to the game, controlling time can be a command. We can pause the game, modify the interaction, provide more time to the player, introduce new ways to play; once done, we resume, the Game provide the command to the MG. Milliseconds, thus, can become seconds, minutes, hours – real-time can become non real-time. If we introduce it to the game, it becomes a feature – perhaps a time warp mechanic[44]. Then we can make it fun. Moreover, if we introduce this new feature as part of the original MG, we respect the rule once again. An accessibility feature improved the game for everyone else.

Once again, we respect the interaction needs of the player and her/his choices. We can provide, she/he can choose how to play. She/he can modify her/his playing choices over a game session as well; it is a run-time tailoring architecture. We are only limited to what we can computationally solve and implement, the available IO technologies, and how we can translate data into information to people. If we can find a solution and implement it, we can plug-and-play accessibility to create Games and enable more people to play them. The MG to the machine, the Game to the Player[45]. Next step: the Game *from* the Player.

[44] For an example, the reader may refer to <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong/tree/interaction_remapping>. In this branch, one interaction profile monitors the distance between the Player's 1 paddle and the ball. When it is going to the side of the player, the game slows down; when it approaches the paddle, we stop time and introduce a different interface (a button, as proof of concept). The game resumes after the button is pressed. With run-time tailoring, this feature is gone once we change profiles.

[45] The reader may consult Appendix B for more information about the architecture, including its formal definition and an example of how to implement games with it.

2.6 The Collaborative Work Model for Co-Creation of Tailorable Digital Games: The Community for the Individual, the Individual for the Community

Once we had a flexible architecture (Section 2.5), we built a collaborative work model around it[46]. As the architecture allowed us to work iteratively towards accessibility, we wanted a development process to work iteratively towards inclusion. We wanted this process to be centered around abilities and skills, to show that everyone can contribute towards making a better game for everyone else. As interaction was decoupled from the logic implementation, people could add or improve existing alternatives based on their abilities.

Therefore, when we refer to games *for everyone*, we aim to enable as many people as possible to play. Games are the final product. With the architecture, we are able to explore traditional "one core fits most" (for mainstream audiences), accessible, and custom versions at the same time to address interaction needs for creation. It only depends on the available elements to tailor the MG, how we combined then – IO is composed to the game –, and of the people helping to achieve it – community-based accessibility.

As in traditional development, programmers can include new accessibility features. However, our audiences include people who are not programmers, who may had never played a game before, or who had never used a computer before. Asset and content creation are fundamental parts of game making. Therefore, they offer more opportunities for collaboration. In particular, this would highlight abilities from people with disabilities – they would have unique skills which could enable their peers to play, as well as improve the game to everyone else.

In this context, we chose to provide pre-defined mechanics for EUGD at first. We defined a mechanic, considered alternatives to use it and convey it to other audiences, implemented them, and let the community provide content based on their abilities[47]. EUD wise, these activities relate to parameterization and annotation. This way, people would start with content creation; storytelling mechanics were an ideal initial scenario. First, they are centered on content creation and communication. Second, the entry barriers are low – skills including speech, writing, drawing, and sign language can be explored to create stories and convert them to suit different interaction needs. Third, people who had never played games would probably have had previous contact with some form storytelling – for instance, books, theater plays, movies, or soap operas.

[46] Appendix D further details the collaborative work model.
[47] We provide the mechanics to define the Meta-Game, creators provide assets and content to convert their Meta-Games into Games.

Fourth, collaborative storytelling could serve as a first contact with collaborative game development. Finally, we could advance to role-playing games as a next step to introduce new game mechanics (for instance, using the nouns and verbs strategy defined by Ahmadi, Jazayeri & Landoni (2012)).

The idea becomes simpler with an example. In a therapy group, there are people who can read, write, speak, and/or listen; different people may have different (dis)abilities and proficiency. One of the members starts creating a game; she cannot read nor write, but she can speak. She records her voice to create her story. Some members can hear and write – they listen to her story and transcribe it into text. Some members cannot hear, but can read the transcription. They create a sign language video for the story. In this quick example, members of the group created three versions to convey the story: in audio, text, and sign language. Using their own abilities and skills, they overcome barriers to enable their peers to understand the content. We follow this same principle for end-user creation of game content and assets – abilities are what matters for creation.

Regardless of mechanics and genres, two possibilities arise when creating accessible game content in EUD scenarios:

1. The creation tool may provide interaction alternatives for its default features. In this case, creators use the built-in features to create their game. When someone plays a game, the tool selects the best content and interaction alternatives to suit the needs of the player.

   In this case, the developers of the tools are responsible for creating and providing interaction alternatives. Although this possibility ensures that there are alternatives for existing content and features, it restricts what is possible to make – every creation mixes and matches pre-existing game assets. The inclusion of new ones is a task for the developers.

2. The creation tool may support user generated content. Creators can make their own game assets (such as code, media, story) to add to their games. This is the opposite of the first possibility – although it provides greater creation flexibility, there are no guarantees that new content will be accessible to heterogeneous interaction needs. In other words, whenever the original creators do not (or cannot) create alternatives, someone else will have to. Otherwise, some feature will be inaccessible for some audiences.

The trade-off may make the second alternative seems worst as the first, as, in traditional approaches, developers provide feature and content to users. Users are passive and isolated (a user, a group of users with similar interaction needs). Design is *for* them. Instead, we should design *with* them, as well as promote Meta-Design to enable them to improve the systems themselves. This way, we can reach *by* them. In this way, users become partners and co-creators. We should empower users to be active (providing efforts to improve the system) in the process, *and* consider them at individual (with their own needs) and collective levels (communities with heterogeneous abilities and needs). In this way, an individual may provide her/his abilities to improve a system for the community, as well as receive improvements from others to overcome her/his own needs.

This is the central idea of our collaborative work model, the Collaborative Co-Creation of Inclusive Digital Games (C3-IDG), which is a pillar of the framework. We focus on what people's abilities, skills, and knowledge, interests as ways to improve a digital game to other members of a community. The process is iteractive, for, whenever someone becomes able to interact and improve the system, she/he may collaborate to further improve it. Provided this new person can also interact and improve, we can create a process of iterative, continuous accessibility improvements towards inclusion. Instead of included and excluded users, we define collaborators as Enablers and Enhancers, whose individual abilities transform To Be Included people into Included users – potentially, new collaborators.

Due to the collaborative nature of the work model, a user who created the content does not have to be the only person creating all alternatives. Rather, a community can implement the required features (or include alternative, equivalent media) to enable more people to play. This decision turns game accessibility itself into a collaborative effort, on which people who could not use before may become able to use and to create, and, possibly, enable others to play. The community for the individual, the individual for the community. The community, working together, provide cycles of inclusion.

The model, thus, leads to a somewhat opportunistic strategy to inclusive design – an end-user powered approach defining mutualistic[48] and communistic improvements, on which every single addition helps more people to play. Moreover, people who could interact with the original system may become able to help to create it. As accessibility has cascading effects, an improvement hardly ever benefit single individuals; on the contrary. As an example, subtitles for dialogues are essential for players with hearing disabilities, and helpful for sighted players to follow the audio (especially if they are not fluent on its language). In turn, players with hearing disabilities might, then, become enablers; for instance, they might provide sign language transcriptions, or translate the subtitles to another language. When we include someone new, his/her abilities and knowledge may further improve the game, and, possibly, include more people, providing another step for this enabling loop.

[48] Mutualism is a biology relationship on which organisms benefit from each other.

2.6.1 Transient Roles of the Model: "I Become What I am Doing"

In the collaborative work model, roles for participants are transient, in the sense that they can change according to what they are doing at a given moment. This transiency happens at two levels. First, there are roles regarding the status of inclusion for a person to a given version of the system:

Included Person for which the system provides suitable IO features for her/his interaction needs. An Included person can use the system to play and create – even if the interaction quality is not perfect.

To Be Included Person for which the system is not yet suitable. In this case, she/he cannot use the system in any way.

Collaborator Someone whose abilities can improve the overall quality of the system.

Enhancer A Collaborator who can improve the system. An Enhancer can modify existing features to improve them (for instance, improving aesthetics and/or user experience).

Enabler An Enhancer who can also enable new To Be Included to access the system. The difference is that an Enabler can provide new IO features which can transform To Be Included into Included users – in this way, they collaborate to improve the overall accessibility of system towards universal access at best, maximum inclusion at worst.

These roles depend on the available interaction features at a given version of the system. For these roles, the goal is allowing Included users to act as Enablers and/or Enhancers to increase the overall quality of the system, to improve quality of use and promote access. In particular, upon inclusion, newly-Included users may become Enablers and/or Enhancers as well, potentially resulting into cycles of improvements – newly-Included people may provide abilities, skills, and knowledge to iterate in the process. This creates a dynamic roughly described as "once you help me, I can potentially help more people" – people's abilities are for improvements.

Creation-wise, the second level defines three additional transient roles:

Supervisor Person who oversees the creation process, guiding it towards its conclusion. For serious contexts, this can be domain experts such as professors or therapists guiding their pupils. For casual contexts, this can be a person deciding the next steps of her/his project.

Creator Person who is creating the content for the game.

Player Person who is playing the created game.

The previous roles depend on the context and current activity that someone is performing. As the framework is for end-users, the roles further helps development, as they are part of a game creation process.

2.6.2 Game Creation Process: Multiple Phases, Multiple Opportunities for Collaboration

To support their creating endeavors of end-users, we defined a game creation process divided in eight phases lead activities. Each phase provides opportunities for people with different skills and abilities to further collaborate and enrich a game project. In this way, diversity brings greater wealth of ideas, needs, discussions, and features to continuously improve a game.

The process decomposes game creation into eight iterative phases (Figure 9):

1. **Conception**. The initial phase. A Supervisor idealizes the game project, define its goals (for instance, reflect about something, solve a problem, or a way of self-expression). It proceeds to Conversion.

2. **Conversion**. Programming is not the end-goal of our framework; rather, we want to enable them to create games. Shortcuts to programming are, therefore, welcome. For instance, designing with drawings, collages, and textual descriptions. The tangible material acts a low-fidelity prototype, which Collaborators (or automatic conversion) can transform into an initial, high-fidelity game prototype[49]. The prototype can be low, medium, or high-fidelity, depending on what is currently available and its goals. Once created, the process proceeds to Evaluation.

3. **Evaluation**. A Supervisor evaluates of the current state of the project, to define what the next phase should be, depending on what is currently needed (or more important to continue the project). The next phase can be Creation, Enrichment, Distribution, Use, or Conclusion.

4. **Creation**. Creation of content for the project. With the initial prototype, Creators use an accessible editor to modify and improve the game. They add and refine art, story, and mechanics to create interactive experiences. For accessible results, the

[49] In the future, we could explore approaches including paper prototyping (PixelPress – <http://projectpixelpress.com>), drawings (Google Quick, Draw! – <https://quickdraw.withgoogle.com>), natural language (GURI VR – <https://gurivr.com/> and WordsEye – <http://www.wordseye.com/>), and creative input devices (Makey Makey – <http://www.makeymakey.com/>, virtual reality). This could promote multimodal creation involving words, speech, drawings, image recognition, movements. . .Different approaches for different abilities. Provided there is a suitable one for a person, she/he could become able contribute. Creation is about abilities.

editor provides guidance on how to create accessible alternatives for new media and mechanics. For instance, subtitles and sign languages for spoken dialogues, onomatopoeias and graphics for sound effects, textual and aural descriptions for images, and input automation. It proceeds to Evaluation.

5. **Enrichment**. Call for the aid of Collaborators, to request improvements for people external to the original project. One single person does not have to provide all game content. Rather, people may work on the project to add and/or improve content collaboratively, according to their abilities, capabilities, interests, and strengths. It proceeds to Evaluation.

6. **Distribution**. Generation of playable versions (or prototypes) of the project, transforming them into playable games. For accessible games, distribution goes beyond sharing the game: it includes the generation of a suitable version. In this phase, the game is configured to suit the users' abilities and capabilities – or personal preferences. It proceeds to Use, if there are suitable alternatives; otherwise, it backtracks to Evaluation (a Supervisor should request the alternatives).

7. **Use**. Players play the generated Game. It proceeds to Evaluation.

8. **Conclusion**. The final phase is the end of the project.

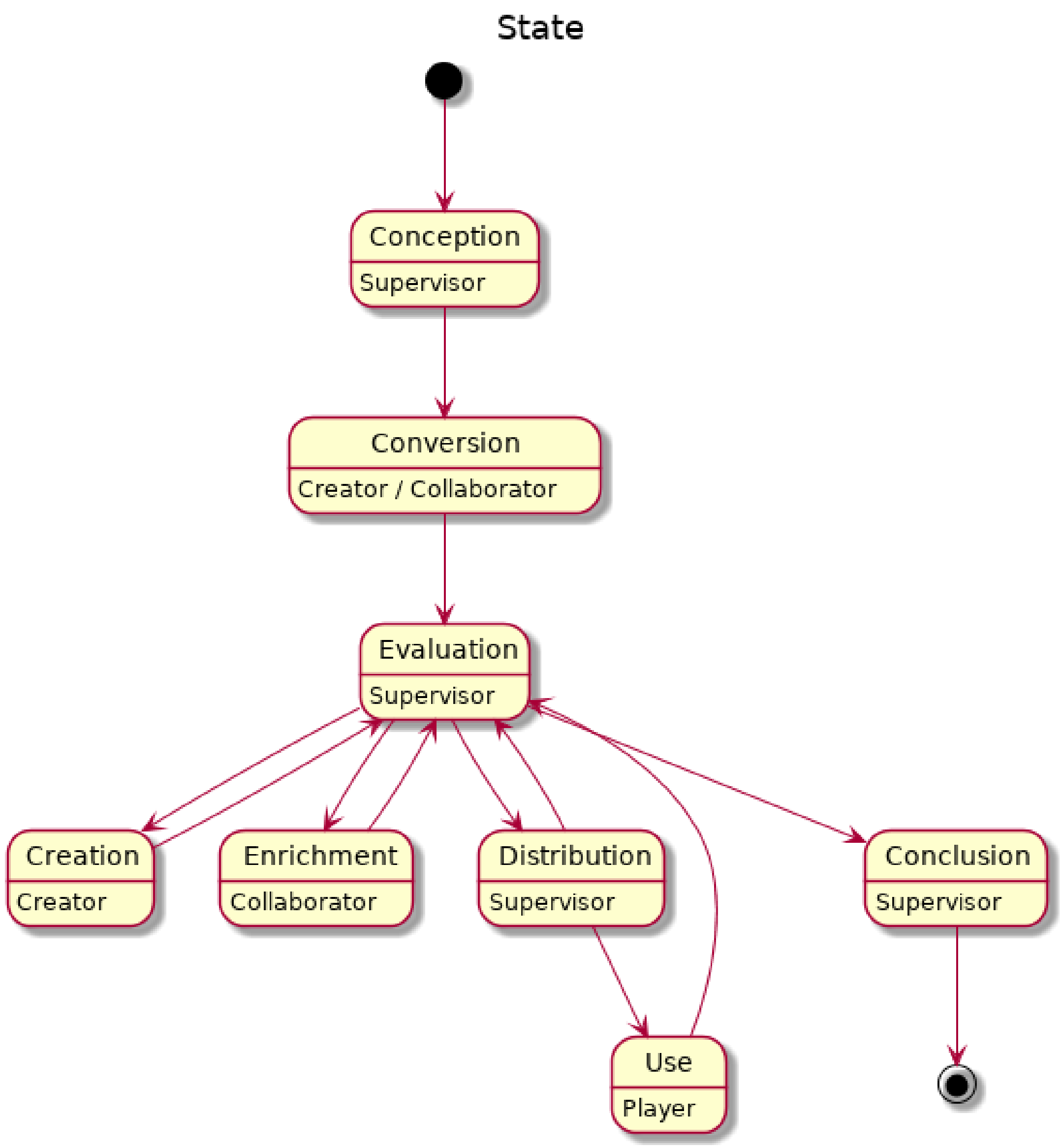
The Conversion phase of the framework tries to explore low-level technologies for initial creation – in special, everyday material for game design. At the one hand, this promotes the use of tactile material, which might be useful for motor and cognitive disabilities; it also avoids overwhelming people who had never used computers or played games. On the other hand, it (potentially) allows the exploration of nouveau approaches for content creation. Currently, we have explored low-fidelity prototypes created with paper and pencil, and high-fidelity prototypes created with our game creation platform, Lepi.

Due to possible contexts of use and the enabler/assisted nature of collaboration (with transfer of knowledge/abilities), a domain expert may explore the framework as tool to support her/his professional activities. In this context, we embrace tailorable game creation as a form of learning, practicing, and 21st literacy. Furthermore, this expands game creation as a potential tool for reflection and self-knowledge – which could be useful for educational and healthcare contexts.

Activities within the Phases

Subsection 2.6.1 and Subsection 2.6.2 defined the roles and the collaborative creation process to create games with the framework. To manage the process, we defined

Figure 9 – A state diagram representing the phases of the collaborative work model.



Source – Created by the author.

a work flow for development, as follows (Figure 10)[50]:

1. Conception:

   a) A Supervisor defines the scope for a new game project, defining domain-specific goals, theme, and activities and milestones.

   b) Actions:

      - Define scope of the project;
      - Select initial assets;
      - Define game parameters;

[50] Although we assume a single Supervisor for convenience, there could more multiple ones.

- Register creators;
- Provide the project.

c) The Supervisor invite Creator(s) to participate in the project.

d) Next phase: Conversion.

2. Conversion:

a) A Creator generates a first prototype for the project using whatever material and fidelity the Supervisor instructs her/him to. The Creator might either use a system to create her/his prototype, or request assistance from a Collaborator.

b) Actions:

- Generate the prototype;
- Submit prototype for evaluation.

c) After finishing, the Creator submits her/his results to the Supervisor.

d) Next phase: Evaluation.

3. Evaluation:

a) The Supervisor oversees and manages the project, and analyzes its current state, verifying how it currently satisfies the proposed goals.

b) Actions:

- Fetch projects for evaluation;
- Select project for evaluation;
- Review project;
- Request improvements for the project;
- Request improvements for assets;
- Request interaction alternatives;
- Approve the project.

c) Next phase:

- Creation, if she/he requests further content creation for the Creator(s), fixes, and/or improvements.
- Enrichment, if she/he desires external Collaborators to improve the project, or to create interaction alternatives for existing content.
- Distribution, if she/he wishes to share the project for someone to play.
- Conclusion, if she/he considers that the project should be finished.

4. Creation:

a) A Creator uses a game editor (for instance, Lepi) to iterate on her/his game (Meta-Game). She/he can add new features and content, and/or improve existing ones.
b) Actions:
   - Fetch available projects;
   - Acquire project assets and content;
   - Iterate on the project;
   - Submit project for evaluation.
c) After finishing, she/he submits her/his results to the Supervisor.
d) Next phase: Evaluation.

5. Enrichment:
   a) The Supervisor shared desired content with Collaborators.
   b) A Collaborator works on a task assigned to her/him – either to improve existing content (Enhancer), or to define new accessibility features and/or content to iterate towards inclusion. They provide their knowledge and skills to perform the tasks.
   c) Actions:
      - Fetch requests;
      - Iterate on request;
      - Submit artifact for evaluation.
   d) Next phase: Evaluation.

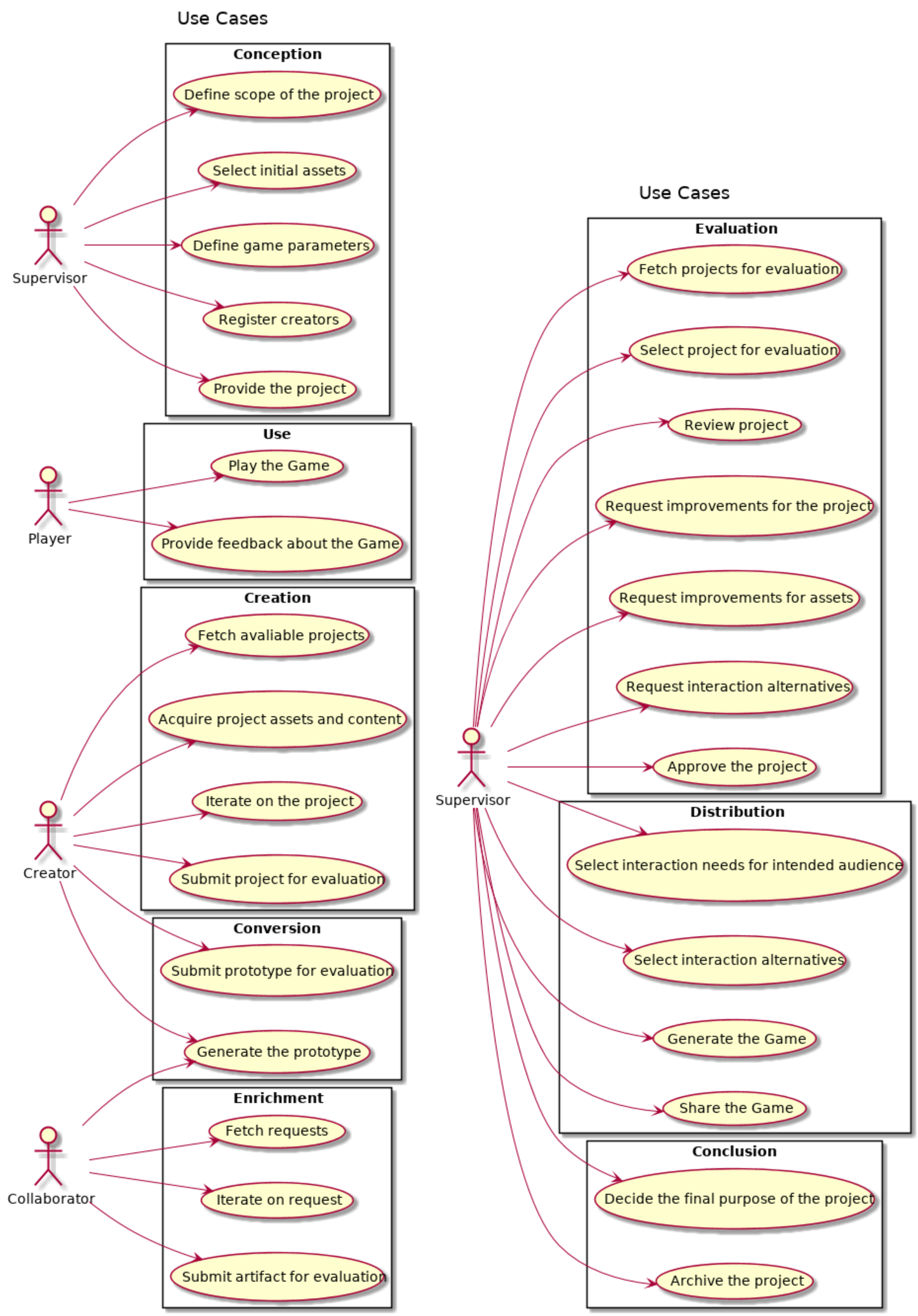
6. Distribution:
   a) The Supervisor defines her/his intended audience for to create a Game from the Meta-Game.
   b) She/he selected what interaction alternatives to include into the Game. The system may support her/him by recommending combinations, excluding conflicting alternatives which can lead to unsuitable interaction, and showing interaction needs not yet contemplated.
   c) Actions:
      - Select interaction needs for intended audience;
      - Select interaction alternatives;
      - Generate the Game;
      - Share the Game.

   d) The system generates the Game;
   e) The Supervisor distributes the Game to her/his intended audience;
   f) Next phase: Use.

7. Use:
   a) Players play the game. They may provide feedback regarding the game.
   b) Actions:
      - Play the Game;
      - Provide feedback about the Game.
   c) Next phase: Evaluation.

8. Conclusion:
   a) Everything shall come to an end sometime. This is it for the project.
   b) Actions:
      - Decide the final purpose of the project;
      - Archive the project.

Figure 10 – A use case diagram relating roles of the collaborative work model with their tasks within each phase of the model.


Use Cases
Conception
Define scope of the project
Select initial assets
Define game parameters
Register creators
Provide the project
Supervisor
Use
Play the Game
Provide feedback about the Game
Player
Creation
Fetch avaliable projects
Acquire project assets and content
Iterate on the project
Submit project for evaluation
Creator
Conversion
Submit prototype for evaluation
Generate the prototype
Enrichment
Fetch requests
Iterate on request
Submit artifact for evaluation
Collaborator
Use Cases
Evaluation
Fetch projects for evaluation
Select project for evaluation
Review project
Request improvements for the project
Request improvements for assets
Request interaction alternatives
Approve the project
Supervisor
Distribution
Select interaction needs for intended audience
Select interaction alternatives
Generate the Game
Share the Game
Conclusion
Decide the final purpose of the project
Archive the project

Source – Created by the author.

2.7 Support Systems and Approaches: Game Creation Tools

Section 2.5 and Section 2.6 defined the theoretical parts of the framework – how people could collaborate to create tailorable games using a suitable architecture. However, as end-users are not professional programmers, the architecture is not particularly useful to them. Similarly, if end-users cannot create the software, the collaborative work model is not useful, either. To enable game creation, we need to provide inclusive EUGD approaches. As developers, we should remove the barriers to enable users to create. Afterwards, together and supported by the framework, we need to empower users them to remove further barriers; this way, they may include their peers.

2.7.1 End-User Creation of (Meta-)Games: Lepi, an Inclusive Game Creation Platform for Inclusive Storytelling Games

Although the architecture described in Section 2.5 was originally conceived for games, it can support other interactive applications – including game development approaches. As an additional test for the architecture and proof of concept, we built the last pillar of our framework – the Lepi Game Creation platform. We opted to implement Lepi using the Godot Game Engine instead of using our own (defined in (GARCIA, 2014))[51] [52]. For the purposes of the framework, Godot provides the following benefits:

- It is open-source, allowing to modify it and include accessibility features directly into the editor and engine;
- It provides a self-contained editor for the creation of games, with features to implement, debug, and profile games;
- It has documentation and examples;
- It has an active community;
- It has support for internationalization;
- It supports for multiple programming languages (C++, C#, GDScript, a visual programming language, and the possibility to integrate other languages);
- It is Multi-platform (Linux, Mac, Windows) and exports to multiple platforms (including Linux, Mac, Windows, iOS, Android, Web);
- It could serve as an advanced tool for people who moved from our own solution (that is, in the future, once it is mature).

[51] <https://godotengine.org/>

[52] Although, at this time, Lepi is an application created with Godot, it could, ideally, extend Godot in the future.

As an initial study and proof of concept, Lepi supports the creation of digital storytelling-based games, such as adventure, point-and-click, and visual novels. In this proof of concept, we implemented features to support people with low literacy, people with hearing disabilities, and average users for creation. Our intention is to expand creation audiences first, then advancing to new mechanics over time[53]. People first, inclusion second, technology next.

As an editor, Lepi abstracts the creation of MG from the users (Figure 11). Although Creators think that they are making Games, they are actually programming MGs. To support *games for everyone*, Lepi promotes content alternatives towards inclusion by tailoring. When producing new content, Creators are encouraged to include content alternatives. We abstracted those as slots. For our current audiences, we support text, audio, and video slots support reading, listening, and sign language (video) skills[54].

A single user should try to provide alternatives for as many slots as she/he can. However, she/he does not need to it alone. During the Evaluation phase of the work model, Supervisors may request Collaborators to craft new alternatives. During the Distribution phase, Supervisors build Games – they decide what interaction alternatives should exist into the generated Game to suit the needs of their Players[55].

To support *games by everyone*, Lepi as a program is, itself, a Game derived from an MG. Creation features are commands implemented without IO references, which we can map to devices and physical level interactions. To enable more people to create, we re-map interaction – we create interaction alternatives, add to an existing (or define a new) user interface, and provide them for use.

2.7.2 Generating Tailorable Games

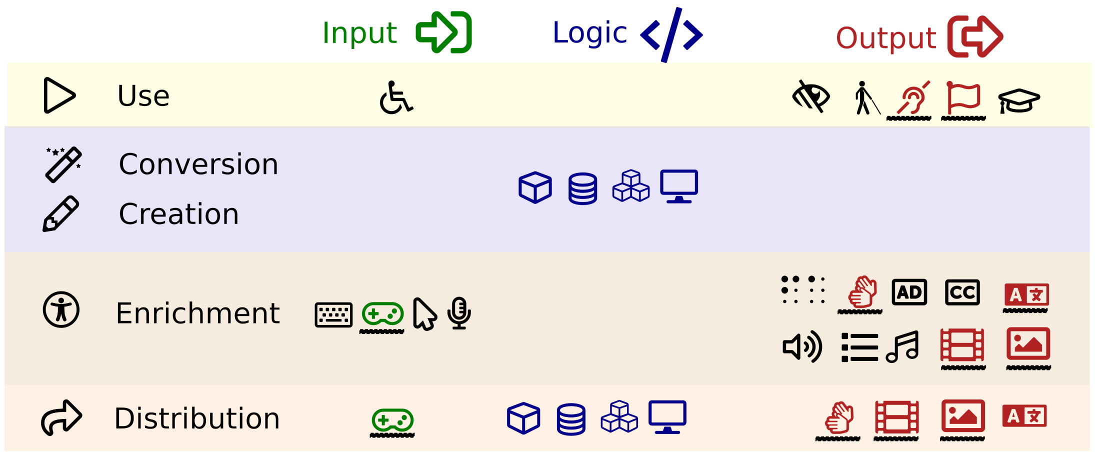
To generate Games, we explore the architecture described in Section 2.5. Missing concrete elements limits what is possible to tailor into the game; once we provide the alternatives for the content, we can generate an accessible Game for an audience. With

[53] Components and events handlers allow us to provide new mechanics as stencils. When Creators attach a component to an entity, it enables the entity to perform the rule (mechanic). This way, we can pre-define (implement) mechanics, figure interaction alternatives based on content (for instance, media for different audiences), and add them as stencils for game creation. The same reasoning applies to events and event handlers – for immediate feedback. Furthermore, we can expand this strategy for programming in the future – people could start programming their own components, event handlers, and subsystems to define new mechanics.

[54] We will expand this same idea for new mechanics. We implement the mechanics (and related commands), consider different ways to convey their information, provide them as templates into Lepi – or a new tool –, and create slots that users can fill with alternatives. As we can define what an entity is able to do via components, we can promote game creation by attaching "mechanics" components into entities.

[55] As the content, in the current version, is orthogonal, resulting Games can have any combination of text, audio, and sign language videos to convey content to a Player. This means that any player able to read, hear, or understand sign language in the languages of created games can play them.

Figure 11 – Creating games with Lepi combines the architecture with the collaborative work model.



Source – Created by the author.

Note – With Lepi, activities involving implementation in the Conversion and Creation phases generate MGs. To promote broader inclusion, a Supervisor may request improvements from Collaborators. Collaborators can create or improve features and interaction alternatives in Enrichment phases providing support for interaction needs. A project may have multiple Collaborators; any Collaborator may also contribute multiple times, providing their abilities, skills, and knowledge to improve the project. Once provided, the features and alternatives may become part of the system, potentially allowing for new ways of perceiving and controlling the Games derived from the MG. At the Distribution phase, a Supervisor can choose among these alternatives to generate Games to support the abilities for the Players at the following Use phase. Elements underlined in this figure generates a possible Game assembled at the Distribution phase. Other combinations would generate different Games from the same MG – the logic is always the same; the interaction can be completely redefined.

the current implementation of Lepi, we have typical use/play cases mapped into a story player.

For generating games suitable for different interaction needs, there should be a way to determine who can interact with currently existing features. At a lower level, this translates to being able to identify what interaction abilities are necessary to let a Player use a given feature.

To transform an MG into a Game, every game feature (mechanic, aesthetics, story, and technology) should map existing game content to the available interaction alternatives. At an architectural level, this means that there should be alternatives to transform abstract elements into concrete elements. Alternatives define what Supervisors can generate at the Distribution phase, for they are the IO-features that may be exported to generate playable Games[56] [57]. In this sense, the Distribution phases can resemble

[56] Although this is, currently, a task for Supervisor, we could do this programmatically. We could track alternatives for each asset of the game, identify which are missing alternatives, and inform a Supervisor to request them.

[57] Likewise, although this is the central idea, in practice, it may not be enough. We need to ensure that selected features are compatible with each other. Following Grammenos, Savidis & Stephanidis (2007),

software product lines, on which the final product (a Game) is composed of available features (to tailor the MG).

2.7.3 Enabling Tailorable Creation

The solution to enable more people to create is the same as to enable them to play. We learn from the needs an audience, consider a solution to enable use, implement it as a combination of IO elements, add to Lepi, and provide the system to the audience. As we are exploring a tailoring based approach, we can create a completely new user interface if required – the underlying processing technology (the MG) is already defined. As we have the commands (*what* a person can do to use the system), we can provide different ways to use them (*how* someone uses the system).

As we are designing systems *for* adaptation, this means that, if there exists a solution to enable use (for instance, assistive technologies), we can introduce it as a combination of interaction alternatives and include them at run-time. This means that we do not need to anticipate what accessibility features we will use, nor plan to add them in advance. Rather, we can introduce them when they are needed – and include them at use-time via run-time tailoring. The only requirement for this is to implement new components, event handlers, and subsystems. Once they are implemented, we can include them into an Interaction Profile to attach them to the system – accessibility features become, thus, plug-and-play solutions to enable use.

2.8 Evaluation

Our goal in the evaluation was to verify whether our research hypothesis presented in Section 2.1 was valid. Thus, we had to verify if the pillars of the framework fulfilled their goals to enable people to create games for people who had, potentially, different abilities than their own.

We defined the architecture upon our reflections about UGE. We tried formalize it with definitions, theorems, and algorithms. If our results are correct, we can express digital games in sets of the elements defined in Subsection 2.5.1. We chose set theory for the definitions because its ease of including and removing elements via unions, intersections, and differences of sets. Sets provide simple operations for developers who want to implement their own variations of the model. Moreover, the use of sets mean that we can work towards inclusion iteratively, adding one interaction feature at a time, including one audience a time, by defining concrete elements to tailor abstract ones. This

a possible approach consists of building a compatibility matrix to map game content alternatives that can be used simultaneously. The matrix helps to avoid the generation of unsuitable or conflicting versions of Games, as well as hinting on how to avoid sensory overloading (for instance, too many sounds at once).

way, every new interaction alternative adds to every other pre-existing ones, potentially enabling new people to interact, and offer more choices and quality of use for people already included.

Moreover, we created two high-fidelity prototypes to showcase the run-time flexibility of the architecture for separating logic from IO. The prototypes serve as demonstrations for the viability and the concept of the architecture. Links to these prototypes are available in Subsection 2.5.4; one of the prototypes is discussed with more detail in Subsection 2.5.2. These prototypes extended our initial related to game accessibility (for instance, audio games in (GARCIA, 2011; GARCIA; NERIS, 2013b) and UA-Games in (GARCIA; NERIS, 2013a; GARCIA, 2014; GARCIA; NERIS, 2014)). The implementation of Lepi serves as an additional viability study for the architecture – as well as a showcase that we can explore it in traditional applications.

The collaborative work model and Lepi were evaluated together[58,59]. We performed ten meetings over four months in a public healthcare service[60], on which people co-created digital games. Our participants were two healthcare professionals, ten users of the service, a team of five Collaborators, and the author of this thesis and his supervisor.

The users of the service were adults from lower socioeconomic grades who were in alcohol and drugs rehabilitation. They had no previous programming experience – some had never used computers before –, and low literacy skills. Our participants followed the work model activities, using Lepi during the Creation phase to implement their games[61]. The users of the service participated as Creators and Players[62]. Healthcare professionals participated as Supervisors. We also invited five people to aid as Collaborators. Three had Computer Science backgrounds (two undergraduates students and an MSc), and two had Nursing backgrounds (an undergraduate student and a PhD). Collaborators acted as Enablers and Enhancers, depending on the tasks.

Our meetings followed a pattern, starting with a description of planned activities, followed by a complete cycle composed of all eight phases of the work model, finished by a discussion of activities and results. Strategies for the Conversion and Creation phases varied according to the abilities of the participants. For instance, there were people who could not read, nor write. In these cases, they recorded their voices to include as game content, they drew their stories, and/or Collaborators helped them to transcribe content

[58] We had two scenarios for the evaluation. The first was a public healthcare service, with people with low literacy. The second was at an inclusive school, where some children had hearing or cognitive disabilities. For the second course, we provided a capacitation course for teachers; however, we could not start the activities with the students yet.

[59] More details of the evaluation are provided in Chapter 3.

[60] We followed research ethics protocols over the evaluation. Certificado de Apresentação de Apreciação Ética from Plataforma Brasil: CAAE: 89477018.5.0000.5504.

[61] Some preferred to use Lepi at the Conversion phase as well, instead of working with other materials.

[62] Some times, they also participated as Collaborators; in one case, a user even participated as a "small scale" Supervisor.

that they dictated. As Lepi supported text and audio as media for story, both approaches were valid. For participants who, initially, drew their stories, Collaborators helped to convert them to textual versions before inserting the content into Lepi.

We noticed that users could create their games using the framework for themselves and their peers (Figure 12 and Figure 13). This happened either independently or assisted by Collaborators – independence and interdependence, as defined by Bennett, Brady & Branham (2018). For the assisted approach, Collaborators helped the Creator to overcome an accessibility barrier. For instance, they would read content for people who could not read; they wrote content for people who could not write. Creators created according to their abilities; Collaborators complemented them when needed. Some people started assisted, becoming more independent over multiple meetings (interdependence moving to independence). Some were independent since the start – initial instructions and punctual questions were enough. Some people would need assistance for a very long time, which is also acceptable.

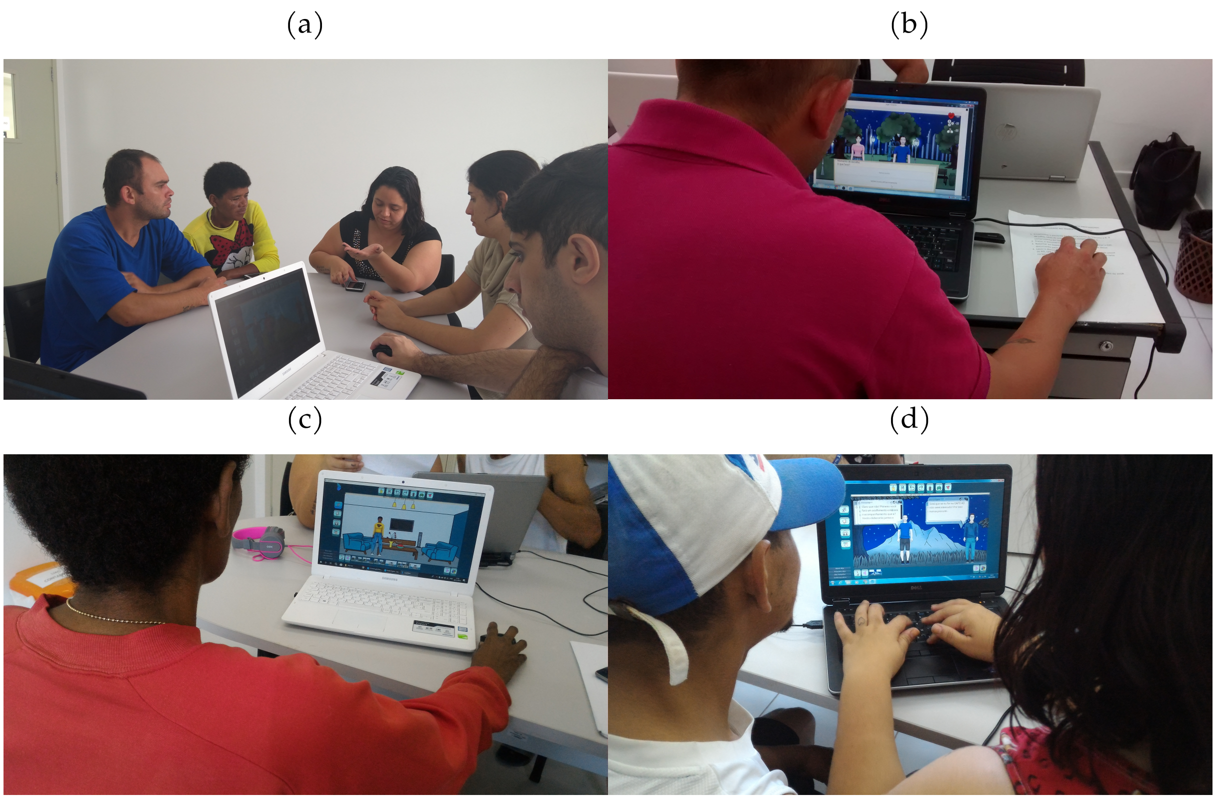
They were all co-creating. More importantly, they were able to co-create games suitable for both themselves and their peers. As the interaction abilities of the participants varied, resulting games had textual, aural, and graphical content to present the game to Players. Accessibility-wise, the most important result was verifying end-user created interaction alternatives.

People who could hear became able to play games originally featuring textual stories after Creators and/or Collaborators included audio transcriptions for the text. Similarly, people who could speak shared their games with people who could hear. At a later Enrichment phase, their stories embedded features textual alternatives included by Collaborators. One of our Collaborators had basic knowledge in Brazilian's Sign Language (LIBRAS). After she included it into some projects, people who could not hear had two options to play – read the textual version, or watch in-game LIBRAS videos for dialogues.

Overall, the alternatives allowed all participants to play based on their abilities. As a result, participants could play all projects created by the peers, which would not be possible otherwise. Without audio versions, for instance, participants who could not read would be unable to play projects with text only content. Once Creators and/or Collaborators introduced the audio, they became able to play independently. Although the participants at the service could hear, participants mentioned that they appreciated the inclusion of LIBRAS.

The framework, therefore, helped people with different interaction needs to play. It also resulted into an emergency of further benefits over meetings. Game creation helped the participants themselves. In some projects, the participants shared stories about themselves – their experiences with alcohol and drugs, the effects, how to overcome

Figure 12 – Participants creating and playing games.

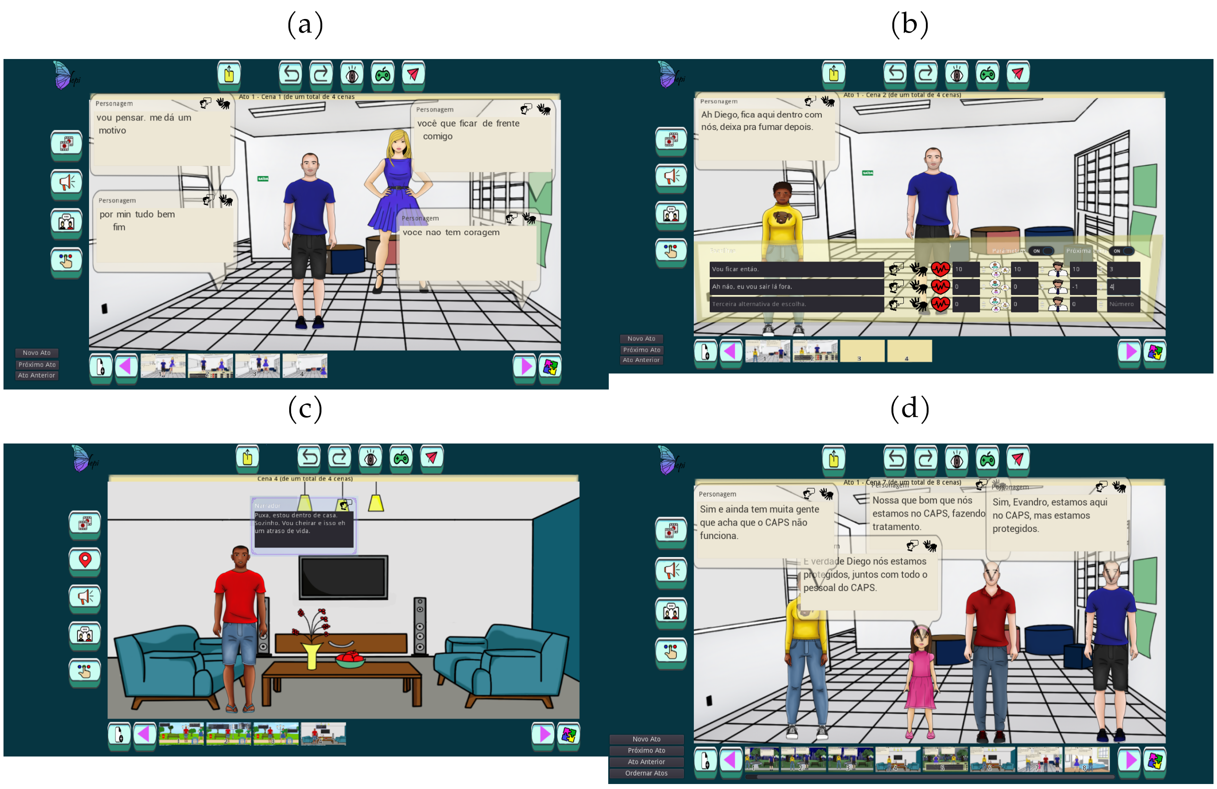


Source – Created by the author.

the addiction. They reflected about their past choices, considered alternatives ways to act, and taught and learned from each other. Over time, this dynamic leaded to self-growth. Moreover, people who had never used computers before started helping others to use the machines. This way, new people could play the games from the Creators – as well as create their own[63].

[63] We provided more details of the evaluation in Chapter 3

Figure 13 – Examples of games created by our participants loaded in Lepi.


(a)
Ato 1 - Cena 1 (de um total de 4 cenas)
Personagem
vou pensar. me dá um motivo
Personagem
você que ficar de frente comigo
Personagem
por min tudo bem fim
Personagem
voce nao tem coragem
Novo Ato
Próximo Ato
Ato Anterior
(b)
Ato 1 - Cena 2 (de um total de 4 cenas)
Personagem
Ah Diego, fica aqui dentro com nós, deixa pra fumar depois.
Parâmetros
ON
Próxima cena
ON
Vou ficar então.
Ah não, eu vou sair lá fora.
Terceira alternativa de escolha.
Número
Novo Ato
Próximo Ato
Ato Anterior
(c)
Cena 4 (de um total de 4 cenas)
Narrador
Puxa, estou dentro de casa. Sozinho. Vou cheirar e isso eh um atraso de vida.
(d)
Ato 1 - Cena 7 (de um total de 8 cenas)
Personagem
Sim e ainda tem muita gente que acha que o CAPS não funciona.
Personagem
Nossa que bom que nós estamos no CAPS, fazendo tratamento.
Personagem
Sim, Evandro, estamos aqui no CAPS, mas estamos protegidos.
E verdade Diego nós estamos protegidos, juntos com todo o pessoal do CAPS.
Novo Ato
Próximo Ato
Ato Anterior
Ordenar Atos

Source – Created by the author.

2.9 Limitations

We are considering very simple games in our study – with simple mechanics and few entities. Nevertheless, this does not necessarily imply that the framework cannot be explored in larger and more complex games. Rather, it means that the framework was not explored in such games yet. Thus, in the future, there is an opportunity to identify which mechanics and genres could benefit from it. With further research and increased maturity, this opportunity could mean that some existing mechanics could be adapted to enable broader audiences to play, as well as improve gaming experiences for currently supported ones. The same reasoning applies to game creation.

Moreover, although strategies in this chapter may apply to the industry, they are not our primary focus at this time, as they need scalability and real world evaluation. In the future, the architecture and the collaborative work model could contribute towards "accessibility modding" from end-user contributions, as well as game accessibility APIs, which developers could provide to enable the community to generate partial input automation. For professional developers, as a MG does not specify IO, the architecture could potentially contribute to porting – especially if explored in game engines targeting multiple devices.

Nevertheless, flexibility has its cost. The architecture incurs overhead that should be unsuitable for high-performance games. Performance, scalability, memory layouts, and cache coherence can be problems for adoption in large projects or real world scenarios. On the bright side, ECS, EDA, and Data-Driven Architectures are used in high-performance games. As we have not assumed any particular implementation, some of these issues could be mitigated with more advanced strategies[64]. However, with thousands or millions of entities, rules could generate thousands of events per game loop, severely impacting performance. Although we can optimize the solution, the discussion is not in the scope of this chapter.

In the collaborative work model, we are also considering small scale projects, with small communities. In particular, ICTs abilities are particularly low worldwide (Organisation for Economic Cooperation and Development, 2016). Thus, we would need to prepare Supervisors from non Computer Science domains to provide these practices. As an initial step towards this goal, we provided a capacitation course. Moreover, in these scenarios, interaction abilities of participants will be similar. We could expand the model to address large scale projects and communities in the future – for instance, adopting Hive (SALEHI; BERNSTEIN, 2018) for large scale collaborative distributed work. Regardless, we are limited by accessible tools for assets creations.

Finally, Lepi is a proof of concept and shall be improved over time. In this initial version, we have considered a subset of interaction alternatives of the audiences of our studies. As we expand the audiences, we will provide more alternatives to address more interaction needs and achieve broader inclusion. Additionally, as we expand the repertory of mechanics, it is probable that we will have to reduce the number of audiences who can create games per genre – at least if we think about individual creation. There is an alternative, though. If we think about individuals and communities (LIU; DING; GU, 2016b), as well as independence and interdependence (BENNETT; BRADY; BRANHAM, 2018), abilities from different people can contribute to remove barriers to enable co-creation and use. We can already do it with generated games (Figure 3); we could further explore it with creation tools.

2.10 Concluding Remarks and Current Work

With the growing importance of ICTs, being able to use and interact with technology becomes an essential ability for work, education, communication, and entertainment. Digital, Media, and Gaming Literacies reinforce this statement, affirming that ICTs contribute to acquisition of 21st skills and abilities. In this context, it is not enough to provide access to new technologies to achieve digital inclusion; we should aim to enable more

[64] For instance, with Data-Oriented Design (FABIAN, 2018).

people to create them as well.

This chapter presented our framework aiming to enable more people to create and play digital tailorable games. The framework provides a software architecture, a collaborative work model, and a game creation platform (Lepi). The software architecture decouples interaction from logic, enabling people to define how they will interact with a game at use time. The collaborative work model promotes collaborative creation of digital games based on abilities. People become co-creators of the solution, providing their own abilities, knowledge, and skills towards including their peers. Once someone is included, she/he may become a new contributor, continuing the process towards inclusion. The game creation platform combines the architecture and the work model into a game making tool to enable end-users to co-create their own games. As the platform implements itself the architecture, it is also able to modify its IO features; thus, we are able to extend it to include more people into creation activities.

Interaction requirements come from the diversity. Solutions can also come from diversity. With the framework, people collaborated to co-create digital games, helping each other in the process. Co-creation of inclusion became a community issue, which the community could also address. The community for the individual, the individual for the community. Contributions add over time, enabling new people to create and play, as well as improving the quality of the project to already included people.

As we compose interaction to games and systems, we can always include more people to create and play. Every new interaction alternative that we add can enable more people to participate, as well as improve usability for people who were already included. With the architecture, thus, accessibility can become an iterative, mutualistic, communistic, and collaborative process. Every new interaction feature contribute towards inclusion. Therefore, provided our results are right, the only limits to inclusion are what are computationally possible, existing IO technologies, our abilities to map interaction to human needs, and the game mechanics involved.

As we work with interaction alternatives in sets of features, this raises the question: provided that we had enough interaction alternatives, could we start moving towards accessibility suitable to the interaction needs of an individual? That is, could we combine interaction alternatives to support interaction needs of the specific person who is currently playing? With the architecture, we can always introduce alternatives for use[65]. As sensory abilities perceive information differently, we cannot ensure that every game can become universal; however, every game can become more accessible and inclusive to a broader public. For traditional software and some game mechanics, however, we could progress towards universal access for use and creation.

[65] More details in Appendix B.

We will continue expanding this project in two ways. First, we plan by enabling more people to create and play. We are refining our solution for current audiences, as well as exploring new ones. At this moment, we are implementing accessibility features in Lepi to support with people visual disabilities to create and play storytelling games. This way, we can expand our support to new input and output strategies (for instance, around the use of assistive technologies, text-to-speech, voice commands, and tactile input). As a result, people with different (dis)abilities will start creating content others with different (dis)abilities. This way, we move towards expanding the results from our evaluation with low literacy.

Second, we can start to introduce new game mechanics (and genres) over time to support more complex creations. Our goal is, initially, to explore non real-time mechanics first (as these will support broader inclusion) – from storytelling to exploration and role-playing mechanics. We will follow our strategy of implementing mechanics, considering interaction alternatives, and providing slots for creation. We provide the code, people provide interaction alternatives according to their abilities to provide content for IO components and event handlers. At the (probably distant) future, we shall be able to explore others EUD activities – or game design frameworks, such as Machinations (ADAMS; DORMANS, 2012) –, so people can start creating their own mechanics and technology.

Chapter 3

Able to Create, Able to (Self-)Improve: How an Inclusive Game Framework Fostered Self-Improvement Through Creation and Play in Alcohol and Drugs Rehabilitation

3.1 Introduction

Information and Communication Technologies (ICTs) are part of the 21st century life. In the same way that traditional literacy defined the abilities of reading and writing as essential for gathering knowledge and communicating, proficient use of ICTs becomes increasingly important, to the point of educators proposing modern types of literacies. Digital Literacy (DL), Media Literacy (ML), Gaming Literacy (GL), and Game Based Learning (GBL) provide examples and reasons to consider ICTs and games as ways of acquiring knowledge and culture (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012; EARP, 2015). GL, for instance, emphasizes that playing develops abilities related to computational thinking (CT), communication, and problem solving (GEE; TRAN, 2015; HAYES; GEE, 2010; MOUMOUTZIS et al., 2014; ROBERTSON, 2012).

For digital media, serious games brought digital playing activities to educational, professional, and healthcare contexts, for learning, training, and rehabilitation. Yet, even before serious games and GL, the benefits and applications of games extended beyond pure entertainment. Professionals from multiple domains often explored games on their activities. In fact, scholars such as Huizinga and Caillois argue that playing is so important for humans that it predates culture itself – games and playing are central activities in the formation of societies (HUIZINGA, 2016; CAILLOIS, 2001). From war to law, religion to arts, studying games can help to understand civilizations.

If games have literacy value, then, ideally, we should allow everyone to create and play them. Over time, tools, methods, and techniques (approaches, henceforth) have been lowering technical barriers and popularizing game creation – for instance, with modding and End-User Development (EUD) practices (HAYES, 2008; GEE; TRAN, 2015; BURKE; KAFAI, 2014; RESNICK; SILVERMAN, 2005).

There are guidelines and advice for EUD and for developing tools, highlighting the importance of collaboration, direct manipulation, gentle increases of programming complexity ("gentle slopes"), and incremental development practices (RESNICK; SILVERMAN, 2005; BURKE; KAFAI, 2014; AHMADI; JAZAYERI, 2014; JAZAYERI; AHMADI, 2011; PENA, 2011; UZUNBOYLU; BAYTAK; LAND, 2010, 2010; POOR, 2014; COMUNELLO; MULARGIA, 2015; HONG; CHEN, 2014; REPENNING; IOANNIDOU, 2006b). For storytelling, studies suggest focusing on story instead of technology, exploring visual programming languages, and converting story to computer code (de Leeuw et al., 2007; MARCHIORI et al., 2011; MARCHIORI et al., 2012; CARBONARO et al., 2008). There are also recommendations for exploring game creation as a tool for learning programming – for instance, exploring the principles of Constructionism (KAFAI; BURKE, 2015) –, acquiring GL (MOUMOUTZIS et al., 2014; MCARTHUR; TEATHER, 2015; LIN; CHIOU, 2010; El-Nasr; SMITH, 2006; MORAIS; GOMES; PERES, 2012; HAYES; GEE, 2010), end-user perception of EUD activities, (PANE; MYERS, 2006; BLACKWELL, 2006; LIEBERMAN; LIU, 2006; RADER; BRAND; LEWIS, 1997), and gender preferences for creation (DENNER; WERNER; ORTIZ, 2012; ROBERTSON, 2012; HAYES, 2008).

Currently, there are game creation approaches for professionals (usually teams consisting of designers, programmers, artists...), and for end-users (BURKE; KAFAI, 2014; KAFAI; BURKE, 2015; RESNICK; SILVERMAN, 2005; GEE; TRAN, 2015; EARP, 2015; WOODS; WOODS, 2015). The youth are an usual intended audience for end-user game creation (EARP, 2015).

However, to the best of our knowledge, studies regarding game creation focus on people without physical, cognitive, and emotional disabilities, living in favorable socioeconomic environments. For GBL, for instance, Earp (2015) noticed in a systematic review that only 6 out of 494 studies (1.21%) considered a public who were, at least, 23 years old, while 351 studies (71.05%) target children, youngsters, and teenagers. Earp (2015) also comments that most studies considering inclusion focus on "game making as a strategy for addressing the underrepresentation of girls and women in computing".

In our analysis of academic studies considering end-user game development, the most common audience are young anglophone people – normally children and adolescents of elementary and high schools in the United Stated of America (UZUNBOYLU; BAYTAK; LAND, 2010; AHMADI; JAZAYERI, 2014; BERLAND et al., 2011; CARBONARO et al., 2008; DENNER; WERNER; ORTIZ, 2012; El-Nasr; SMITH, 2006; HAYES, 2008; HAYES; GEE, 2010; IOANNIDOU; REPENNING; WEBB, 2009; JAZAYERI; AHMADI, 2011; KAUHANEN; BIDDLE, 2007; KESER et al., 2010; LIN; CHIOU, 2010; MCARTHUR; TEATHER, 2015; MORAIS; GOMES; PERES, 2012; MOTA; FARIA; de Souza, 2012; MOUMOUTZIS et al., 2014; PETRE; BLACKWELL, 2007; REA; IGARASHI; YOUNG, 2014; REPENNING, 2013; ROBERTSON, 2012; SMITH; CYPHER; TESLER, 2001). Communities of game modders – people who modify their favourite games – form a second popular audience for studies (POOR, 2014; POSTIGO, 2008; SOTAMAA, 2010). Although older (in the study performed by Poor (2014), the average age of participants was 31 years old), modders grew using ICTs and playing digital games.

Therefore, to the best of our knowledge, usual audiences for game creation and modding are youths ("digital natives") belonging to a normal distribution of abilities[1]. These are, thus, audiences composed by young people without physical, cognitive, and emotional disabilities. The more one person's abilities from these assumptions, the harder it is play games and ICTs. Thus, instead of reducing social and digital inclusion gaps, current approaches are contributing to exclude those who deviate from the average. Thus, age, socioeconomic grades, literacy, language and (dis)abilities are barriers we ought to overcome.

We can revert this situation, for we can create technology to foster inclusion. Instead of restricting and excluding, we should aim to enable and include, promote creation and consumption practices. Ideally, the inclusion should aim for the broadest possible audience as an end-goal – as proposed by the Universal Design (UD) (STORY; MUELLER; MACE, 1998).

Game accessibility provides resources to help developers to enable more people to play (YUAN; FOLMER; HARRIS, 2011; International Game Developers Association, 2004; ELLIS et al., 2013; BARLET; SPOHN, 2012; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; GARCIA; NERIS, 2014). Although these studies are currently aimed at developers, we could explore these resources to design game creation approaches to enable people with disabilities to make their own games.

To support this goal, we have been defining a framework to enable more people to create and play digital games, regardless of their (dis)abilities. Our lemma is "games *by* everyone, *for* everyone", for we are working towards universalizing related practices, lowering entry barriers, and exploring (dis)abilities as opportunities for improving

[1] The term average users (FISCHER, 2001; NERIS, 2010) describes users belonging to a normal distribution of abilities. This excludes, for instance, users with disabilities.

games for everyone[2]. The framework aims to extend modding with "accessibility modding": besides game content, people can contribute to create accessibility features and improve inclusion.

In our studies involving unusual (not yet considered) audiences (including people with low literacy and heterogeneous interaction needs)[3], we noticed evidence of the participants transforming themselves when they made their own games, shared it with their peers, and collaborated to improve their creations with what they knew and were able to do. By focusing on what participants were *able* to do[4], the activities boosted their self-esteem, inner trust, and self-value, for they perceived they were capable of doing and overcoming their previous limits. For this, abilities and disabilities became opportunities for improvements, both for themselves and their peers. From the union of the best abilities of each one, they created better systems for themselves and everyone else.

In particular, this workflow could benefit serious contexts such as education and healthcare. Healthcare professionals can already explore game playing as a rehabilitation tool for their patients to address mental, physical, and behavioral conditions (ATILLA, 2010; KHARRAZI; FAIOLA; DEFAZIO, 2009; KHARRAZI et al., 2012; PRESCHL et al., 2011). In Psychology, individuals express themselves with actions (CIALDINI, 2006). In a clinical scenario, doing is the highest degree of competence (MILLER, 1990). Competence starts with fact gathering (knowing), proceeding with interpretation/application (knowing how), demonstrating knowledge (showing), and performing in practice (doing). We can infer, therefore, that creating (doing) requires knowing, knowing how, and showing. From Psychology self-knowledge can contribute to psychotherapeutic treatment – Cognitive Behavioral Therapy (CBT), for instance, stimulates the acquisition of self-knowledge as means of acknowledging one's own values. ICTs and games can explore CBT as for rehabilitation, aiming to promote behavioral and self-esteem improvements, and provide motor and cognitive stimuli (MADER; NATKIN; LEVIEUX, 2012; RICHARDS; RICHARDSON, 2012).

Therefore, with suitable approaches, healthcare professionals could explore game creation as well as playing.

In this chapter, we describe how our framework helped participants who are currently digitally – and socially – marginalized to create and play games, including their peers and transforming themselves in the process. A group of EUD users (adults

[2] The lemma and our goals do not assume that everyone will be able to create and play every game. That is impossible. Rather, it means that we should always strive for inclusion and accessibility. Even when a system cannot become universal, it can always become more inclusive.

[3] As end-users (including people with disabilities and/or situations of vulnerability) participated actively in our study, we followed research ethics protocols throughout the entire processes. Certificado de Apresentação de Apreciação Ética from Plataforma Brasil: CAAE: 89477018.5.0000.5504.

[4] Instead of excluding participants from what they could not do.

in alcohol and drugs rehabilitation) used our framework as a support activity for their rehabilitation to create and play their own games. In the process, creation became a “sandbox”, on which participants developed games for self-expression and to share their experiences and knowledge to others. Ultimately, the framework enabled them to create and play for a greater end – it provided participants with opportunities for self-expression, learning, and growth. Nevertheless, this was only possible because they could co-create in the first place. Therefore, we provide a summary of design recommendations to enable inclusive creation based on the lessons that we have learned during this study.

3.2 A Framework Towards Games For Everyone, By Everyone

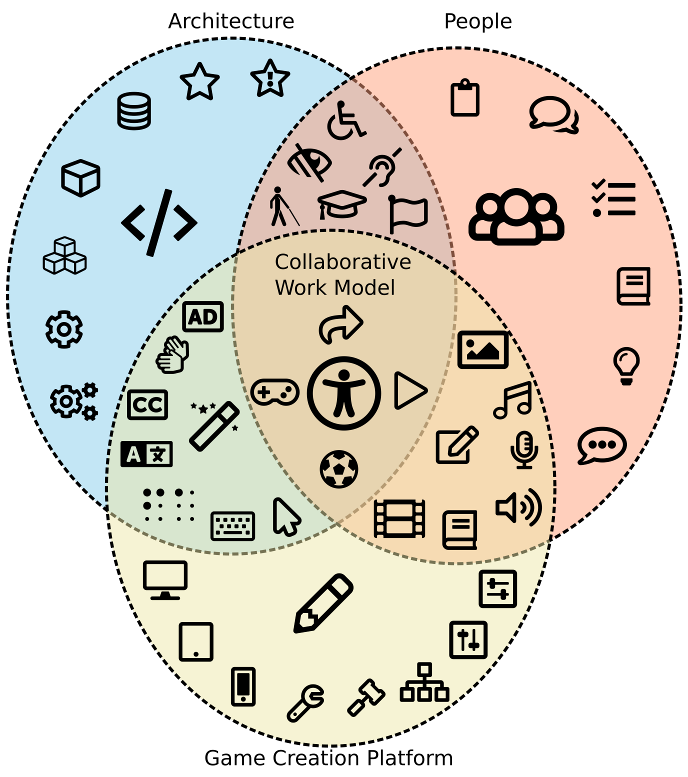
To enable more people to create and play digital games, we should expand our perspectives, focusing on game creation on people first, technology second. In this perspective, abilities, knowledge, skills, and wiliness can foster creation. This way, we find opportunities for inclusion and collaboration based on what people are able to do. As a result, people with disabilities can start contributing, co-creating, and playing games.

When we searched for EUD approaches which enabled end-users to make their own games, we noticed that they were designed for young anglophone students, without disabilities. Thus, although approaches existed, they were not yet suitable for everyone. In particular, interaction needs from our intended audiences – end-users with heterogeneous (dis)abilities who were not fluent at English – deviated from the usual EUD audience. Although the language barrier could be minimized, it would be hard to enable end-users to create accessible content for their games, as the approaches were not designed to support it. In particular, resulting games hard to be accessible as well – for as many audiences as possible. In particular, the needs from creators may be different from those of the players. Therefore, inclusion should encompass creation (game making) *and* play (resulting games).

We, therefore, started at one step below. Instead of using existing approaches (which could not ensure inclusive creation and play), we defined a framework aiming towards supporting inclusive creation of tailorable games. Our goal is enabling people with heterogeneous abilities to co-create and play games, creating accessibility alternatives towards inclusion as a community. Hence, the term “universal” refers to the context of UD. The lemma of our framework was, thus, “games *by* everyone, *for* everyone”, for we needed to support creation and play alike aiming at maximum inclusion.

We argue that a framework to support inclusive game creation could enable

Figure 14 – The pillars of the framework supporting people.



Source – Created by the author.

Note – The architecture provides a way to implement games without assumptions of how player will interact with the game. It allows for the implementation of abstract games from a combination of components (raw data illustrated as boxes and databases) and entities, events and event handlers (stars), and subsystems (engines). These games can explore different interaction alternatives to change how a player perceives and commands it, (re-)defining input and output (IO) interactions at use-time. Interaction needs originate from people ((dis)abilities, skills, literacy, language). Strategies to address then are implemented in the game creation platform (sign language, captions, audio-descriptions, internationalization, support for assistive technologies and automation). Content to fill the strategies can be created by people (from multimedia content). Once people (as a community) create interaction alternatives and provide content, they can enable people to play.

people to express their best abilities into creation. To overcome barriers, we aimed to support communication, collaboration, and inclusion; combined collective efforts to overcome individual needs. Thus, our framework was composed of three main pillars: a software architecture[5] (A), a collaborative work model[6] (M), and a game creation platform (L) (Figure 14).

The architecture enables developers to implement games for adaptation, based on semantics of use instead of physical-level actions (what a player can do, instead of how she/he will do it). Like a tailor, we create and adjust input and output (IO) interactions alternatives to suit the abilities of a player for commanding and perceiving the game. Our tailoring architecture allows arbitrary re-definition of human-computer interaction

[5] Further detailed in Appendix B.
[6] Further detailed in Appendix D.

at use-time[7]. As a result, interaction becomes similar to an add-on that provide plug-and-play accessibility to games. This allows us to consider interaction needs of one audience a time, iteratively, to enable creation and play. As we can combine any existing alternatives, we can promote accessibility for new audiences, and better usability and choice for included ones.

In particular, the architecture enables exploring a "one core fits most" version of a game, as well multiple accessible versions using the use-time tailoring approach. With the latter, one can combine existing accessibility features of the system to define custom ways of interaction based on her/his own needs. This merges Universal Design with accessibility, as both approaches become able to co-exist in a system. We called these result as tailorable games. Designers can create interfaces to suit the needs of the broadest extension of people, as well as custom interfaces for specific interaction needs. Users can choose and combine alternatives to define her/his best way of interacting with the system.

The collaborative work model explores the architecture for both game creation *and* accessibility modding, providing a dynamic and iterative process towards enabling more people to play from community collaborations. The community can provide access (with accessibility modding), as well as improve projects (with conventional modding). Thus, we can enable access to new audiences, and provide better experience of use, usability and interaction choices to those already included.

As a result, rather than excluded, people are yet to be included. Every member of the community can improve a project based on their own abilities, knowledge, and skills to create and improve game content *and* to provide accessibility features to enable play. As a result, included people can further co-create and enable new people to play, forming cycles of inclusion based on their abilities.

Finally, the game creation platform brings the architecture benefits to EUD practices. It promotes end-user creation of tailorable games, implementing the architecture itself and for resulting games. As a proof of concept, we have implemented Lepi for storytelling games. Lepi currently supports usual audiences, people with hearing disabilities (providing graphics and text content for all media, sign language support as videos) and people with low literacy (providing audio descriptions, large icons, voice audio recordings)[8]

[7] Like tailors, we adjust input and output (IO) interactions to suit abilities of a player (interaction as add-ons for plug-and-play accessibility). Our tailoring architecture allows use-time re-definition for perceiving and controlling games (for instance. text, audio, video, and/or graphics to convey content; controllers, assistive technologies, and/or automation to command).

[8] We are currently expanding Lepi to support to vision disabilities (with text-to-speech and voice input for simple commands), then we plan to support motor disabilities. First we include new audiences, then provide new mechanics, re-starting the process.

Figure 14 outlines relations between the framework pillars and people. The collaborative work-model is the inner part of the figure, as it merges people, the architecture and the game creation platform into a process to support end-user game creation. The other parts cooperate to enable co-creation of tailorable games. From the architecture and people, results interaction needs to address (for instance, disabilities). The goal is to translate game entities and components (from Entity-Component Systems (NYSTROM, 2014)), events and event handlers (from Event-Driven Architectures (NYSTROM, 2014)) into interaction alternatives. The intersection from the architecture and the game creation platform abstract this task into slots for accessible content (for instance, audio-description, closed-captions, translations, alternative input devices, graphical, aural or haptic effects). Slots representing alternatives to convey an information. The intersection between people and the game creation platform represent how people create and implement the alternatives. Once someone creates an alternative, she/he may add the artifact into its slot (from drawings, voice records, text, sign language videos, or other media). This way, the resulting game can combine assets to generate accessible versions for different audiences of players.

3.3 Fostering Self-Improvement Through Game Creation and Play With Adults in Alcohol and Drugs Rehabilitation

To evaluate and refine the framework outlined in Section 3.2, we conducted meetings focusing on end-user game creation and playing activities in a public healthcare service (a Centro de Atenção Psicossocial – Álcool e Drogas). The service catered to support the rehabilitation of adults with alcohol and drug addiction. Our meetings become an additional activity provided by the service to help its users.

3.3.1 Overview of the Meetings

We have performed ten meetings at the healthcare service, spanning four months. They followed a similar pattern:

1. They started with a brief description of the planned activities.
2. Next, we performed a complete cycle of the collaborative work model. The work model had eight phases:

   Conception idealization of the project;

   Conversion first prototype of the game;

Evaluation evaluation and guidance from the Supervisor;

Creation development of the game;

Enrichment improvement of game assets and usability, inclusion alternatives of accessibility alternatives;

Distribution generation of the playable games;

Use play sessions with other participants;

Conclusion end of the project.

The work model provided transient roles, defined according to what participants were performing at a given time. They could be Supervisors, Creators, Collaborators, and Players. In particular, Collaborators could act as Enablers – providing accessibility features to enable others to create and play –, and/or Enhancers – improving the quality of existing features to refine the game.

3. After the cycle, we discussed the activities and results with the healthcare professionals to gather feedback and inform them about the next meeting.

In the first meeting, we presented our project to the users of service, inviting them to participate. Recruiting and meetings happened at the service. The meetings happened at the healthcare service as a biweekly activity, lasting two hours each. We explained our goals (refine and evaluate the framework; attempt game creation to support therapy) for users of the service and provided a schedule of planned activities per meeting. Interested users of the service participated out their free will, without other incentives or rewards. Attendance was optional (both at the activity, as well as at the service itself). Our participants were recovering from alcohol and drugs abuse at the service. They were not hospitalized and they were not surrogates; they were receiving non-compulsory support (guidance to stop use and avoid lapses) at the service.

From the second meeting onward, participants interacted only with games and our game creation platform. As most participants had never used a computer before, the second and third meetings employed games to teach them how to use mouse and keyboard.

From the fourth meeting onward, participants stopped consuming existing games and started to create their own. They used Lepi, our game creation platform, and explored the work model to create, share, and play their games. Each meeting introduced new features and added complexity to the development process. This continued until the last two meetings, on which our participants presented their games to other users of the service, who had never participated in our activities.

3.3.2 Participants and Their Goals

Our team consisted of the two authors of this chapter (an MSc and PhD candidate, and a PhD in Computer Science) and five supporters who acted as Collaborators in the work model (an MSc and two undergraduates in Computer Science, and a PhD and undergraduate in Nursing). The service provided two healthcare professionals to support and monitor the meetings, who participated in the research as Supervisors. For the professionals, the meetings provided an additional strategy to perform their duty of aiding the users of the service. Finally, ten users of the service agreed to participate in our study as apprentices[9] – in the work model, they participated as Creators, Collaborators, and Players, according to phases of the work model and performed activities.

The participants were not from the usual intended audience of EUD game creation approaches. They were adults (29 years old or older) from lower socioeconomic grades, with no programming experience, low literacy skills (ranging from primary literacy skills in Portuguese, to not being able to read and write), and either basic computer skills (for instances, they were able to use office productivity suites) or never having used a computer before. Emotionally-wise, they presented a subset of characteristics of young drug addicts described by Rodrigues et al. (2015). Our participants were emotionally vulnerable and insecure, with anxiety, low self-esteem and performance, suffering from internal and external pressures (their own, from colleagues, from society), and sensing a lack of power and hope[10]. Thus, from (RODRIGUES et al., 2015), we would have to avoid situations involving failure during creation activities, as the participants could judge themselves as unable to new perform activities, blame themselves (instead of the tools), and give up.

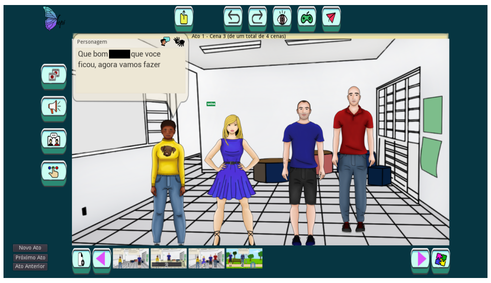
Figure 15 and Figure 16 illustrate some games and participants. Table 1 outlines motivations, needs, goals, and wishes of the participants and how the framework helped to achieve them. It provides a summary of the following subsections. The features and strategies in the table summarize how each pillar of the framework contributed to enable the participants to create and play games. Later, these strategies will be generalized into design recommendations from lessons learned in Section 3.4.

[9] As participation of the users in the service was not mandatory, their presence varied in each meeting. This was, actually, positive for the study, as it provided heterogeneity of proficiency using the game creation platform itself – it had to be suitable for new and old participants alike, at different phases of the creation process.

[10] Other characteristics mentioned by Rodrigues et al. (2015) (such as aggressiveness, impulsiveness, hyperactivity, oppositional defiant disorder) were not observed in our study involving older participants.

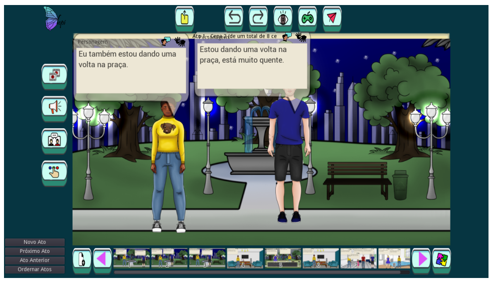
Figure 15 – Examples of three games created by our participants using Lepi.

(a) Participants meeting each other at a room in the service.



Que bom [redacted] que voce ficou, agora vamos fazer

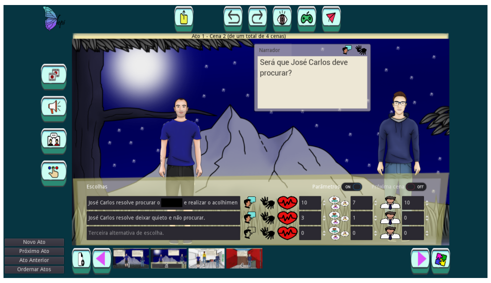
(b) Two participants meeting at a town square.



Eu também estou dando uma volta na praça.

Estou dando uma volta na praça, está muito quente.

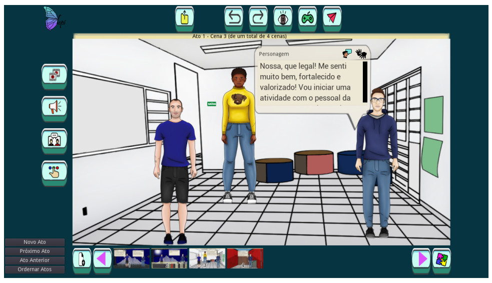
(c) Game to promote the healthcare service. A person invites his friend to seek aid in the service.



Será que José Carlos deve procurar?

José Carlos resolve procurar o [redacted] e realizar o acolhimen	10	7	10
José Carlos resolve deixar quieto e não procurar.	3	1	0
Terceira alternativa de escolha.	0	0	0

(d) Scene resulting from seeking aid in the service, following Figure 15c.

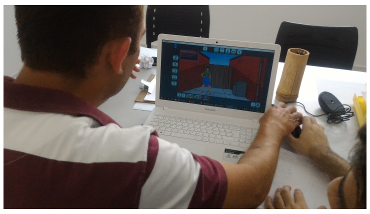


Nossa, que legal! Me senti muito bem, fortalecido e valorizado! Vou iniciar uma atividade com o pessoal da

Source – Created by the author.

Figure 16 – Participants creating and playing games over multiple meetings.

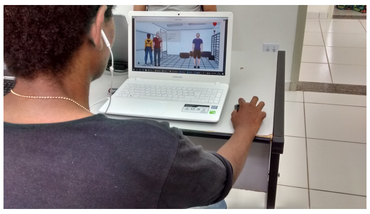
(a) P_2 creating his game.



(b) P_2 proud of his game.

(c) Participants creating their games.

(d) P_3 playing a game on which her image was not her.



(e) Supervisor helping a Creator.

(f) The creator from Figure 16e decided to dictate his story.

Source – Created by the author.

Table 1 – Summary of how the framework supported the goals of the participants.

Goal	Participants needs and/or wishes	Pillar(s)	Framework features/strategies
1.1	Be able to succeed	A, M, L	Increase creation complexity over time (M, L); avoid creation errors (L)
1.2	Learn how to use computers	L	Prefer easier interactions (L)
1.3	Create the game	A, M, L	Provide alternatives for creation (A, M, L); promote external aid with Collaborators (M)
1.4	Help friends to play the game	A, M, L	Provide alternatives for use (A, L); promote creation of accessible content (M) and external aid (Collaborator; M)
1.5	Understand what the game is presenting	A, M, L	Provide interaction alternatives (text, speech, sign) for play (A, L); promote creation of alternatives (M); explore aid for creation and play (M)
1.6	Be part (inside of) the game	M, L	Co-create image of participants (M); include images into game (L)
1.7	Be with friends in-game, in a common physic space	M, L	Co-create assets (M); inclusion of assets in game (L)
1.8	Show what was being learned in the service	M, L	Define game choices with different outcomes and scores (L); play and discuss games with domain experts (M)
1.9	Promote the healthcare service, removing negative stigmas	L	Co-create as marketing, game play as publicity (L)
1.10	Help players to develop character and values – especially kids	M, L	Co-create to teach (L); provide scores to provide feedback (L); play and discuss to learn (M)
1.11	Express ideas (share ideas in a different media)	M, L	Co-create for self-expression (L); supervision for evaluation (M)
1.12	Feel important, at least for a day	M, L	Co-create to self-express, demonstrate skills and knowledge, and demonstrate progress (L); validation from community use (M)
1.13	Learn with a game created by others	M, L	Play to experience creation from others, receive feedback from domain experts, and foster community discussion (M); scores to measure performance (L)
1.14	Understand what would happen after bad choices	M, L	Perceive outcomes from choices (L); receive advice, guidance, and support from main experts (M)
1.15	Share opinions, discuss similar experiences	M, L	Play to foster discussion (L); discuss with community and receive guidance from domain expert (healthcare professional) (M)
1.16	Share the game with friends (with different interaction needs)	A, M, L	Define interaction alternatives (M); include alternatives in project and export project as games with different accessibility features (A, L)
1.17	Share experiences and perceptions	M, L	Co-create to teach and self-express (L); share game to communicate with others (M)

Pillars: Architecture (A), Collaborative Work Model (M), Lepi (L)

Source – Created by the author.

3.3.3 Becoming Able to Create: Exploring the Framework to Suit Interaction Needs of the Participants

"I want to be able to do it". "I can do it, I will be able to.". When the activities started, three participants stated that the most important goal was being able to succeed (Goal 1.1 in Table 1). Overall, the participants wanted to have opportunities, to feel they were members of the society ("I hear in the streets: 'you can't do it, because you are an addicted, you are this, you are that'; no, we are not like that, we can do it"'). To define a path for the success, the framework needed to support interaction needs and skills enable participants to create and play games. This need was, thus, the ultimate goal of the study, requiring all three pillars of the framework. At a high-level, the architecture allowed to tailor the game creation platform (Lepi) to suit the interaction needs of the audience. The work model guided the creation process. Lepi enabled the creation of the software.

The process to achieve Goal 1.1 was incremental. It was built gradually exploring the work model, increasing complexity of activities over time. The initial step was enabling the participants to create digital games. Therefore, before any creation activities, participants needed basic computer skills – they had to learn how to use computers (Goal 1.2)[11]. Among the participants, there were people who had never used ICTs and digital games before. Some were scared of touching a mouse, with the fear of being unable to interact with the computer ("I never thought I would be able to use the computer; I had seen others using it, but never thought I would."); others did not have access to computers ("you know, it is a very good experience, because children in school have computers, and I would never have one."), nor could afford to do computing courses ("I never thought I would do a computing course in my life.").

To promote the first contact and provide basic training, participants played two games to develop basic mouse and keyboard skills at the second meeting. At the third, we introduced two additional games: a therapeutic game for elders with depression – composed of puzzle and action mini-games –, and a visual novel in Portuguese. As visual novels emphasizes the story over other mechanics, we found they were suitable for initial playing and creation activities. Storytelling activities are closer to traditional media – for instance, books, movies, and soap operas –, focusing on content. This way, our activities could start focusing on the creation on stories, introducing more complex strategies and game mechanics incrementally[12].

Once the participants had minimal proficiency with input devices, we proceeded to creation activities (Goal 1.3). The architecture of the framework enabled us to define different IO schemes to suit our participants, for creation and use alike (play)[13]. Lepi provided abstract commands for creation (for instance, adding media and characters, editing dialogues), for which we could implement physical-level interactions. In practice, we could define custom bindings for any available feature in the platform, for any input device we wanted to use. Likewise, Lepi supported graphical (image and video), textual, and aural output interchangeable[14].

As our intended audience had fine motor skills, the participants used mouse and keyboard to provide input (this is illustrated, for instance, in Figure 16d). The strategies to map abstract commands into the input devices varied according to the interaction needs of the participants. For instance, although some participants could not read nor write, they were able to listen, speak, and draw. Thus, there were two ways of overcoming

[11] Strictly speaking, this goal precedes the framework. For our purposes, this meant that the creation platform itself required simpler interactions.

[12] The focus on the story allowed us to explore interactive storytelling practices of increasing difficult over multiple meetings. We could, for example, start creation activities with linear stories, following with non-linear, branching stories afterwards. This way, we could explore "gentle slopes" within non-programming activities.

[13] The goal of the architecture is to allow users and developers to change IO interactions within the system. To avoid repetitions it is, hereafter, implicit in systems which required alternatives of use.

[14] Due to the architecture, we could add other input and output features, if needed.

the writing barrier: providing alternatives in Lepi, or exploring the work model for collaboration. This resulted into two approaches for content creation: independent and assisted. Independent meant that a Creator performed activities independently, on her/his own – this is how existing approaches work. Assisted meant that a Collaborator (acting as Enabler) helped the Creator to perform her/his activities. In the assisted approach, the Collaborator acted as a human-powered assistive technology, extending the Creator's abilities with her/his own to overcome interaction barriers (Figure 16f). Furthermore, the assisted approach works both for IO. In both cases, the Collaborator acts as an interpreter and/or mediator. For input, she/he translated commands from the Creator to the system. For output, she/he transformed content information from the System to the Creator.

With the two approaches, participants could create content based on their own abilities, skills, preferences, and goals. The participants stated that they enjoyed creation activities from the beginning ("creating is very good. It is an experience of my own life"; "it is, a new experience, a good experience."), and it was something that they looked forward to do at the service ("I don't mind missing the lunch, I want to create my game."; "I count the time for the workshop.").

For first prototypes, the Conversion phase of the work model varied. Strategies included to generate a first prototype included:

- Following traditional practices, typing stories and exploring the visual programming language offered by Lepi. This was the strategy of choice for participants who had previous experience with ICTs and could write well.
- Creating low-fidelity prototypes with paper and pencil containing drawings, sketches, graph-based schemes, or comic books. These were strategies adopted by participants who preferred to communicate visually. Participants who could write annotated their illustrations. Participants who could not write dictated the content to Collaborators.
- Creating low-fidelity prototypes via speaking, with voice recording or transcription. Participants who did not like to draw or preferred speaking followed this strategy. Collaborators transcribed the text for participants who could not write.

After Conversion, the healthcare professionals performed the Evaluation phase to analyze the projects. They advised the Creators on how to proceed in order to succeed – treatment wise. For instance, they asked questions to prompt reflections, requested the description of past experiences, provided alternative scenarios for thinking, and asked the Creators to explain their reasoning and rationale for story branches and their impacts to the Player's score. Game-wise, Creators addressed content-related requests at the Creation phase. For the most independent participants, the Conversion and Creation phases were similar, as they used had used Lepi from the start (their first prototype was a high-fidelity one). For assisted participants, the strategies to implement a high-level prototype (the game) varied to suit the preferences of each Creator. For characters, environments, scenes and story flows (transitions and branches), decisions (choices and consequences), they used the visual constructs of Lepi. For defining their story, participants opted to type them, record a voice narration of their voices, or asked a Collaborator to type for them. To each according to their abilities.

After further phases of Creation and Evaluation, they played each other games in the Distribution and Use phases. They could also play games that we created[15], which showcased further possibilities supported by Lepi. Following this process over meetings, some participants who, initially, followed the assisted approach started using Lepi on their own[16]. Participants who had permanent assistance on their initial uses started becoming more independent. Compared to their first games, their following creations were more complex, with longer stories and exploring more features and resources provided by Lepi.

3.3.4 Easing Creation, Enhancing Play: Accessible Platform to Creators, Accessible Games to Players

To better suit the interaction needs of the participants, we tweaked Lepi every meeting. New versions provided new features, improvements to accessibility and usability, and new default assets to explore. To improve accessibility and usability, changed included: alternative colors schemes (from darker to lighter); increased contrast; increased font sizes; types faces with better readability; short and objective text instructions (whenever possible); and large graphical icons. For input, we avoided interactions that participants had had difficult – such as holding to drag –, and offered large areas for selection. We also tried to provide input automation whenever applicable – for instance, for scene playback using audio. Automation avoided errors and minimized user input, reducing the chance of mistakes. This was helpful for the audience – Rodrigues et al. (2015) highlighted that people with drug addiction may have difficulty to concentrate. Therefore, we tried to keep the activities continuous, avoiding idle intervals, dispersion, and errors. This helped participants to maintain their concentration. In the same way, the presence of Supervisors and Collaborators was helpful, for Creators could ask and request assistance when they had doubts on how to proceed.

[15] In particular, the Collaborators with Nursing background created several games involving drugs and alcohol usage in daily activities.

[16] In some cases, Collaborators transcribed narrations or dictated characters, to allow Creators to type their stories.

Nevertheless, adapting Lepi's user interface satisfied only part of the requirements. To promote play, created games had to accessible as well, for the abilities of Players could be different from those of the Creators. This was the primary goal of the architecture, which we explored both for Lepi and for games created with Lepi. As a result, the IO interactions of resulting games could be changed at the Distribution phase to suit Players' needs at the Use phase.

For default assets included with Lepi, we offered images, textual description, speech narration, and sign language videos as output alternatives. For Creator made content, participants had to define and include their own alternatives. With the work model, this meant that Creators and Collaborators could provide alternatives according to their abilities and skills. At the healthcare service, Creators told their stories using text, speech, and drawings. In Lepi, stories were, often, added as text. As a result, participants who could not read could not play during the Use phase ("I see the story and options, but I can't read them."). This was a concern from participants in Goal 1.4 and Goal 1.5.

As the architecture and Lepi allowed inclusion and choice of interaction alternatives during use-time, the work model guided the process of including them into the creation process. We planned this for the Enrichment phase. In the Enrichment phase, a Supervisor may request Collaborators to improve the project, either refining game assets and usability, or request the inclusion of accessibility features and alternatives of use. For the healthcare service audience, the main request was providing speech content for the textual one, to allow participants who could not read to play the games without assistance. Between meetings, our team of Collaborators acted as Enablers to convert text into voice. The voice content was inserted back into the projects; enabling Players who could listen to play games from other participants on their own.

As each Creator developed their own game, we did not know how they would react to the inclusion of alternatives defined by others to their projects – it was an external collaboration. However, they greatly appreciated the inclusion, for they increased the quality of the projects, enhanced the playing experience, and allowed their friends to play their games[17]. In particular, some participants noted that they could record their voices and include the recording into their own games in the future, to avoid the situation.

Furthermore, in practice, following an assisted approach, a Collaborator can act as an Enabler at the Use phase to translate the content to those who cannot perceive it[18]. In particular, the roles in the work model are transient; participants can contribute based

[17] As one of our Collaborators knew basic Língua Brasileira de Sinais (LIBRAS), some games also received videos to enable people with hearing disabilities to play. Although none of the participants in the healthcare service needed sign language, they appreciated the inclusion, considering it was important and helpful.

[18] In the case of motor disabilities, she/he could also help the Player to play. In this scenario, the Enrichment phase could provide automation features to ease interaction with the game, or the inclusion of other input devices and/or assistive technologies for independent use.

on their skills and abilities, as well as the context of use. Thus, whenever the activity occurs in a shared environment (such as this service or a classroom), a Creator may become a Collaborator to provide the assistance. For example, in this study, a Supervisor requested that one participant read his story to help his friend who could not read (Figure 16b). This Creator, thus, became a Collaborator (Enabler) at that point, who assisted his friend. He provided an ability (reading) that was necessary to overcome the accessibility barrier, exploring another ability that he possessed (speaking) and one that his friend had (listening). At other meetings, participants acting as Players explored this strategy at Use phases whenever there was not an audio transcription of the story added to the project yet.

3.3.5 Being Part of the Game: Creators within the Creation

One of our participants (hereafter called P_1) was an outlier. Unlike the remaining participants, his interaction needs were similar to the usual audience for game creation solutions. Although P_1 had never programmed before, he used ICTs and played games daily. From the first meeting, P_1 had a unique wish: he wanted his "game character to be inside the game. He should look like me and walk like me!" (Goal 1.6). At this time, no other participant made the same request. To address the request within the framework's work model, we asked a Collaborator to create a game model representing P_1, which we added a base asset once it was completed.

At the fourth meeting, P_1's character model was not available as an asset yet. Thus, in the first round of creation, participants created their games with default characters. On the fifth meeting, however, we included his model as a new default asset. Once a friend (P_2) saw the model, he promptly – and happily – identified P_1 – "Look, it is P_1!". P_2 requested a character to represent himself as well. With both models, P_2 quickly perceived that he would be able to create games on which he and his friends were the characters. He, thus, requested that the healthcare service became an environment (a place) (Goal 1.7), as, this way, he could tell stories of the participants and the activities the performed at the service. Furthermore, P_2 perceived that he and his friends could "voice my characters to help my friends to play my game."

Huizinga (2016) defined game playing as being in a "magic circle"; P_2 was, at this time, the Creator of his own "magic circle" – in particular, one that he defined himself. Later, other participants requested their own models as well (Figure 15a). Ultimately, a healthcare professional and a participant co-created a game to promote the healthcare service (Goal 1.9). In their game, they described how the service truly worked, trying to demystify the public opinion about it, and having the approval of people who used the service (Figure 15c and Figure 15d).

3.3.6 Self-Improving with Games: Self-Expression Leading to Self-Knowledge

After several Evaluation phases, a healthcare professional affirmed that most stories related – directly or indirectly – to the participants' own lives. Although, in the first meeting, we encouraged that the participants shared either their own stories or fictitious ones (for instance, to share their experiences with alcohol and drugs), some quickly opted to portrait themselves in the stories.

The inclusion of their own images and the service's environment was a catalyst for this option. This awareness suggested that some participants were, already, exploring the framework to create games as mean of self-expression – something we were expecting to happen in the future. "I will do this story for me, because I want to live this moment, show that I can do it. I am able, because no one is born knowing." Moreover, we were not expecting participants to explicitly admit their own substance abuse in their stories from the start. They were honest about the facts (for instance, how they happened, what they had done, when it happened, with whom they were): "If you put what you are passing in the game, you can go much much further than you think.". According to professionals, even fictitious stories were related to participants' lives.

Professionals and collaborators helped participants to consider different ways to address situations (for example, other ways to act) as decisions in stories. By analyzing their past experiences and reflecting about them, participants started to identify other ways they could act from a given scenario. At first, many participants had difficult to define other ways to act other than how they had acted originally. Like traditional media, they were thinking linearly, creating linear stories with a single outcome – the real one, what happened to them. With the support work model, during Evaluation phases, healthcare professional started broadening options, suggesting Creators to consider different scenarios and ways to address a given situation. This way, they conceived alternatives and choices – from linear, to branching stories. According to a participant, this approached "helped us to think more about life", because "in any path, there is always a better exit."

As part of the creation process in work model, we asked the Supervisors to define attributes to the game – in this case, health, social relationships, and work. These would vary according to Players' choices when they played the game. Good choices lead to improvement in the attributes; bad choices lead to decreases. With multiple choices, Creators could compare prompts and outcomes to see how could they could affect their own lives[19]. Then, they would define a score for how suitable was their action. Although the choice was that of a character, the result was a consequence of the Player choice,

[19] This was easier to observe when the in-game character was the person herself/himself.

attributed based by the Creator's own judgment. For Players, scores provided means to track progress. For Creators, they provided another way to show and apply what they were learning at the service, as well as share this knowledge with others. This helped to materialize abstract ideas in quantifiable outcomes in Lepi (Goal 1.8). Outcomes based by decisions and results, with could foster reflection during creation and play ("people will learn a lot with my game, they can start thinking that they can re-start their life if they believe in it.").

With their games, creators could "can count experiences, learn good things, move always forwards, interact with friends." In this way, their creations were self-expressions of what they remembered, analyzed, and materialized from game elements within Lepi to make games (Goal 1.11). Their prompts for decisions were had consequences that they oversaw and judge according to fixed parameters ("creation helps to find paths, consider the consequences for each").

For Computer Science, the framework had sole goals of creation and playing. For serious activities, however, games became means instead of ends – for participants who act as Creators, the journey becomes more important than the resulting artifact. From enabling participants to make and collaborate, greater results may emerge. For instance, although participants recognized that they could share values and teach others (Goal 1.10), the creation aided them to learn as well ("creating games is great for participation and learning."). Participants applied the knowledge provided by the professionals in practice – even if they were unaware of it. Moreover, creation activities could provide therapeutic value themselves, as, according to a participant, "creation occupies the mind with fun. I avoid thinking bad things."

3.3.7 Sharing Knowledge: Learning and Experiencing from Playing

Still, the resulting game is an important artifact, as others can play it (Goal 1.16): "people can do what I did, see what I did, what I created (...) then the person can say 'I can change, it only depends on me, we can find a way for everything"'.

Since the first meeting, P_1 mentioned that game creating and playing could be good for his own treatment and of others. In the third, when he saw the novel, P_1 had an insight that his game could be a tool to bring awareness on the effects of drugs and alcohol abuse, and the content could advise players to avoid it based on his experience. He wished, thus, to share his game in schools, because, then, "kids can play my game and learn from my mistakes".

In the work model, the Supervisor decides who can play the game, as well as the interaction features that will be available. This step, thus, combines all three pillars to generate playable games that support the abilities of Players. Playing is part of phase of the work model (Use phase); it also has benefits. For the Creators, we observed that sharing their games made them proud (Goal 1.12): "knowing that someone will play my game is very good."; "it is very fun to show people my game; you keep thinking 'how is it possible to create a game and see your friends playing it?"'

Creators could teach their Players what they knew with the in-game content. They can show that they could do, how they overcome their disbelieves, insecurities, and fears. Creators made measurable progress overtime, as they could track how their games evolved – each new scene and inserted content showed progress. They observed this progressed both with the created technology and within their therapeutic practices at the service ("when I started, I wanted to learn many things, but I see that I learned a lot already.").

Players can lean from others what they shared (Goal 1.13). In special, games provide a safe environment for experimenting, meaning that even bad choices can lead to learning. Some players purposely made bad choices to discover their effects (Goal 1.14), comparing them to the better ones, seeing how their differed.

Furthermore, we observed that the participants adopted a new posture when showing their games to others. A participant who was initially scared to use the mouse taught a friend to use them to interact with her game. A second participant stated that "seeing my friends playing, for me, is an honour, because I could share the game with them here to see that they care about me". Another explained his reasoning to the scores in an activity where the healthcare professional acting as a Supervisor used the game with other participants. They discussed the game (Goal 1.15), shared similar experiences Goal 1.17. In this way, the Supervisor addressed multiple participants at once – the game was a tool for her professional practice, contributing towards her own goals.

3.3.8 Games Transforming Participants: Anecdotal Stories

Once again, this was evidence of changes and improvement. The participants were initially uncertain if they could create games; some of them had not even played digital games – or used a computer – before. Gradually, they became (visibly) more confident, skilled, and proud. At the end, some of them started teaching their peers how to play and improve their games. To illustrate some transformations over the meetings, we share the reactions one participant (P_2) in a meeting, whose experienced motivated another (P_3) to further participate (Figure 16a, Figure 16b). In a Use phase, a Supervisor asked P_2 to explain his game to the public (Goal 1.11, Goal 1.15, Goal 1.17). As there were participants who could not read in the room, she also requested P_2 to read his story aloud (Goal 1.5). P_2 started nervous and insecure about his creation, with his body language reflecting his low confidence ("I don't know...", "I am not sure..."). As he proceeded reading his story and answering questions, he – visibly and gradually

– started changing. His body language improved, he started to smile and laugh, and, at a certain point, his eyes started glowing. Upon peer acceptance (Goal 1.12), he was clearly proud and pleased with his creation, because it showed him it was successful (Goal 1.1; Figure 16b). In special, when we showed him the improved version of the game, he became delighted. After he interacted with a new version of his game with speech, he became enthusiastic about the changes, as they increased the quality of his game and allowed his friends to play the game without his help (Goal 1.3, Goal 1.5).

With this new version, a participant (P_3) shared her experiences with the public (Goal 1.17). P_3 provided her own insights and personal experiences about his story. P_2 and the Supervisor greatly appreciated her insights (Goal 1.8, Goal 1.13). It provided approval and appreciation for P_2 – who asked if he could present his game to the community in an upcoming soirée at the service. For the Supervisor, it offered new information regarding P_3, which she could explore to further help the participant.

After this experience, P_3 became motivated to create her own games, as she had not participated in creation activities before. "I want my kids to [play my game and] think, 'wait a second; my mother is making games, I can also do it'. It is very beautiful." She had never went to school, nor used ICTs – she was one of the participants scared of using computers and touching the mouse. P_3 saw P_1 and P_2's models, and requested her own as well (Goal 1.6). After a few meetings, she was using mouse and keyboard (Goal 1.2) to create her game (Goal 1.3) – sometimes typing text that she dictated to a Collaborator who, then, transcribed it to her, sometimes adding her voice to narrate (Goal 1.11). In all her stories, P_3 included herself, her friends, and the service's environment (Goal 1.7; Figure 15a and Figure 15b). As a Creator and a mother, she advised her friends to take better decisions on her stories (Goal 1.10, Goal 1.17), the same way her Supervisor advised herself (Goal 1.15). As a Player, she always chose the best options when prompted for decisions; she also played her own games several times (Goal 1.13, Goal 1.8). P_3 told about her games to her daughter, who could not believe her: "my daughter thought I was lying when I told her I created a game.". At a later meeting, when the healthcare professionals invited other users to play the games (Goal 1.16), P_3 started teaching others to play her game (Goal 1.4) – including her boyfriend. From scared of using computers, she taught other people how to use them.

Finally, with her own character, P_3's model also became an asset for creation (Figure 16d). In P_3's stories, her model was truly herself ("I wanted to put myself in the game to show that I could do it, I can be there, I can be anywhere, I can think what I should do, what I should not do."). In other stories from other participants, her model was someone else, with different names. Although it was her own image, it was not her. Every time she saw her image, she asked: "is this me?" She was an actress. She felt important for a day (Goal 1.12). She (and other participants) had succeeded (Goal 1.1).

Even though she had “never imagined I would be able to do it”, she perceived that she was able to do it, could further improve (“you can create lots of things there to show people.”), and have opportunities for social inclusion (“I had never had this many opportunities in life.”). This feeling was shared by other participants (for instance, “I could never imagine something like that could come out from my head.”), as well as by a healthcare professional. After the meetings, the professional stated that the game creation activity had “contributed [with therapeutic activities] by a enabling patients to have contact with technology, develop cognitive abilities and creativity, encourage game creation, [self-]identify with the activity, and provide opportunities to think about real or imaginary situations to learn behaviors to deal with daily situations of conflict [in real life], develop self-esteem during creation, and teach patients that they are responsible for their choices and [their] consequences.”

3.4 Design Recommendations from Lessons Learned

The framework aimed to enable inclusive co-creation of inclusive games. To achieve this, it provided:

1. A flexible software architecture that enabled use-time modification of human-computer interaction;
2. A collaborative work model to transform inclusion into a community problem, which the community could collaborate to address;
3. Tools to enable the workflow, accessible to a subset of interaction needs (usual audiences, hearing disabilities, and low literacy) and a single genre (storytelling) as proof of concept.

In this first study, we noticed that the framework contributed to the proposed workflow. With the tool, participants could co-create story-based games with their abilities. With the collaborative work model, contributors inserted media and content to address other interaction needs of the community. Due to architecture, participants could insert this new content to the game. This way, participants could co-create inclusion as a community. As proposed by Liu, Ding & Gu (2016a), we applied the communistic lemma “from each according to their abilities, to each according to their needs” to game creation.

The strategies outlined in Table 1 can be summarized into the following design recommendations:

1. Design for semantics of use. If we design for semantics of use, we do not assume abilities for use, and, therefore, we do not exclude people by design.

2. Implement for modification. We can re-shape human-computer interaction at-use time to enable people to perceive and command digital games according to their needs and abilities.

3. Provide different ways to create and play. Digital inclusion does not have to imply "one-size fits all solutions"; rather, a game may have the same rules, but play differently (multiple accessible versions as one game).

4. Compose interaction. We can provide interaction alternatives and (re-)combine them to define custom user interfaces to enable interaction. We can group alternatives into profiles aimed at particular interaction needs (for instance, for visual, hearing, or motor disabilities) – or allow people to define their own.

5. Focus on abilities and skills. People can always contribute based on their own strengths (abilities, skills, knowledge, interests, experience). Therefore, one important goal for co-creation is to identify and provide opportunities for contributions.

6. Foster community inclusion. Accessibility towards inclusion can be an iterative process, based on abilities.

7. Consider inclusion as a dynamic process. Once people are included and able to create, they can enable more people to use and/or create.

    - Corollary: people with disabilities can become contributors once included.

8. Foster community collaboration. People can teach, learn and benefit from each others strengths, and perceive how they, as a community, can achieve more than individually. This workflow benefits both sharer and receiver: they may feel empowered, important and valued, as well as creating bonds with each other. In serious contexts, domain experts can further benefit from community collaboration and discussion to provide advice, guidance, and feedback to their participants.

3.5 Concluding Remarks

DL, ML, and GL show the growing importance of technology for acquiring knowledge and developing important skills for the 21st century. However, although there exists approaches to enable people to create their own digital solutions, their intended audience are, currently, narrow — especially when we consider digital games. If games are a new form of literacy, enabling people to play and create are important steps towards digital and social inclusion. For this, reducing entry barriers and providing interaction alternative are important to enable a broader public to actively co-create.

In this chapter, we presented our experiences of exploring a framework towards universalizing the practices of creating and playing digital games with a unusual audience. We have explored inclusive game co-creation as an attempt to support therapeutic practices (assisting self-improvement) by adults in drug and alcohol misuse rehabilitation. Co-creation was supported by a framework for game accessibility towards accessibility modding. We presented outcomes from several workshops to demonstrate how the framework supported the participants' self-improvement.

Once able to create, our participants created games as activities as an additional tool to combat the addiction to alcohol and drugs, improving their lives in the process. With a combination of an architecture, a collaborative work model, and a game creation platform that, together, could suit interaction needs of the participants, the framework enabled participants to create their games. From the architecture, we could build creation tools and games able to modify IO interactions arbitrarily, at use time. From the model, we could guide end-users to create games collaboratively, potentially learning from (non-computing) domain experts in the process. From the game creation platform, participants created and shared their own games. We provided the framework and observed; the framework's approaches and the participants efforts, motivations, and collaborations made greater results emerge. During the process – their journeys –, the participants learned about the treatment, themselves, their friends.

Once able to create, they became able to self-improve performing the activities – and further learn and develop with the health of domain experts. Games became platforms for self-expression and self-growth. Participants created to teach and share what they knew, and played to experience and learn from others. Instead of an end, gaming became a journey, on which the participants acquired and shared knowledge and skills, crafting a game in the process. They learned and showed that they were capable of improving themselves, to create, to share and teach; during uncertainties, Supervisors were there to support them. The game was the artifact, their way of expression, their gift to others. It started from acceptance, believing in oneself that she/he would be able to do it. Next, working towards doing it, iterating to improving the creation, and continuously observing the progress towards the completion. Thus, for Creators, the main result was self-growth, improved confidence and hope, and the recognition that they are able to do whatever they put effort into, even if it seems impossible at first.

One is not restricted by what she/he cannot do; rather, her/his potential is limitless when she/he performs what she/he is able to do. Human-Computer Interaction should, therefore, provide technology to enable more people to create and use ICTs, regardless of (dis)abilities. Our framework is a step in this direction; we are currently working towards including new audiences – for instance, people with visual and hearing disabilities as creators and players –, as well as supporting new goals – for instance, digital co-creation as therapy.

Chapter 4

Conclusion

We have started this research with technology, we have finished it with people. Over the course of the research, we progressed from technology for developers (a software architecture), to collaborative co-creation and playing (with the collaborative work model and Lepi), stepping towards transforming lives

In Section 1.2, we have stated our research hypothesis as: "if end-users used creation tools suitable to their interaction needs and followed a collaborative work model to iteratively improve accessibility features to be inserted into a software architecture able to modify human-computer interaction at use-time, then they would be able create games satisfying heterogeneous interaction needs of possible players". Then we defined the three pillars for the architecture: the software architecture, the C3-IDG, and Lepi.

In Section 2.8, we discussed how each pillar was evaluated. In Chapter 3, we further described how the framework, as a whole, was evaluated to promote inclusive creation of inclusive games. For our study considering a subset of audiences, interaction needs, and mechanics, we observed that, with the Lepi, people could co-create story-based games with their abilities. With the collaborative work model, people inserted media, assets, and content to address other interaction needs of the community. Due to the architecture, people could insert this new content to the game. This way, people could co-create inclusion as a community – from each according to their abilities, to each according to their needs – and introduce their contributions in game projects.

We, therefore, argue that we have confirmed our hypothesis that people could co-create games for themselves and for people with different abilities and interaction needs, provided that they had suitable tools and a collaborative work model. More importantly, we noticed that creation and play can transform people, improve their lives and from their peers, promote learning, teaching and sharing, and bring fun and joy.

With the architecture, we can indefinitely enable more people to create and play.

The interactive Meta-Game (Meta-System) does already exist in there. It is up to us to map its commands to suitable input alternatives, and convert its raw data into output alternatives to convey information. We enable the first creators and players, they enable the next ones, defining cycles of inclusion towards universal access – as close as we can reach it.

The Game to the Player. The Game from the Player. The Game from the Community. The Game to the Community. An End-User Development Framework for Tailorable Games, towards games by everyone, for everyone.

4.1 Contributions and Results

We have outlined the main contributions and results of this thesis in Section 1.5.

With the framework,

- Due to the architecture, inclusion is restricted to currently available technologies, to what it is possible to implement with them, solve computationally in suitable time intervals, and how we can map interaction to the interaction needs of a user;
- Due to the co-creation models, inclusion can be extended iteratively and mutualistically by communities, based on abilities, skills, and knowledge of their members;
- Due to the co-creation tools, inclusion can be built and improve by end-users acting as non-professional developers.

The framework as a whole extends the state of the art by enabling the entire workflow. This results from individual contributions from its pillars, which extends the state of the art by:

- (Architecture, Models, Tools) Extending who can create and play digital games;
- (Architecture) Defining how to implement digital games (and systems) with composable interactions;
- (Architecture) Defining how to describe interaction with IO semantics and intents (from commands).
- (Models) Exploring communistic, mutualistic, and collaborative contributions from abilities, skills, and knowledge;
- (Models) Noticing that interdependence can promote independence in software;
- (Tools) Allowing end-user creation of MGs;

- (Tools) Abstracting inclusion as part of the development process.

With these contributions, we may revisit the quote from Aguado-Delgado et al. (2018), who affirmed that, currently, there were no initiatives to guarantee universal access in digital games. Although, in this thesis, we have not managed to change their statement, we can partially address their reasons.

With the framework described in this thesis, we avoided the first reason in two ways. For the "lack of quality in the generated/adapted video games", we have not assumed any user interfaces. For the "segregation of disabled players" we are able to continuously include more people into creation and play activities.

We address the second reason ("frameworks often depend on the use of specific technologies") when we became able to introduce new IO features in software. Assuming that the underlying technology implements the architecture, developers may become able to register new devices to define multiple user interfaces and ways to provide interaction – even if they had not planned for them before.

Finally, when end-users become able contribute, communities may be able to overcome omissions from the developers of the game ("guidelines, techniques, strategies, etc. that facilitate the development of accessible video games are ignored by many developers"). Thus, with the framework, we can explore a different approach towards universal access: coordinated collaborations from an active and empowered community, supported by a dynamic inclusion process with iterative contributions towards inclusion.

4.2 Final Remarks

Although this may be the end of a work, it is the start of a longer journey. We can define new tools (or extend Lepi) to enable the creation of more complex games with new mechanics. We can iteratively support more people to create and play digital games. We can explore the framework in serious contexts (education, healthcare, training), towards inclusive game creation for improvement.

Moreover, digital games provide many accessibility constraints for game development. Thus, they required a robust solution to support run-time tailoring. We can simplify and generalize a solution for traditional software. This way, we could further contribute towards digital inclusion and social inclusion.

We shall explore this route, because, once people can use and create, everyone is equal to a digital system.

4.3 Future Work

We outline a non-exhaustive list of possibilities below. As a possible heuristic to identify unique possible contributions, the reader may combine two or more research/theoretical frameworks and explore it within the models. Another possible heuristic is that we can extend the framework by depth (technology related; game design and programming complexity), or by breadth (human related; audiences to create and play; compatibility with assistive technologies).

1. Architecture:
    - Industry related – applied approaches:
        - Verify scalability for large scale scenarios;
        - Improve performance, reduce overheads;
        - Implement on other game development APIs, SDKs, engines, and frameworks;
            * Start by most popular / used first.
            * Ideal scenario: include the tailoring approaches themselves into game engines.
        - Collection of best programming practices;
        - Evaluate efforts of implementing the approaches versus traditional approaches;
        - Evaluate difficulty of learning the approach / the required practices.
    - EUD / MD related:
        - Define a design language to abstract the architecture;
        - Explore the architecture in other EUGD approaches.
    - UD / UA-games related:
        - Create a library of mechanics with pre-defined interaction alternatives;
        - Implement multiplayer support for people with different interaction needs;
        - Multiplayer:
            * Client-server (network/architecture) like;
            * As adaptations happen at local machines, each person can play her own version of the adapted game.
        - Support real-time games with limited time constraints;
        - Figure how we can benefit the most from assistive technologies.
        - Cognitive abilities:

        * Explore cheating as a form of assistance.
        * Propose allowed Meta-Game changes (warning: affects multiplayer games for all players):
            · Time can be an input outside the game.
            · Difficult can be parameterized in components.
    - Motor abilities:
        * Record runs of multiple players to provide variation for automation, if there is any other solution.
- Semiotics:
    - Explore communication using games – identify purpose of mechanics to find how designers use them to convey messages. This could help defining interaction alternatives per purpose (potentially a taxonomy).
- Artificial intelligence and machine learning;
    - Explore automation and input reduction to assist players (bots, pathfinding, auto-aim);
    - "Computer Assisted Gameplay";
    - "Computer Assisted Game Design";
    - "Computer Assisted Human-Computer Interaction".

2. Work model:
    - CSCW related:
        - Expand to support larger scale scenarios;
        - Define distributed workflows;
        - Explore existing CSCW models within the work model;
        - Implement the accessibility model for other domains.
        - Extend/modify the creation model other domains;
        - Find ways to foster collaboration.
    - MD / EUD related:
        - Identify more ways to enable people to contribute;
        - Define formats for enriched media alternatives;
        - Automated strategies for the Conversion phase.
    - UD / UA-games related:
        - Implement media creation tools for new audiences;
        - Collaboration workflows based on abilities;

    - Foster collaboration in heterogeneous teams, with different interaction needs;
    - Analyze relations of interdependence among participants.
  - HCC related:
  - PD related:

3. Lepi:
   - EUD / MD related:
     - Merge the solution into Godot, to benefit from its editor;
     - Support new mechanics and game genres;
     - Support new EUD activities (macros, PbD, VPL, scripting);
     - Support game design frameworks (for instance, Machinations);
     - Support alternative input devices and strategies.
       - Speech;
         - Single letter shortcut.
       - Modal editing (Vim like) for commands.
     - Explore non-verbal game creation / games without stories, mechanics only.
     - Advance on the 'gentle slope' creation method.
     - Define interaction adaptations per mechanic and genres.
   - CSCW related:
     - Explore accessible, collaborative and distributed creation of accessible games;
   - UD / UA-games related:
     - Explore game accessibility guidelines within Lepi – both for use and for creation;
     - Extend the solution to enable new audiences to create;
     - Support assistive technologies for input and output;
     - Improve functionality to cycle among interaction alternatives;
     - Define special playing modes to simulate disabilities;
     - Define automated player simulating common problems from a lack of interaction alternatives;
     - Automate game generation by player abilities at the Distribution phase;
   - Artificial intelligence and machine learning:
     - AI assisted EUGD;

- Recommendation of next steps based on history of actions;

4. Framework in general:

   - Practice:
     - Implement a Web system (or other support tool) to centralize practices;
       * Register participants by abilities;
       * Register skills to request Collaborators;
       * Showcase projects;
       * Match abilities and skills required to implement a missing interaction feature at a project to Collaborators who could provide it.
     - Version control system.
   - Inclusive Design / UD / Universal Access / HCC related:
     - Support systems to optimize workflows (Web system, for instance);
     - Content creation tools for people with disabilities;
     - Human perceptions regarding inclusive game creation and play;
     - Explore co-creation and cooperative / multiplayer play between people with different interaction needs – same MG for both, different Games for each;
     - Support cognitive disabilities;
     - Support multiple disabilities;
     - Investigate accessible game co-creation as culture;
     - Investigate accessible game co-creation in education, healthcare, rehabilitation, training;
     - Investigate accessible game co-creation for digital inclusion;
     - Investigate effects of inclusion in shared living environments over time.

5. Other ideas:

   - Theory of computation, artificial intelligence, and machine learning:
     - Although it is not needed for the definitions (as, in practice, as games for humans), for what games could we define a human-agent that can play the entire game?
       * How does NP-completeness affect the creation of such agent?
       * Many interesting games are NP-hard / NP-complete / PSPACE-complete.
   - Game design:
     - What are universal game genres / mechanics?
       * First Person Shooter (FPS);

    * Card games;
    * Board games;
    * Storytelling games / interactive fiction;
    * Text based games;
    * Slow paced games;
    * Casual games.
  - How does tailorable design affect traditional game design approaches?
  - Figure ethics of extending commercial systems;
  - Define an inclusive game creation curriculum for serious activities.
  - Define polymorphic mechanics.
- Programming:
  - Enforce the separation of concerns at the language level (same programming language, different ways to present it).
  - Design a programming language that represents code using different modalities (for instance, supporting textual and visual programming; perhaps expand it to support tactile).
- Compilers:
  - Implement a compiler that enforces the separation proposed in this thesis as a feature.
  - Explore "static-time" tailoring: a compiler that generates tailored versions to avoid the overhead and run-time memory and performance penalties.
- Security:
  - Explore risks and vulnerabilities from co-creation, sharing, and attaching interaction.
- Privacy:
  - Research risks for confidentiality, privacy, anonymity.

Furthermore, if we consider that people can apply the framework to non-Computer Science domains, the research opens possibilities for humanistic studies, for instance. Readers with different backgrounds may find nouveau opportunities that we have not – due to our technology bias.

Bibliography

ADAMS, E.; DORMANS, J. *Game Mechanics: Advanced Game Design*. 1 edition. ed. Berkeley, CA: New Riders, 2012. ISBN 978-0-321-82027-3. Cited once on page 103.

Aguado-Delgado, J. et al. Accessibility in video games: A systematic review. *Universal Access in the Information Society*, ago. 2018. ISSN 1615-5297. Cited 8 times on pages 33, 34, 46, 47, 133, 274, 277, and 278.

AHMADI, N.; JAZAYERI, M. Analyzing the learning process in online educational game design: A case study. In: *Proceedings of the Australian Software Engineering Conference, ASWEC*. [S.l.: s.n.], 2014. p. 84–93. ISBN 978-1-4799-3149-1. Cited 6 times on pages 50, 52, 53, 106, 107, and 279.

AHMADI, N.; JAZAYERI, M.; LANDONI, M. Helping Novice Programmers to Bootstrap in the Cloud: Incorporating Support for Computational Thinking into the Game Design Process. In: *2012 IEEE 12th International Conference on Advanced Learning Technologies*. [S.l.: s.n.], 2012. p. 349–353. Cited 4 times on pages 51, 52, 53, and 83.

AHMADI, N.; JAZAYERI, M.; REPENNING, A. Engineering an Open-Web Educational Game Design Environment. In: *2012 19th Asia-Pacific Software Engineering Conference*. [S.l.: s.n.], 2012. v. 1, p. 867–876. Cited 4 times on pages 50, 52, 53, and 279.

AHMADI, N.; JAZAYERI, M.; REPENNING, A. Performance Evaluation of User-created Open-web Games. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012. (SAC '12), p. 730–732. ISBN 978-1-4503-0857-1. Cited 2 times on pages 50 and 279.

ALATALO, T. An Entity-Component Model for Extensible Virtual Worlds. *IEEE Internet Computing*, v. 15, n. 5, p. 30–37, set. 2011. ISSN 1089-7801. Cited once on page 182.

ARCHAMBAULT, D. et al. Towards Generalised Accessibility of Computer Games. In: PAN, Z. et al. (Ed.). *Technologies for E-Learning and Digital Entertainment*. [S.l.]: Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, 5093). p. 518–527. ISBN 978-3-540-69734-3 978-3-540-69736-7. Cited once on page 181.

ATILLA, T. Video games in psychotherapy. *Review of General Psychology*, v. 14, n. 2, p. 141–146, 2010. ISSN 1939-1552(Electronic);1089-2680(Print). Cited once on page 108.

BARLET, M. C.; SPOHN, S. D. *Includification: A Practical Guide to Game Accessibility*. 2012. Cited 6 times on pages 33, 46, 107, 161, 277, and 317.

BASAWAPATNA, A. R.; REPENNING, A.; KOH, K. H. Closing The Cyberlearning Loop: Enabling Teachers To Formatively Assess Student Programming Projects. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2015. (SIGCSE '15), p. 12–17. ISBN 978-1-4503-2966-8. Cited 3 times on pages 50, 52, and 53.

BASAWAPATNA, A. R. et al. The Consume - Create Spectrum: Balancing Convenience and Computational Thinking in Stem Learning. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2014. (SIGCSE '14), p. 659–664. ISBN 978-1-4503-2605-6. Cited 4 times on pages 51, 52, 171, and 279.

BENNETT, C. L.; BRADY, E.; BRANHAM, S. M. Interdependence As a Frame for Assistive Technology Research and Design. In: *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2018. (ASSETS '18), p. 161–173. ISBN 978-1-4503-5650-3. Cited 12 times on pages 54, 55, 98, 101, 168, 270, 274, 280, 281, 287, 294, and 298.

BENNETT, C. L. et al. An Intimate Laboratory?: Prostheses As a Tool for Experimenting with Identity and Normalcy. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2016. (CHI '16), p. 1745–1756. ISBN 978-1-4503-3362-7. Cited 3 times on pages 54, 280, and 281.

BERLAND, M. et al. Programming on the move: Design lessons from IPRO. In: *Conference on Human Factors in Computing Systems - Proceedings*. [S.l.: s.n.], 2011. p. 2149–2154. ISBN 978-1-4503-0228-9. Cited 4 times on pages 50, 52, 53, and 107.

BERNARDO, C. G. et al. Multimodality by electronic games as assistive technology for visual disabilities. In: *2016 1st International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW)*. [S.l.: s.n.], 2016. p. 1–8. Cited once on page 46.

BIGHAM, J. P.; LADNER, R. E.; BORODIN, Y. The Design of Human-powered Access Technology. In: *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2011. (ASSETS '11), p. 3–10. ISBN 978-1-4503-0920-2. Cited 4 times on pages 274, 280, 281, and 287.

BLACKWELL, A. F. Psychological Issues in End-User Programming. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 9–30. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited once on page 106.

BOUISSAC, P. (Ed.). *Encyclopedia of Semiotics*. New York: Oxford University Press, 1998. ISBN 978-0-19-512090-5. Cited once on page 303.

BRADY, E.; BIGHAM, J. P. *Crowdsourcing Accessibility: Human-Powered Access Technologies*. Boston: Now Publishers Inc, 2015. ISBN 978-1-68083-034-7. Cited 4 times on pages 274, 280, 281, and 287.

BUEHLER, E. et al. Sharing is Caring: Assistive Technology Designs on Thingiverse. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2015. (CHI '15), p. 525–534. ISBN 978-1-4503-3145-6. Cited 2 times on pages 54 and 280.

BURKE, Q.; KAFAI, Y. B. Decade of Game Making for Learning: From Tools to Communities. In: ANGELIDES, r. C.; AGIUS, H. (Ed.). *Handbook of Digital Games*. [S.l.]: John Wiley & Sons, Inc., 2014. p. 689–709. ISBN 978-1-118-79644-3. Cited 4 times on pages 51, 55, 106, and 279.

BURLESON, W. et al. Game as life - Life as game. In: *Proceedings of IDC 2009 - The 8th International Conference on Interaction Design and Children*. [S.l.: s.n.], 2009. p. 272–273. ISBN 978-1-60558-395-2. Cited once on page 50.

CAILLOIS, R. *Man, Play and Games*. Reprint edition. Urbana: University of Illinois Press, 2001. ISBN 978-0-252-07033-4. Cited 4 times on pages 9, 42, 106, and 169.

Cambridge Advanced Learner's Dictionary & Thesaurus. *Framework*. [S.l.]: Cambridge University Press, 2017. Cited 2 times on pages 35 and 57.

CANO, A. R.; FERNÁNDEZ-MANJÓN, B.; GARCÍA-TEJEDOR, Á. J. Using game learning analytics for validating the design of a learning game for adults with intellectual disabilities. *British Journal of Educational Technology*, v. 49, n. 4, p. 659–672, 2018. ISSN 1467-8535. Cited once on page 46.

CARBONARO, M. et al. Interactive story authoring: A viable form of creative expression for the classroom. *Computers & Education*, v. 51, n. 2, p. 687–707, set. 2008. ISSN 0360-1315. Cited 5 times on pages 50, 52, 53, 106, and 107.

CARDONHA, C. et al. A Crowdsourcing Platform for the Construction of Accessibility Maps. In: *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*. New York, NY, USA: ACM, 2013. (W4A '13), p. 26:1–26:4. ISBN 978-1-4503-1844-0. Cited once on page 280.

CARVALHO, A. C. P. d. L. F. de et al. *Grandes Desafios Da Pesquisa Em Computação No Brasil - 2006 – 2016*. São Paulo, 2006. 26 p. Cited once on page 34.

CHAKRABORTY, J. et al. Designing video games for the blind: Results of an empirical study. *Universal Access in the Information Society*, v. 16, n. 3, p. 809–818, ago. 2017. ISSN 1615-5297. Cited once on page 46.

CHEUNG, G. Customization for games: Lessons from variants of Texas Hold'em. In: *Conference on Human Factors in Computing Systems - Proceedings*. [S.l.: s.n.], 2011. p. 1849–1854. ISBN 978-1-4503-0228-9. Cited 2 times on pages 51 and 52.

CHEUNG, M. *Therapeutic Games And Guided Imagery: Tools for Mental Health And School Professionals Working With Children, Adolescents, And Their Families*. [S.l.]: Lyceum Books, 2006. ISBN 0-925065-94-3. Cited once on page 42.

CHILTON, L. et al. Seaweed: A web application for designing economic games. In: *Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '09*. [S.l.: s.n.], 2009. p. 34–35. ISBN 978-1-60558-672-4. Cited 2 times on pages 50 and 51.

CIALDINI, R. B. *Influence: The Psychology of Persuasion, Revised Edition*. Revised edition. New York: Harper Business, 2006. ISBN 978-0-06-124189-5. Cited once on page 108.

CLIFFORD, R. et al. The Complexity of Flood Filling Games. *Theory of Computing Systems*, v. 50, n. 1, p. 72–92, jan. 2012. ISSN 1433-0490. Cited once on page 270.

COMUNELLO, F.; MULARGIA, S. User-generated video gaming: Little big planet and participatory cultures in Italy. *Games and Culture*, v. 10, n. 1, p. 57–80, 2015. ISSN 1555-4120. Cited 4 times on pages 51, 52, 106, and 279.

COOPER, S.; DANN, W.; PAUSCH, R. Alice: A 3-D tool for introductory programming concepts. In: *Journal of Computing Sciences in Colleges*. [S.l.]: Consortium for Computing Sciences in Colleges, 2000. v. 15, p. 107–116. Cited once on page 49.

CYPHER, A.; SMITH, D. C. KidSim: End User Programming of Simulations. In: *Conference Companion on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1995. (CHI '95), p. 35–36. ISBN 978-0-89791-755-1. Cited once on page 50.

DAHL, T. et al. A Virtual World Web Client Utilizing an Entity-Component Model. In: *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*. [S.l.: s.n.], 2013. p. 7–12. Cited once on page 182.

DARIN, T.; ANDRADE, R.; SÁNCHEZ, J. SLUP: A Standard List of Usability Problems in Multimodal Video Games designed for People Who Are Blind. *Anais Estendidos do Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais (IHC)*, out. 2018. ISSN 0000-0000. Cited once on page 46.

de Borba Campos, M.; OLIVEIRA, J. D. Usability, Accessibility and Gameplay Heuristics to Evaluate Audiogames for Users Who are Blind. In: ANTONA, M.; STEPHANIDIS, C. (Ed.). *Universal Access in Human-Computer Interaction. Methods, Techniques, and Best Practices*. [S.l.]: Springer International Publishing, 2016. (Lecture Notes in Computer Science), p. 38–48. ISBN 978-3-319-40250-5. Cited once on page 46.

de Leeuw, K. et al. A documental approach to adventure game development. *Science of Computer Programming*, v. 67, n. 1, p. 3–31, jun. 2007. ISSN 0167-6423. Cited 2 times on pages 50 and 106.

DEMAINE, E. D.; HOHENBERGER, S.; Liben-Nowell, D. Tetris is Hard, Even to Approximate. In: WARNOW, T.; ZHU, B. (Ed.). *Computing and Combinatorics*. [S.l.]: Springer Berlin Heidelberg, 2003. (Lecture Notes in Computer Science), p. 351–363. ISBN 978-3-540-45071-9. Cited 2 times on pages 167 and 270.

DENNER, J.; WERNER, L.; ORTIZ, E. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, v. 58, n. 1, p. 240–249, jan. 2012. ISSN 0360-1315. Cited 5 times on pages 51, 52, 53, 106, and 107.

DESURVIRE, H.; WIBERG, C. User Experience Design for Inexperienced Gamers: GAP—Game Approachability Principles. In: BERNHAUPT, R. (Ed.). *Game User Experience Evaluation*. Cham: Springer International Publishing, 2015, (Human–Computer Interaction Series). p. 169–186. ISBN 978-3-319-15985-0. Cited once on page 46.

EARP, J. Game Making for Learning: A Systematic Review of the Research Literature. In: *Proceedings of 8th International Conference of Education, Research and Innovation, ICERI2015*. Seville, Spain: IATED Academy, 2015. p. 6426–6435. ISBN 978-84-608-2657-6. Cited 7 times on pages 34, 42, 43, 53, 105, 106, and 279.

El-Nasr, M. S.; SMITH, B. K. Learning Through Game Modding. *Comput. Entertain.*, v. 4, n. 1, jan. 2006. ISSN 1544-3574. Cited 7 times on pages 41, 42, 51, 52, 53, 106, and 107.

ELLIS, B. et al. *Game Accessibility Guidelines: A Straightforward Reference for Inclusive Game Design*. 2013. Http://www.gameaccessibilityguidelines.com/. Cited 4 times on pages 33, 107, 161, and 277.

FABIAN, R. *Data-Oriented Design: Software Engineering for Limited Resources and Short Schedules*. [S.l.]: Richard Fabian, 2018. ISBN 978-1-916478-70-1. Cited once on page 101.

FERREIRA, J. J. et al. Combining cognitive, semiotic and discourse analysis to explore the power of notations in visual programming. In: *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. [S.l.: s.n.], 2012. p. 101–108. Cited once on page 50.

FISCHBACH, M.; WIEBUSCH, D.; LATOSCHIK, M. E. Semantic Entity-Component State Management Techniques to Enhance Software Quality for Multimodal VR-Systems. *IEEE Transactions on Visualization and Computer Graphics*, v. 23, n. 4, p. 1342–1351, abr. 2017. ISSN 1077-2626. Cited once on page 182.

FISCHER, G. User Modeling in Human–Computer Interaction. *User Modeling and User-Adapted Interaction*, v. 11, n. 1-2, p. 65–86, mar. 2001. ISSN 0924-1868, 1573-1391. Cited 2 times on pages 42 and 107.

FISCHER, G. End-User Development and Meta-design: Foundations for Cultures of Participation. In: PIPEK, V. et al. (Ed.). *End-User Development*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, 5435). p. 3–14. ISBN 978-3-642-00425-4 978-3-642-00427-8. Cited once on page 297.

FISCHER, G.; FOGLI, D.; PICCINNO, A. Revisiting and Broadening the Meta-Design Framework for End-User Development. In: PATERNÒ, F.; WULF, V. (Ed.). *New Perspectives in End-User Development*. Cham: Springer International Publishing, 2017. p. 61–97. ISBN 978-3-319-60291-2. Cited 2 times on pages 175 and 297.

FISCHER, G. et al. Meta-design: A Manifesto for End-user Development. *Commun. ACM*, v. 47, n. 9, p. 33–37, set. 2004. ISSN 0001-0782. Cited 2 times on pages 47 and 175.

FOGLI, D.; COLOSIO, S.; SACCO, M. Managing accessibility in local e-government websites through end-user development: A case study. *Universal Access in the Information Society*, v. 9, n. 1, p. 35–50, mar. 2010. ISSN 1615-5297. Cited once on page 280.

FONTANA, T. et al. How Game Engines Can Inspire EDA Tools Development: A Use Case for an Open-source Physical Design Library. In: *Proceedings of the 2017 ACM on International Symposium on Physical Design*. New York, NY, USA: ACM, 2017. (ISPD '17), p. 25–31. ISBN 978-1-4503-4696-2. Cited once on page 182.

FORTES, R. P. M. et al. Game Accessibility Evaluation Methods: A Literature Survey. In: ANTONA, M.; STEPHANIDIS, C. (Ed.). *Universal Access in Human–Computer Interaction. Design and Development Approaches and Methods*. [S.l.]: Springer International Publishing, 2017. (Lecture Notes in Computer Science), p. 182–192. ISBN 978-3-319-58706-6. Cited once on page 46.

GAMES, I. A. Gamestar Mechanic: Learning a designer mindset through communicational competence with the language of games. *Learning, Media and Technology*, v. 35, n. 1, p. 31–52, mar. 2010. ISSN 1743-9884. Cited 2 times on pages 42 and 49.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. ed. [S.l.]: Addison-Wesley Professional, 1994. ISBN 0-201-63361-2. Cited 6 times on pages 53, 54, 182, 183, 184, and 185.

GARCIA, F. E. *Um Estudo Sobre Diretivas de Design Para Audio Games*. Tese (Trabalho de Conclusão de Curso) — Universidade Federal de São Carlos, São Carlos, 2011. Cited 2 times on pages 46 and 97.

GARCIA, F. E. *Um Motor para Jogos Digitais Universais*. Tese (Dissertação) — Universidade Federal de São Carlos, São Carlos, 2014. Cited 28 times on pages 33, 34, 35, 42, 43, 46, 47, 57, 61, 93, 97, 160, 162, 175, 177, 178, 183, 211, 218, 219, 221, 266, 274, 275, 276, 277, 283, and 301.

GARCIA, F. E.; NERIS, V. P. d. A. Design de Jogos Universais: Apoiando a Prototipação de Alta Fidelidade com Classes Abstratas e Eventos. In: *Proceedings of the 12th Brazilian Symposium on Human Factors in Computing Systems*. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2013. (ICH '13), p. 82–91. ISBN 978-85-7669-278-2. Cited 5 times on pages 57, 97, 183, 276, and 283.

GARCIA, F. E.; NERIS, V. P. d. A. Design Guidelines for Audio Games. In: KUROSU, M. (Ed.). *Human-Computer Interaction. Applications and Services*. [S.l.]: Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, 8005). p. 229–238. ISBN 978-3-642-39261-0 978-3-642-39262-7. Cited 6 times on pages 33, 46, 97, 161, 181, and 277.

GARCIA, F. E.; NERIS, V. P. d. A. A Data-Driven Entity-Component Approach to Develop Universally Accessible Games. In: STEPHANIDIS, C.; ANTONA, M. (Ed.). *Universal Access in Human-Computer Interaction. Universal Access to Information and Knowledge*. [S.l.]: Springer International Publishing, 2014, (Lecture Notes in Computer Science, 8514). p. 537–548. ISBN 978-3-319-07439-9 978-3-319-07440-5. Cited 17 times on pages 33, 53, 54, 57, 61, 97, 107, 160, 162, 177, 178, 182, 183, 274, 276, 277, and 283.

GARCIA, F. E.; RODRIGUES, K. R. H.; NERIS, V. P. d. A. Uma Linguagem de Modelagem de Interação para Aplicações Terapêuticas. In: *Simpósio Brasileiro Sobre Fatores Humanos Em Sistemas Computacionais*. São Paulo: [s.n.], 2016. Cited once on page 59.

GEE, E. R.; TRAN, K. M. Video Game Making and Modding. In: *Handbook of Research on the Societal Impact of Digital Media*. Hershey, PA: Information Science Reference, 2015. p. 238–267. ISBN 978-1-4666-8310-5. Cited 5 times on pages 33, 41, 42, 105, and 106.

GIANNAKOPOULOS, G. et al. Accessible electronic games for blind children and young people. *British Journal of Educational Technology*, v. 49, n. 4, p. 608–619, 2018. ISSN 1467-8535. Cited once on page 34.

GRAMMENOS, D.; SAVIDIS, A.; STEPHANIDIS, C. Unified Design of Universally Accessible Games. In: *Proceedings of the 4th International Conference on Universal Access in Human-Computer Interaction: Applications and Services*. Berlin, Heidelberg:

Springer-Verlag, 2007. (UAHCI'07), p. 607–616. ISBN 978-3-540-73282-2. Cited 9 times on pages 33, 44, 46, 65, 66, 79, 95, 162, and 277.

GRAMMENOS, D.; SAVIDIS, A.; STEPHANIDIS, C. Designing universally accessible games. *Magazine Computers in Entertainment (CIE) - SPECIAL ISSUE: Media Arts and Games*, v. 7, p. 29, fev. 2009. ISSN 15443574. Cited 11 times on pages 33, 46, 47, 65, 77, 79, 162, 181, 232, 274, and 277.

GRAMMENOS, D.; SAVIDIS, A.; STEPHANIDIS, C. Unified Design of Universally Accessible Games. In: STEPHANIDIS, C. (Ed.). *Universal Access in Human-Computer Interaction. Applications and Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 4556, p. 607–616. ISBN 978-3-540-73282-2. Cited 11 times on pages 33, 44, 46, 79, 107, 162, 175, 180, 181, 232, and 277.

GREGORY, J. *Game Engine Architecture, Second Edition*. 2 edition. ed. Boca Raton: A K Peters/CRC Press, 2014. ISBN 978-1-4665-6001-7. Cited 12 times on pages 49, 53, 54, 61, 160, 162, 178, 182, 183, 184, 218, and 303.

HABICHT, H.; OLIVEIRA, P.; SHCHERBATIUK, V. *User Innovators: When Patients Set Out to Help Themselves and End Up Helping Many*. Rochester, NY, 2012. Cited once on page 281.

HARA, K.; LE, V.; FROEHLICH, J. Combining Crowdsourcing and Google Street View to Identify Street-level Accessibility Problems. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2013. (CHI '13), p. 631–640. ISBN 978-1-4503-1899-0. Cited once on page 280.

HAREL, I.; PAPERT, S. (Ed.). *Constructionism*. Westport, CT, US: Ablex Publishing, 1991. xi. ISBN 0-89391-785-0 (Hardcover); 0-89391-786-9 (Paperback). Cited 2 times on pages 42 and 50.

HAYES, E. Game content creation and IT proficiency: An exploratory study. *Computers & Education*, v. 51, n. 1, p. 97–108, ago. 2008. ISSN 0360-1315. Cited 5 times on pages 51, 52, 53, 106, and 107.

HAYES, E.; GEE, J. No selling the genie lamp: A game literacy practice in The Sims. *E-Learning*, v. 7, n. 1, p. 67–78, 2010. ISSN 1741-8887. Cited 8 times on pages 33, 41, 51, 52, 53, 105, 106, and 107.

HONG, R.; CHEN, V.-H. Becoming an ideal co-creator: Web materiality and intensive laboring practices in game modding. *New Media and Society*, v. 16, n. 2, p. 290–305, 2014. ISSN 1461-4448. Cited 4 times on pages 51, 52, 106, and 279.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. 2 edition. ed. Boston: Addison Wesley, 2000. ISBN 978-0-201-44124-6. Cited 3 times on pages 159, 269, and 283.

HUIZINGA, J. *Homo Ludens: A Study of the Play-Element in Culture*. Kettering, OH: Angelico Press, 2016. ISBN 978-1-62138-999-6. Cited 5 times on pages 42, 106, 121, 169, and 294.

HURST, A.; TOBIAS, J. Empowering Individuals with Do-it-yourself Assistive Technology. In: *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2011. (ASSETS '11), p. 11–18. ISBN 978-1-4503-0920-2. Cited 2 times on pages 54 and 280.

International Game Developers Association. *Accessibility in Games: Motivations and Approaches*. [S.l.], 2004. Cited 7 times on pages 33, 42, 46, 107, 161, 181, and 277.

IOANNIDOU, A.; REPENNING, A.; WEBB, D. Using scalable game design to promote 3D fluency: Assessing the AgentCubes incremental 3D end-user development framework. In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. [S.l.: s.n.], 2008. p. 47–54. Cited 5 times on pages 49, 50, 52, 53, and 279.

IOANNIDOU, A.; REPENNING, A.; WEBB, D. C. AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages & Computing*, v. 20, n. 4, p. 236–251, ago. 2009. ISSN 1045-926X. Cited 6 times on pages 49, 50, 52, 53, 107, and 279.

JAZAYERI, M.; AHMADI, N. End-user Programming of Web-native Interactive Applications. In: *Proceedings of the 12th International Conference on Computer Systems and Technologies*. New York, NY, USA: ACM, 2011. (CompSysTech '11), p. 11–16. ISBN 978-1-4503-0917-2. Cited 6 times on pages 50, 52, 53, 106, 107, and 279.

KAFAI, Y. B.; BURKE, Q. Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist*, v. 50, n. 4, p. 313–334, out. 2015. ISSN 0046-1520. Cited 2 times on pages 50 and 106.

KAHLER, H. et al. Computer Supported Cooperative Work: The Journal of Collaborative Computing. *Computer Supported Cooperative Work (CSCW)*, v. 9, n. 1, p. 1–4, mar. 2000. ISSN 0925-9724, 1573-7551. Cited 3 times on pages 46, 175, and 180.

KANE, S. K. Everyday Inclusive Web Design: An Activity Perspective. *Information Research: An International Electronic Journal*, v. 12, n. 3, abr. 2007. ISSN 1368-1613. Cited 2 times on pages 280 and 281.

KAUHANEN, M.; BIDDLE, R. Cognitive Dimensions of a Game Scripting Tool. In: *Proceedings of the 2007 Conference on Future Play*. New York, NY, USA: ACM, 2007. (Future Play '07), p. 97–104. ISBN 978-1-59593-943-2. Cited 5 times on pages 50, 52, 53, 107, and 279.

KESER, H. et al. Troubleshooting assessment: An authentic problem solving activity for it education. *Procedia - Social and Behavioral Sciences*, v. 9, p. 903–907, jan. 2010. ISSN 1877-0428. Cited 4 times on pages 50, 52, 53, and 107.

KHARRAZI, H.; FAIOLA, A.; DEFAZIO, J. Healthcare Game Design: Behavioral Modeling of Serious Gaming Design for Children with Chronic Diseases. In: JACKO, J. A. (Ed.). *Human-Computer Interaction. Interacting in Various Application Domains*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, 5613). p. 335–344. ISBN 978-3-642-02582-2 978-3-642-02583-9. Cited once on page 108.

KHARRAZI, H. et al. A Scoping Review of Health Game Research: Past, Present, and Future. *Games for Health Journal*, v. 1, n. 2, p. 153–164, abr. 2012. ISSN 2161-783X, 2161-7856. Cited once on page 108.

KITCHENHAM, B. *Procedures for Performing Systematic Reviews*. Keele, Staffs, 2004. 1–26 p. Cited 3 times on pages 36, 47, and 182.

KOH, K. H. et al. Early Validation of Computational Thinking Pattern Analysis. In: *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. New York, NY, USA: ACM, 2014. (ITiCSE '14), p. 213–218. ISBN 978-1-4503-2833-3. Cited 3 times on pages 50, 52, and 53.

LAGERSTRÖM, S. et al. Meta-designing interactive outdoor games for children: A case study. In: *ACM International Conference Proceeding Series*. [S.l.: s.n.], 2014. p. 325–328. ISBN 978-1-4503-2272-0. Cited once on page 50.

LANGE, P.; WELLER, R.; ZACHMANN, G. Wait-free hash maps in the entity-component-system pattern for realtime interactive systems. In: *2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. [S.l.: s.n.], 2016. p. 1–8. Cited once on page 182.

LASECKI, W. S.; KUSHALNAGAR, R.; BIGHAM, J. P. Helping Students Keep Up with Real-time Captions by Pausing and Highlighting. In: *Proceedings of the 11th Web for All Conference*. New York, NY, USA: ACM, 2014. (W4A '14), p. 39:1–39:8. ISBN 978-1-4503-2651-3. Cited once on page 280.

LI, Q.-C.; WANG, G.-P.; ZHOU, F. High-extensible scene graph framework based on component techniques. *Journal of Zhejiang University: Science*, v. 7, n. 7, p. 1247–1252, 2006. Cited once on page 182.

LIEBERMAN, H.; LIU, H. Feasibility Studies for Programming in Natural Language. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 459–473. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited once on page 106.

LIEBERMAN, H. et al. End-User Development: An Emerging Paradigm. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 1–8. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited 3 times on pages 34, 47, and 278.

LIN, H. Z. S.; CHIOU, G. F. Modding Commercial Game for Physics Learning: A Preliminary Study. In: *2010 Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL)*. [S.l.: s.n.], 2010. p. 225–227. Cited 5 times on pages 51, 52, 53, 106, and 107.

LIU, P.; DING, X.; GU, N. "Helping Others Makes Me Happy": Social Interaction and Integration of People with Disabilities. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. New York, NY, USA: ACM, 2016. (CSCW '16), p. 1596–1608. ISBN 978-1-4503-3592-8. Cited 10 times on pages 54, 126, 270, 274, 280, 281, 285, 287, 297, and 298.

LIU, P.; DING, X.; GU, N. "Helping Others Makes Me Happy": Social Interaction and Integration of People with Disabilities. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. New York, NY, USA: ACM, 2016. (CSCW '16), p. 1596–1608. ISBN 978-1-4503-3592-8. Cited 2 times on pages 101 and 168.

MACLAURIN, M. Kodu: End-user Programming and Design for Games. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. New York, NY, USA: ACM, 2009. (FDG '09), p. 2:xviii–2:xix. ISBN 978-1-60558-437-9. Cited once on page 49.

MADER, S.; NATKIN, S.; LEVIEUX, G. How to Analyse Therapeutic Games: The Player / Game / Therapy Model. In: HERRLICH, M.; MALAKA, R.; MASUCH, M. (Ed.). *Entertainment Computing - ICEC 2012*. [S.l.]: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, 7522). p. 193–206. ISBN 978-3-642-33541-9 978-3-642-33542-6. Cited 2 times on pages 42 and 108.

Mahelaqua et al. Community-oriented Spoken Web Browser for Low Iiterate Users. In: *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2013. (CSCW '13), p. 503–514. ISBN 978-1-4503-1331-5. Cited once on page 280.

MALINVERNI, L. et al. An inclusive design approach for developing video games for children with Autism Spectrum Disorder. *Computers in Human Behavior*, v. 71, p. 535–549, jun. 2017. ISSN 0747-5632. Cited once on page 46.

MANGIRON, C.; ZHANG, X. Game Accessibility for the Blind: Current Overview and the Potential Application of Audio Description as the Way Forward. In: MATAMALA, A.; ORERO, P. (Ed.). *Researching Audio Description: New Approaches*. London: Palgrave Macmillan UK, 2016, (Palgrave Studies in Translating and Interpreting). p. 75–95. ISBN 978-1-137-56917-2. Cited once on page 46.

MANKOFF, J.; HAYES, G. R.; KASNITZ, D. Disability Studies As a Source of Critical Inquiry for the Field of Assistive Technology. In: *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2010. (ASSETS '10), p. 3–10. ISBN 978-1-60558-881-0. Cited 3 times on pages 33, 276, and 282.

MARCHIORI, E. J. et al. A visual language for the creation of narrative educational games. *Journal of Visual Languages & Computing*, v. 22, n. 6, p. 443–452, dez. 2011. ISSN 1045-926X. Cited 2 times on pages 50 and 106.

MARCHIORI, E. J. et al. A narrative metaphor to facilitate educational game authoring. *Computers & Education*, v. 58, n. 1, p. 590–599, jan. 2012. ISSN 0360-1315. Cited 2 times on pages 50 and 106.

MAZAYEV, A.; MARTINS, J. A.; CORREIA, N. Improving Accessibility Through Semantic Crowdsourcing. In: *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-Exclusion*. New York, NY, USA: ACM, 2016. (DSAI 2016), p. 408–413. ISBN 978-1-4503-4748-8. Cited once on page 280.

MCARTHUR, V.; TEATHER, R. J. Serious mods: A case for modding in serious games pedagogy. In: *2015 IEEE Games Entertainment Media Conference (GEM)*. [S.l.: s.n.], 2015. p. 1–4. Cited 5 times on pages 51, 52, 53, 106, and 107.

MCDANIEL, R. G.; MYERS, B. A. Gamut: Demonstrating whole applications. In: *UIST (User Interface Software and Technology): Proceedings of the ACM Symposium*. [S.l.: s.n.], 1997. p. 81–82. Cited once on page 50.

MCDANIEL, R. G.; MYERS, B. A. Getting More out of Programming-by-demonstration. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1999. (CHI '99), p. 442–449. ISBN 978-0-201-48559-2. Cited 3 times on pages 50, 52, and 53.

MCSHAFFRY, M. L.; GRAHAM, D. *Game Coding Complete, Fourth Edition*. 4. ed. [S.l.]: Course Technology PTR, 2012. ISBN 1-133-77657-4. Cited 10 times on pages 49, 53, 54, 61, 160, 162, 178, 182, 183, and 218.

MEADOWS, D. H. *Thinking in Systems: A Primer*. White River Junction, Vt: Chelsea Green Publishing, 2008. ISBN 978-1-60358-055-7. Cited once on page 298.

MELONIO, A.; GENNARI, R. How to Design Games for Deaf Children: Evidence-Based Guidelines. In: VITTORINI, P. et al. (Ed.). *2nd International Workshop on Evidence-Based Technology Enhanced Learning*. [S.l.]: Springer International Publishing, 2013, (Advances in Intelligent Systems and Computing, 218). p. 83–92. ISBN 978-3-319-00553-9 978-3-319-00554-6. Cited once on page 181.

MILLER, G. E. The assessment of clinical skills/competence/performance. *Academic medicine: journal of the Association of American Medical Colleges*, v. 65, n. 9 Suppl, p. S63–67, set. 1990. ISSN 1040-2446. Cited once on page 108.

MORAIS, D.; GOMES, T.; PERES, F. Desenvolvimento De Jogos Educacionais pelo Usuário Final: Uma Abordagem Além do Design Participativo. In: *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2012. (IHC '12), p. 161–164. ISBN 978-85-7669-262-1. Cited 5 times on pages 51, 52, 53, 106, and 107.

MØRCH, A. Three levels of end-user tailoring: Customization, integration, and extension. In: *Computers and Design in Context*. [S.l.: s.n.], 1997. p. 51–76. Cited once on page 60.

MOTA, M. P.; FARIA, L. S.; de Souza, C. S. Documentation Comes to Life in Computational Thinking Acquisition with Agentsheets. In: *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2012. (IHC '12), p. 151–160. ISBN 978-85-7669-262-1. Cited 4 times on pages 50, 52, 53, and 107.

MOUMOUTZIS, N. et al. The ALICE experience: A learning framework to promote gaming literacy for educators and its refinement. In: *2014 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL)*. [S.l.: s.n.], 2014. p. 257–261. Cited 9 times on pages 33, 41, 50, 51, 52, 53, 105, 106, and 107.

NERIS, V. P. d. A. *Estudo e Proposta de um Framework para o Design de Interfaces de Usuário Ajustáveis*. Tese (Tese (Doutorado)) — Universidade de Campinas, Campinas, 2010. Cited 8 times on pages 35, 42, 43, 46, 107, 175, 179, and 180.

NERIS, V. P. d. A.; BARANAUSKAS, M. C. C. Interfaces for All: A Tailoring-Based Approach. In: FILIPE, J.; CORDEIRO, J. (Ed.). *Enterprise Information Systems*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Business Information Processing, 24). p. 928–939. ISBN 978-3-642-01346-1 978-3-642-01347-8. Cited 4 times on pages 35, 43, 274, and 276.

NERIS, V. P. d. A.; BARANAUSKAS, M. C. C. Designing tailorable software systems with the users' participation. *Journal of the Brazilian Computer Society*, v. 18, n. 3, p. 213–227, set. 2012. ISSN 1678-4804. Cited once on page 276.

NIEBORG, D.; VAN, D. G. The mod industries? The industrial logic of non-market game production. *European Journal of Cultural Studies*, v. 11, n. 2, p. 177–195, 2008. ISSN 1367-5494. Cited 3 times on pages 41, 51, and 52.

NYSTROM, R. *Game Programming Patterns*. 1 edition. ed. [S.l.]: Genever Benning, 2014. ISBN 978-0-9905829-0-8. Cited 2 times on pages 112 and 178.

OBRENOVIC, Z.; ABASCAL, J.; STARCEVIC, D. Universal accessibility as a multimodal design issue. *Communications of the ACM*, v. 50, p. 83–88, maio 2007. ISSN 00010782. Cited once on page 301.

ORENDT, E. M.; HENRICH, D. Design of robust robot programs: Deviation detection and classification using entity-based resources. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. [S.l.: s.n.], 2015. p. 1704–1710. Cited once on page 182.

Organisation for Economic Cooperation and Development. *Skills Matter: Further Results from the Survey of Adult Skills*. Paris: Organisation for Economic Co-operation and Development, 2016. ISBN 978-92-64-25804-4. Cited once on page 101.

OSSMANN, R.; ARCHAMBAULT, D.; MIESENBERGER, K. Computer Game Accessibility: From Specific Games to Accessible Games. In: *Proceedings of CGAMES 2006 Conference*. Dublin, Ireland: [s.n.], 2006. p. 104–108. Cited once on page 181.

OSSMANN, R.; MIESENBERGER, K. Guidelines for the Development of Accessible Computer Games. In: MIESENBERGER, K. et al. (Ed.). *Computers Helping People with Special Needs*. [S.l.]: Springer Berlin / Heidelberg, 2006, (Lecture Notes in Computer Science, v. 4061). p. 403–406. ISBN 978-3-540-36020-9. 10.1007/11788713_60. Cited once on page 181.

OWENS, T. Modding the history of science: Values at play in modder discussions of sid meier's civilization. *Simulation and Gaming*, v. 42, n. 4, p. 481–495, 2011. ISSN 1046-8781. Cited 2 times on pages 51 and 52.

PANE, J. F.; MYERS, B. A. More Natural Programming Languages and Environments. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 31–50. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited 2 times on pages 106 and 279.

PENA, J. Collaborative Framework for Browser Games Development. In: *Proceedings of the 2011 Workshop on Open Source and Design of Communication*. New York, NY, USA: ACM, 2011. (OSDOC '11), p. 65–72. ISBN 978-1-4503-0873-1. Cited 3 times on pages 50, 106, and 279.

PEREIRA, A. F. et al. Game accessibility guidelines for people with sequelae from macular chorioretinitis. *Entertainment Computing*, v. 28, p. 49–58, dez. 2018. ISSN 1875-9521. Cited once on page 46.

PERRONE, C.; CLARK, D.; REPENNING, A. WebQuest: Substantiating education in edutainment through interactive learning games. *Computer Networks and ISDN Systems*, v. 28, n. 7, p. 1307–1319, maio 1996. ISSN 0169-7552. Cited 3 times on pages 50, 52, and 53.

PETRE, M.; BLACKWELL, A. F. Children as Unwitting End-User Programmers. In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*. [S.l.: s.n.], 2007. p. 239–242. Cited 4 times on pages 51, 52, 53, and 107.

PIPEK, V.; KAHLER, H. Supporting Collaborative Tailoring. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 315–345. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited 2 times on pages 46 and 276.

PIPER, A. M. et al. SIDES: A Cooperative Tabletop Computer Game for Social Skills Development. In: *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2006. (CSCW '06), p. 1–10. ISBN 978-1-59593-249-5. Cited 2 times on pages 54 and 280.

PIPER, A. M.; WEIBEL, N.; HOLLAN, J. Audio-enhanced Paper Photos: Encouraging Social Interaction at Age 105. In: *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2013. (CSCW '13), p. 215–224. ISBN 978-1-4503-1331-5. Cited once on page 280.

POOR, N. Computer game modders' motivations and sense of community: A mixed-methods approach. *New Media and Society*, v. 16, n. 8, p. 1249–1267, 2014. ISSN 1461-4448. Cited 7 times on pages 41, 51, 52, 53, 106, 107, and 279.

PORTER, J. R. Understanding and Addressing Real-world Accessibility Issues in Mainstream Video Games. *SIGACCESS Access. Comput.*, n. 108, p. 42–45, jan. 2014. ISSN 1558-2337. Cited 4 times on pages 46, 274, 277, and 318.

PORTER, J. R.; KIENTZ, J. A. An Empirical Study of Issues and Barriers to Mainstream Video Game Accessibility. In: *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2013. (ASSETS '13), p. 3:1–3:8. ISBN 978-1-4503-2405-2. Cited 5 times on pages 46, 47, 274, 277, and 318.

POSTIGO, H. Video game appropriation through modifications: Attitudes concerning intellectual property among modders and fans. *Convergence*, v. 14, n. 1, p. 59–74, 2008. ISSN 1354-8565. Cited 4 times on pages 41, 51, 53, and 107.

POZZI, S.; BAGNARA, S. Individuation and diversity: The need for idiographic HCI. *Theoretical Issues in Ergonomics Science*, v. 14, n. 1, p. 1–21, jan. 2013. ISSN 1463-922X. Cited 3 times on pages 33, 273, and 276.

PRESCHL, B. et al. E-Health Interventions for Depression, Anxiety Disorders, Dementia and Other Disorders in Older Adults: A Review. *Journal of CyberTherapy and Rehabilitation*, v. 3, n. 4, p. 371–385, 2011. Cited once on page 108.

RADER, C.; BRAND, C.; LEWIS, C. Degrees of Comprehension: Children's Understanding of a Visual Programming Environment. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1997. (CHI '97), p. 351–358. ISBN 978-0-89791-802-2. Cited once on page 106.

REA, D. J.; IGARASHI, T.; YOUNG, J. E. PaintBoard: Prototyping Interactive Character Behaviors by Digitally Painting Storyboards. In: *Proceedings of the Second International Conference on Human-Agent Interaction*. New York, NY, USA: ACM, 2014. (HAI '14), p. 315–322. ISBN 978-1-4503-3035-0. Cited 4 times on pages 50, 52, 53, and 107.

REIN, P. et al. Group-Based Behavior Adaptation Mechanisms in Object-Oriented Systems. *IEEE Software*, v. 34, n. 6, p. 78–82, nov. 2017. ISSN 0740-7459. Cited once on page 182.

REPENNING, A. Excuse Me, I Need Better AI!: Employing Collaborative Diffusion to Make Game AI Child's Play. In: *Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames*. New York, NY, USA: ACM, 2006. (Sandbox '06), p. 169–178. ISBN 978-1-59593-386-7. Cited once on page 50.

REPENNING, A. Conversational programming in action. In: *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. [S.l.: s.n.], 2011. p. 263–264. Cited once on page 50.

REPENNING, A. Making programming more conversational. In: *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. [S.l.: s.n.], 2011. p. 191–194. Cited 3 times on pages 50, 52, and 53.

REPENNING, A. Conversational Programming: Exploring Interactive Program Analysis. In: *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. New York, NY, USA: ACM, 2013. (Onward! 2013), p. 63–74. ISBN 978-1-4503-2472-4. Cited 4 times on pages 50, 52, 53, and 107.

REPENNING, A.; AMBACH, J. The Agentsheets Behavior Exchange: Supporting Social Behavior Processing. In: *CHI '97 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 1997. (CHI EA '97), p. 26–27. ISBN 978-0-89791-926-5. Cited 2 times on pages 50 and 279.

REPENNING, A.; IOANNIDOU, A. AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D. In: *IEEE Symposium on Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006*. [S.l.: s.n.], 2006. p. 27–34. Cited 3 times on pages 49, 50, and 279.

REPENNING, A.; IOANNIDOU, A. What Makes End-User Development Tick? 13 Design Guidelines. In: LIEBERMAN, H.; PATERNÒ, F.; WULF, V. (Ed.). *End User Development*. [S.l.]: Springer Netherlands, 2006, (Human-Computer Interaction Series, 9). p. 51–85. ISBN 978-1-4020-4220-1 978-1-4020-5386-3. Cited 2 times on pages 106 and 279.

REPENNING, A.; IOANNIDOU, A. End-user Visualizations. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. New York, NY, USA: ACM, 2008. (AVI '08), p. 492–493. ISBN 978-1-60558-141-5. Cited once on page 50.

RESNICK, M. et al. Growing Up Programming: Democratizing the Creation of Dynamic, Interactive Media. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2009. (CHI EA '09), p. 3293–3296. ISBN 978-1-60558-247-4. Cited 2 times on pages 51 and 52.

RESNICK, M. et al. Scratch: Programming for All. *Commun. ACM*, v. 52, n. 11, p. 60–67, nov. 2009. ISSN 0001-0782. Cited once on page 49.

RESNICK, M.; SILVERMAN, B. Some Reflections on Designing Construction Kits for Kids. In: *Proceedings of the 2005 Conference on Interaction Design and Children*. New York, NY, USA: ACM, 2005. (IDC '05), p. 117–122. ISBN 978-1-59593-096-5. Cited 4 times on pages 51, 55, 106, and 279.

RICE, M. T. et al. *Crowdsourcing to Support Navigation for the Disabled: A Report on the Motivations, Design, Creation and Assessment of a Testbed Environment for Accessibility*. [S.l.], 2013. Cited once on page 280.

RICHARDS, D.; RICHARDSON, T. Computer-based psychological treatments for depression: A systematic review and meta-analysis. *Clinical Psychology Review*, v. 32, n. 4, p. 329–342, jun. 2012. ISSN 0272-7358. Cited once on page 108.

ROBERTSON, J. Making games in the classroom: Benefits and gender concerns. *Computers & Education*, v. 59, n. 2, p. 385–398, set. 2012. ISSN 0360-1315. Cited 8 times on pages 33, 41, 51, 52, 53, 105, 106, and 107.

RODRIGUES, K. et al. Personas-Driven Design for Mental Health Therapeutic Applications. *SBC Journal on Interactive Systems*, v. 6, n. 1, p. 18–34, out. 2015. ISSN 2236-3297. Cited 4 times on pages 59, 114, 119, and 290.

RODRIGUES, K. R. H. et al. Enriquecimento De Personas Para Apoio ao Design De Aplicações Terapêuticas Para a Saúde Mental. In: *Proceedings of the 13th Brazilian Symposium on Human Factors in Computing Systems*. Porto Alegre, Brazil, Brazil: Sociedade Brasileira de Computação, 2014. (IHC '14), p. 51–60. ISBN 978-85-7669-291-1. Cited 2 times on pages 59 and 290.

SALEHI, N.; BERNSTEIN, M. S. Hive: Collective Design Through Network Rotation. *Proc. ACM Hum.-Comput. Interact.*, v. 2, n. CSCW, p. 151:1–151:26, nov. 2018. ISSN 2573-0142. Cited 2 times on pages 101 and 281.

SALEN, K. Gaming Literacies: A Game Design Study in Action. *Journal of Educational Multimedia and Hypermedia*, v. 16, n. 3, p. 301–322, jul. 2007. ISSN 1055-8896. Cited once on page 42.

SAVIDIS, A.; STEPHANIDIS, C. Unified User Interface Development: The Software Engineering of Universally Accessible Interactions. *Universal Access in the Information Society*, v. 3, n. 3-4, p. 165–193, out. 2004. ISSN 1615-5289, 1615-5297. Cited once on page 274.

SAVIDIS, A.; STEPHANIDIS, C. Inclusive development: Software engineering requirements for universally accessible interactions. *Interacting with Computers*, v. 18, n. 1, p. 71–116, jan. 2006. ISSN 0953-5438. Cited 2 times on pages 181 and 274.

SCACCHI, W. Modding As a Basis for Developing Game Systems. In: *Proceedings of the 1st International Workshop on Games and Software Engineering*. New York, NY, USA: ACM, 2011. (GAS '11), p. 5–8. ISBN 978-1-4503-0578-5. Cited once on page 41.

SCHELL, J. *The Art of Game Design: A Book of Lenses*. 1. ed. [S.l.]: Morgan Kaufmann, 2008. ISBN 0-12-369496-5. Cited 4 times on pages 42, 50, 169, and 305.

SCHULER, D.; NAMIOKA, A. *Participatory Design: Perspectives on Systems Design*. Hillsdale, N.J.: L. Erlbaum Associates, 1993. ISBN 0-8058-0951-1 978-0-8058-0951-0 0-8058-0952-X 978-0-8058-0952-7. Cited once on page 297.

SELLERS, M. *Advanced Game Design: A Systems Approach*. 1 edition. ed. Indianapolis, IN: Addison-Wesley Professional, 2017. ISBN 978-0-13-466760-7. Cited once on page 298.

SHIRAISHI, Y. et al. Crowdsourced real-time captioning of sign language by deaf and hard-of-hearing people. *International Journal of Pervasive Computing and Communications*, v. 13, n. 1, p. 2–25, abr. 2017. ISSN 1742-7371. Cited 2 times on pages 54 and 280.

SIPSER, M. *Introduction to the Theory of Computation 2nd (Second) Edition*. [S.l.]: 2nd Edition, 2005. Cited 3 times on pages 159, 269, and 283.

SMITH, D. C.; CYPHER, A.; TESLER, L. Novice Programming Comes of Age. In: LIEBERMAN, H. (Ed.). *Your Wish Is My Command*. San Francisco: Morgan Kaufmann, 2001, (Interactive Technologies). p. 7–19. ISBN 978-1-55860-688-3. Cited 4 times on pages 50, 52, 53, and 107.

SMITH, J. D.; GRAHAM, T. C. N. Raptor: Sketching Games with a Tabletop Computer. In: *Proceedings of the International Academic Conference on the Future of Game Design and Technology*. New York, NY, USA: ACM, 2010. (Futureplay '10), p. 191–198. ISBN 978-1-4503-0235-7. Cited once on page 50.

SOTAMAA, O. When the game is not enough: Motivations and practices among computer game modding culture. *Games and Culture*, v. 5, n. 3, p. 239–255, 2010. ISSN 1555-4120. Cited 4 times on pages 51, 52, 53, and 107.

STORY, M. F.; MUELLER, J. L.; MACE, R. L. *The Universal Design File: Designing for People of All Ages and Abilities. Revised Edition.* [S.l.: s.n.], 1998. Cited 3 times on pages 107, 175, and 179.

TAKAGI, H. et al. Social Accessibility: Achieving Accessibility Through Collaborative Metadata Authoring. In: *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, NY, USA: ACM, 2008. (Assets '08), p. 193–200. ISBN 978-1-59593-976-0. Cited once on page 280.

TEKINBAŞ, K. S.; ZIMMERMAN, E. *Rules of Play: Game Design Fundamentals*. Cambridge, Mass: The MIT Press, 2003. ISBN 978-0-262-24045-1. Cited once on page 169.

TONON, K.; BAECKER, R. GameSoup: A Two-stage Game Development Environment. In: *Proceedings of the International Academic Conference on the Future of Game Design and Technology*. New York, NY, USA: ACM, 2010. (Futureplay '10), p. 255–256. ISBN 978-1-4503-0235-7. Cited 2 times on pages 50 and 279.

TORRENTE, J. et al. Evaluation of semi-automatically generated accessible interfaces for educational games. *Computers & Education*, v. 83, p. 103–117, 2015. ISSN 0360-1315. Cited 2 times on pages 33 and 277.

URBANEK, M.; GÜLDENPFENNIG, F.; SCHREMPF, M. T. Building a Community of Audio Game Designers - Towards an Online Audio Game Editor. In: *Proceedings of the 2018 ACM Conference Companion Publication on Designing Interactive Systems*. New York, NY, USA: ACM, 2018. (DIS '18 Companion), p. 171–175. ISBN 978-1-4503-5631-2. Cited 2 times on pages 34 and 46.

UZUNBOYLU, H.; BAYTAK, A.; LAND, S. M. A case study of educational game design by kids and for kids. *Procedia - Social and Behavioral Sciences*, v. 2, n. 2, p. 5242–5246, jan. 2010. ISSN 1877-0428. Cited 6 times on pages 51, 52, 53, 106, 107, and 279.

van Herk, R.; VERHAEGH, J.; FONTIJN, W. F. ESPranto SDK: An Adaptive Programming Environment for Tangible Applications. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2009. (CHI '09), p. 849–858. ISBN 978-1-60558-246-7. Cited 2 times on pages 50 and 279.

VOYKINSKA, V. et al. How Blind People Interact with Visual Content on Social Networking Services. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. New York, NY, USA: ACM, 2016. (CSCW '16), p. 1584–1595. ISBN 978-1-4503-3592-8. Cited once on page 280.

WESTIN, T. Inclusive Digital Socialisation : Designs of Education and Computer Games in a Global Context. 2017. Cited once on page 33.

WESTIN, T. et al. Game Accessibility Guidelines and WCAG 2.0 – A Gap Analysis. In: MIESENBERGER, K.; KOUROUPETROGLOU, G. (Ed.). *Computers Helping People with Special Needs*. [S.l.]: Springer International Publishing, 2018. (Lecture Notes in Computer Science), p. 270–279. ISBN 978-3-319-94277-3. Cited once on page 46.

WIEBUSCH, D.; LATOSCHIK, M. E. Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In: *2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. [S.l.: s.n.], 2015. p. 25–32. Cited once on page 182.

WOLBER, D. Pavlov: Programming by stimulus-response demonstration. In: *Conference on Human Factors in Computing Systems - Proceedings*. [S.l.: s.n.], 1996. p. 252–259. Cited once on page 50.

WOODS, C.; WOODS, C. The Rise of Interactive Game Development and Multimedia Project Creation Among School-Aged Children. In: *Society for Information Technology & Teacher Education International Conference*. [S.l.: s.n.], 2015. v. 2015, p. 1971–1975. ISBN 978-1-939797-13-1. Cited once on page 106.

World Wide Web Consortium. *Accessibility, Usability, and Inclusion*. 2016. Https://www.w3.org/WAI/intro/usable. Cited 3 times on pages 35, 42, and 276.

YILDIZ, S. et al. Design of a Game Community Based Support System for Cognitive Game Accessibility. In: BROOKS, A. L.; BROOKS, E.; VIDAKIS, N. (Ed.). *Interactivity, Game Creation, Design, Learning, and Innovation*. [S.l.]: Springer International Publishing, 2018. (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), p. 238–247. ISBN 978-3-319-76908-0. Cited 2 times on pages 34 and 46.

YOON, D.; KIM, K.-J. 3D Game Model and Texture Generation Using Interactive Genetic Algorithm. In: *Proceedings of the Workshop at SIGGRAPH Asia*. New York, NY, USA: ACM, 2012. (WASA '12), p. 53–58. ISBN 978-1-4503-1835-8. Cited once on page 50.

YUAN, B.; FOLMER, E.; HARRIS, F. Game Accessibility: A Survey. *Universal Access in the Information Society*, v. 10, n. 1, p. 81–100, mar. 2011. ISSN 1615-5289. Cited 22 times on pages 33, 42, 46, 47, 57, 61, 79, 107, 161, 164, 178, 181, 217, 247, 251, 255, 256, 259, 274, 277, 301, and 303.

ZYSKOWSKI, K. et al. Accessible Crowdwork?: Understanding the Value in and Challenge of Microtask Employment for People with Disabilities. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. New York, NY, USA: ACM, 2015. (CSCW '15), p. 1682–1693. ISBN 978-1-4503-2922-4. Cited once on page 281.

Appendix

APPENDIX A

This Thesis in Items

1. Although we cannot make every game universal, we should always strive for maximum accessibility and inclusion.
2. In potential, this thesis show that infinite accessible versions of a game can co-exist with the traditional one. We called this result as a tailorable game.
3. To include everyone (as many as possible), we implement for no one.
    - The Meta-Game to the machine.
    - The Game to the Player.
4. Run-time tailoring for full interaction re-mapping.
    - This thesis is very programming oriented, and considers games exploring our architecture as databases from which we transform data into information to enable play.
    - The algorithm allows defining custom Games to suit how someone needs to perceive and control it.
    - If we are able to inspect the data, extract information from it, and convey it to the Player, we can enable her/him to perceive a Game.
    - Likewise, if we can map commands to her/his abilities, we can enable her/him to control a game.
    - Finally, we can re-map interaction to remove time constraints from gameplay.
    - People play the same Meta-Game. However, the Game from one player can be completely different from a Game from another.
5. Computation Theory (SIPSER, 2005; HOPCROFT; MOTWANI; ULLMAN, 2000):

- Turing Machine: mathematical model that represents what machines can compute.
- Turing Completeness: ability of systems to simulate Turing machines.
- Any Turing Machine can simulate any other Turing Machine.
- Universal Turing Machines: Turing Machine that is able to simulate any other Turing Machine.
- Church-Turing thesis: any real-world computation can have an equivalent computation performed by a Turing Machine.
- In practice, any Turing Complete programming language is able to simulate an algorithm that any other Turing Complete language can simulate.
- The Halting Problem defines whether a computer program can finish in finite time.
- Complexity theory defines what can be solved by computers and how efficiently.
  - Complexity classes include NL, P, NP, PSPACE, EXPTIME, EXPSPACE.
  - We are concerned with problems up to the P complexity class, because they can be solved by Deterministic Turing Machines, and that can be solved with finite time (generally in the order of milliseconds, for games).

6. If our results are correct, we can think about games as Entity-Component Systems (ECS) and Event-Driven Architectures (EDA).

   - ECS decomposes games into entities and components; EDA decouples implementation with events and event handlers (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014; GARCIA, 2014).
   - If we limit the number of game elements, we reduce the scope to look for solutions.
   - This allows us to introduce new / remove existing features at use-time (run-time).
   - If we implement these architectures in a Turing Complete programming language following our approach, we can simulate any computation as event handlers and subsystems (fetching data from components stored on entities).
     - Therefore, we are not limited by technology.
     - In this thesis, the run-time tailoring algorithm enables changing how a game works at use-time.
     - We can, thus, create and insert anything that has been programmed so far – and will be programmed in the future – to software and games.

This means that we can introduce accessibility at the times when they are needed. We can start inserting many accessibility features (for instance, from guidelines (YUAN; FOLMER; HARRIS, 2011; International Game Developers Association, 2004; ELLIS et al., 2013; BARLET; SPOHN, 2012; GARCIA; NERIS, 2013b) and the collection from Game Accessibility Guidelines[1]) even if they were not previously there.

- In this thesis, we think of game elements in terms of:

Components (C) are structures to hold data for a purpose / responsibility.

- Components provide abilities to an entity.
- Components provide semantics to entity.
    - For instance, we can suffix the names of components with "able".
- If we want to know what an entity can do, we can query its components.

Entities (E) are sets of components.

- An entity is nothing by itself (it can be as simple an identifier, for example).
- An entity becomes able to do something once we attach a component that provide that ability to it.
- An entity becomes unable to do something once we detach the respective component from it.

Events (V) are structures to abstract that something of interest has just happened in the simulation.

Event Handlers (E) are subroutines to act once an event has happened.

- Events and event handlers define many-to-many relationships.
- An event can have multiple event handlers.
- An event handler can process multiple events (if applicable).

Subsystems (S) are system that processes components to do computation.

- A subsystem handles entities which have all components that concerns to it.

Commands (A) are events used to interact with a simulation.

- Commands represent intents for actions: *what* something wants to do, rather than *how* it will do.

Rules (R) express relations between entities; each Rule results into an event.

- Rules can represent mechanics, or small interactions between entities.
- The resulting event provides a callback that enable us to arbitrarily act once something has happened.

[1] <http://gameaccessibilityguidelines.com/>

- We can explore Data-Driven Architectures (DDA) (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014; GARCIA, 2014) to store customization and specializations to transform Meta-Games into Games. We can also explore input-mapping (or input re-mapping) (GREGORY, 2014; GARCIA, 2014) to provide players with options of devices and bindings for inputs. We store these preferences in Interaction Profiles (P).

7. We can fully decouple interaction from logic with a rigid separation of input and output (IO) concerns from the logic ones.

   Meta-Game (MG) $MG = \{C_{\text{Meta}}, V_{\text{Meta}}, R_{\text{Meta}}, S_{\text{Meta}}, A_{\text{Meta}}, E_{\text{Meta}}, H_{\text{Meta}}\}$

   Games (G) $G = \{C_{\text{Game}}, V_{\text{Game}}, R_{\text{Game}}, S_{\text{Game}}, A_{\text{Game}}, E_{\text{Game}}, H_{\text{Game}}, P\}$

8. The Meta-Game is abstract – it does not have human players.

   - It is a logic construct.
   - We can use the Unified Design (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007) to design it, then we can think on the elements of the architecture to refine the design and implement it.
   - The Meta-Game runs only using logical elements.
   - The Meta-Game only reacts to its own commands (events).
     - Commands express an intent: the wiliness to do something in the game.
     - As commands are events, their origin are irrelevant.

9. Games are concrete – they have human players.

   - They can be custom-tailored to suit abilities from Players.
   - Once we fully interact IO and we can introduce them to the game arbitrarily, at the correct times (events) and all the time (components), we can fully re-map interaction. We can generate infinite specializations of the same Meta-Game; however, they can be totally different in aesthetics and in how to interact.

10. If we think about interaction alternatives as sets of concrete elements, we can easily add and remove human-related interaction from a system using set operations (union, intersection, differences).

    - When we add concrete elements to the Meta-Game, it becomes a potentially playable Game.

- When we remove all concrete elements from a Game, it is reverted to its unique Meta-Game[2].

11. Interaction is, thus, composed/aggregated to the system.
    - We can convert a Meta-Game to any possible Game, and revert it back to the Meta-Game state. Then we can convert it to other possible Games.
    - The Run-Time Tailoring algorithm describes the steps to implement these transformations in software.
12. If we compose interaction, accessible can co-exist with conventional (as well as other variations – universal, inclusive...). We called this result as tailorable. For games, we can define tailorable games.
    - We can decouple interaction.
    - We can reshape (re-map, tailor) interaction.
    - We can alternate among versions.
    - Players can combine existing alternatives to make their own.
      - We can pre-define defaults by ability, they can tweak to make their own.
13. If we think of interaction as sets of interaction abilities, accessibility can become an iterative process – at best, towards universal access; at worst, towards maximum inclusion.
    - The logic construction (Meta-Game) already exists.
    - We can enable use of one audience at a time.
    - Can we provide interaction alternatives to enable human play?
      - Every time we can, we can enable a new audience to play.
      - All interaction alternatives sums to previous one.
    - Interaction alternatives are software and media.
      - We are only limited to what we can implement.
      - If we can implement a solution (and it finished in a suitable time interval for the game), we can define the alternative and introduce it to the system.
      - For games, solutions should be fast.
        - We can split computations into several frames (or specific threads) if needed.
    - If we can provide interaction alternatives to convey information and allow commands, we can progress with inclusion until universal access.

2 Technically, the Meta-Game is always contained into the Game.

    - We should enable people to perceive information and command the system.
  - Once again, although we cannot make every game universal, we should strive for inclusion.

14. Meta-Game: semantic simulation.
    - Logic components and events can generate semantics (*what*) to games.
    - Only handles entities that are essential to the simulation.
      - Aesthetic entities are not simulated.
      - If they have purposes for the logic (in other words, they have logic components), they are not aesthetic.
        - Position and world transforms are not always logic – if they only serve for feedback (for instance, visual, aural, of haptic effect), they serve for aesthetic purposes (although they do convey information, they are not required for the game logic).
    - IO components and event handlers can generate physical-level actions (/how) to games.
    - Events help designers to think about implicit happenings.
    - Rules can generate events to define "breakpoints" with the events – we can assume control of the execution at the right time that it is needed, either to tailor the solution or to inform the player what is happening.
      - Entity `Collide` Entity →`Collided`.
      - Entity `Walk` Entity →`Walked`.
      - Entity `Interact` Entity →`Interacted`.

15. Interaction model (implicit in this thesis and expanding the model from (YUAN; FOLMER; HARRIS, 2011); based on Semiotics and human senses): Perceive, Plan, Control, Wait.
    - Perceive: receive and request information.
      - How can we convert all this data into accessible information to the player?
      - How should we provide this information to the player?
        - What is the best way to provide this information for {vision, audition, olfaction, gustation, and somatosensation}?
          - · Different abilities require different strategies to convey the information.
          - · Once we acknowledge this fact, we can tailor the interaction in run-time to suit these needs.

        * If we can provide the data, we can fulfill this step.
    - Could we avoid overloading the player with information?
    - With entities, the player can request descriptions of what are around her/him, what they are, and what they can do (components), and how she/he can interact with them (components and commands).
- Plan: think about response and decide the play.
    - How can we help the player to think on (or plan) her/his play?
    - Can we provide heuristics?
    - Can we simplify the problem?
    - Can we modify the time constraints (from real-time to non-real time, for instance)?
- Control: provide response as command.
    - How can we enable the player to provide her/his command?
    - Can we provide partial automation?
- Wait: let the system process the command and determine its response.
    - How can we show the player that her/his command was accepted/rejected?
    - How can we provide immediate feedback?

16. Meta-Game: only essential entities for the simulation.
    - It is the simulation behind every interactive game.
    - The Meta-Game is a "game for machines".
    - It does not have to output information, because machines do not need output.
    - It only reacts to its Commands.
    - We abstract the player from the game to, at a later step, provide an accessible Game for him/her.
17. Games: sensory activities tailored to the needs of the player.
    - How should we map the abstract data and provide it to the player?
    - Output with components:
        - Constant feedback.
    - Output with events:
        - Instant feedback.
        - Constant feedback (start/end pairs).
    - What components and event handlers should we create and attach to the Meta-Game to create enable play?

    - Accessibility can become a plug-and-play feature of games.

- We can provide any aesthetic features and user interfaces that we want.
- Once an event happen (for instance, due to a Rule), we can insert anything that is computationally possible to enable play. For instance, we can define a custom user interface that appears to change how the game play.
- Alternatively, we can define input components and subsystems that monitor the Meta-Game to provide automation.
- We can think of time as an input to the Meta-Game. This way, we can even remove time constraints from real-time interactive systems.
    - For instance, we could assume a sequence of:
      Task →Command `Do Something` →New Task.
    - We could change it to something like:
      Command `Pause` →Sub Task 1 →Local (non Meta-Game) command →Sub Task 2 →Local (non Meta-Game) command →…→Sub Task Final →Command `Resume` →Command `Do Something`.
      As the Meta-Game is paused and the local commands do not concern it, it only reacts to the Commands `Pause`, `Resume`, and `Do Something`. We converted a real-time interaction into one with longer (possible infinite) time for interaction.
      Players who could play in real-time do not use the components, event handlers, and subsystems to re-map the interaction. People who need it may become able to play when they are introduced. Players who think they provide a fun effect (or want to try something new) may also use it after it was provided as an interaction alternative.
- As we can introduce any IO to a Game, we can always make it more fun.

18. Commands: semantic input to express intents.

- Commands provide ease of automation.
- Commands provide ease of exploring devices (mouse, keyboard, controller, voice, assistive technologies) for input.
    - If we can map a device (and its buttons, axis, triggers…) to a command, we can use it to enable play.
- Commands can be automated by algorithms, to assist the player.
- Commands API: macros/combinations of commands and logic to define higher-level commands to assist players.
    - `Move` →`Go To Entity`.

    - `Attack` →`Attack Entity`.
    - `Enter` →`Enter Entity`.

19. Games that can become simulations can *probably* become universal.

    - Automation (in-game, analyzing the game states) can contribute to it.
    - If the player cannot provide the input, the system can send a command to play on her/his behalf.
    - Partial automation can assist players (input reduction).
        - We can provide partial automation from traditional game middleware solutions.
            * Pathfinding.
            * Raycasting.
            * Interpolations.
            * Bots: Aiming / moving.
    - AI and machine learning could enable playing patterns for predicting next moves over time.

20. Full automation is outside the P complexity class for most (any?) non-trivial game (for instance, Tetris is NP-complete (DEMAINE; HOHENBERGER; Liben-Nowell, 2003)).

    - There are alternatives.
        a) Small scale (partial) automation.
            - Game Artificial Intelligence (AI) as assistance to enable play.
            - The game provides heuristics and lets the player determine the response.
            - AI-assistance can guide players.
                * AI as a "co-pilot".
        b) People helping each other to play.
            - Cooperative play – players share the same entity to combine their abilities and enable play.
            - Human as a "co-pilot".

21. As interaction is composed, we can adapt it over time.

    - Run-time tailoring.
    - Players can change it.
    - We could monitor the game to adapt it.

- We can define custom user interfaces to suit different abilities (reshape, remap, tailor the interaction).
- With creativity, we can transform some real time interactions into non-real time ones.

22. Interaction alternatives can be created by communities instead of being the sole responsibility of developers.

    - If we decouple and compose interaction alternatives, we can make accessibility a collaborative and iterative process.

23. Individuals and Communities: the individual for the community, the community for the individual.

    - Communistic interactions (LIU; DING; GU, 2016b):
      - "From each according to their abilities, to each according to their needs".
    - Independence and interdependence (BENNETT; BRADY; BRANHAM, 2018).
      - People can help each other.
      - "Cross-ability cooperation": the abilities from one person can complement the abilities from another.
    - Mutualistic contributions:
      - As we are considering sets of interaction alternatives, every new addition adds to every previous one. A new alternative may enable more people to play. For people who can already play, it provides more choices – which may lead to better usability, improvements for use experience, and even new ways to interact.
      - Accessibility can have a compound effects. Features that enable use for some users can enhance usability and quality interaction for others.

24. With sets of interaction alternatives, inclusion can be transient.

    - Transient roles: abilities and inclusion may change over time.

      Included Person for whom the existing interaction alternatives enable use.

      To Be Included Person for whom existing interaction alternatives are not yet enough to enable use.

      Collaborator Person who can contribute to improve the system.

      - **Enabler** Person who can provide an interaction alternative to enable use.
      - **Enhancer** Person who can improve existing interaction alternatives to improve usability and experience of use.

- Included people can become Contributors – abilities and skills are what truly matter.
- People with disabilities can become Collaborators once they are Included.
- Even imperfect contributions can lead to inclusion.
    - Once people are included, they can act as Enhancers to improve former contributions.

25. Once an interaction alternative becomes part of a game, we can transition gradually from interdependence to independence (for the public that it was designed).
    - When independence is not possible, co-creation, co-use, and cooperative play are perfectly valid strategies to enable use and creation.
    - Independence and interdependence can co-exist.
26. Game design is a multi-domain discipline.
    - Every area of human knowledge can contribute in game design (CAILLOIS, 2001; SCHELL, 2008; HUIZINGA, 2016; TEKINBAŞ; ZIMMERMAN, 2003). Therefore, everyone can contribute.
    - Every piece of knowledge can lead to the construction of interesting games.
    - People can provide technology, aesthetics, story, and/or mechanics to create games (SCHELL, 2008).
27. Towards universal access, we need to enable creation and use alike.
    - Once people can create and use, they are equal to any other user of any digital system.
    - If we implement for semantics of use and for modification, without assuming abilities or fixed ways interact with a system –, we can move towards including everyone.
        - We should design and implement *for* adaptation.
    - If we enable one person / group to use and create, they can enable more people to use and create.
    - We can form cycles of inclusion once people can co-create digital systems.
28. "Accessibility modding" – a collaborative work model for co-creation of inclusion.
    - We can include audiences gradually, by defining interaction alternatives to suit their needs.
    - Every alternative sums to every other (mutualistic contributions).

- Creation comes from abilities and skills – everyone can contribute.
- People can share interaction alternatives.
- People can share interaction profiles.

29. "Inclusive creation" – a collaborative work model for co-creation of tailorable games.

    - If we provide tools, people can co-create games.
    - Transient roles: roles can also change according to what someone is currently doing.

      Supervisor person who orchestrates the creation, determining the next steps to advance the project.

      Creator person who is creating game content.

      Collaborator person who can provide improvements for accessibility, usability, and use experience (assets, art, content).

      Player person who is playing a project.

    - A creation process with multiple phases can provide more opportunities for collaboration and making.

30. In our small scale game creation process for tailorable games, we defined the following phases:

    Conception idealization and planning of the project.

    Conversion generation of an initial prototype.

    - We can explore low and medium fidelity prototyping techniques to generate initial prototypes, then request an initial implementation from a Collaborator.

    Evaluation definition of how the project should continue. Who should act next?

    - Should we create more content? Creators at Creation.
    - Should we improve its accessibility? Collaborators (Enablers) at Enrichment.
    - Should we improve existing content? Collaborators (Enhancers) at Enrichment.
    - Should we generate accessible versions and organize a play session? Players at Distribution and Use.
    - Should we finish the project?

    Creation game making activities using technology and creation/authoring tools.

Enrichment external improvements to the project, requested to Collaborators to enable access for broader audiences, improve usability, and provide a better experience of use.

- We can co-create inclusion as a community.

Distribution if the tool generate Meta-Games, we can define which interaction alternatives will be attached to the resulting Games.

Use once we have a Game, Players should play it. If they cannot play it, we have to further enrich it.

Conclusion every project should end at some point.

31. We can support the Conversion and Creation phases with creation tools for end-user development.
    - If the creation tool implements Meta-Games and Games, we can generate games sharing their same benefits.
    - If the creation tool is a Meta-Game itself, we can enable more people to create Meta-Games/Games iteratively.
        - The commands already exist; we just have to define new interaction alternatives to suit the needs of new Creators.

32. If we define tailorable creation tools, many potential Creators might had never used a computer before.
    - We can initiate the activities exploring activities that they are already used to.
    - Storytelling is a rich strategy for newcomers, as they probably have had previous experiences with it (for instance, from books, soap operas, movies, and theatre plays).
    - We can advance gradually to other activities over time. Based on the Create-Consume Spectrum (BASAWAPATNA et al., 2014), we defined the following “gentle-slope” activities:

      Linear Storytelling Simple stories with a single narrative thread.

      Branching Storytelling Branching (non-linear) storytelling;

      Collaborative Storytelling Multiple people contribute to create a story together;

      Storytelling with Pre-Defined Mechanics Characters, places, and objects become interactive entities in a game world.

      Computational Thinking Patterns Pattern-based authoring, on which complex interactions may result from a combination of mechanics.

Creation of New Mechanics End-User Development (EUD) and programming approaches to enable Creators to move towards defining their own games and technologies.

33. Co-creation using pre-defined mechanics:

    - Once we figure interaction alternatives for a mechanic, we can implement it in an authoring tool.
    - We can abstract interaction alternatives as "slots" that people can fill with content.
    - We can provide slots so that people can contribute to create accessibility.
    - We can provide slots so that people can consider the needs of their peers.
    - Community efforts to fill slots and promote use.

34. Towards end-user programming of new mechanics with co-creation of accessibility:

    - Creators can define mechanics with entities, events, subsystems, and event handlers.
    - To convey the information, they need to think on different ways to present its results.

35. As proof of concept, we have created a tailorable game creation platform: Lepi.

    - Lepi explores the storytelling strategy from the activities – up to Collaborative Storytelling.
    - The commands are related to *what* Creators can do define their story – for instance, insert characters and dialogues, insert and cycle among scenes, and play the project.
      - If the default interface is suitable to the needs of a new audience, we can improve it.
      - Otherwise, we can define new interfaces around these commands to suit interaction needs of audiences with other interaction needs.
      - With the architecture, accessible and traditional can co-exist – to alternate between them, we just have to switch profiles and tailor the digital system.
    - Lepi supports creation by traditional audiences of average users, as well as those from people with hearing disabilities and low literacy.
    - Lepi provides slots for text, audio, and videos (for sign language). Resulting games can have any combination of these resources to convey the story to Players.

- Creators can insert characters, objects, and places into their stories. These media have pre-defined interaction alternatives for the three slots – this way, we can convey what they represent to Players with different abilities.
- With the collaborative work model, when a Creator cannot provide an interaction alternative, a Supervisor can request its inclusion by Collaborators.

36. The architecture, the collaborative work model, and support systems (Lepi, at this time) defined our framework.
    - Three pillars to enable inclusive co-creation of tailorable games.
37. Our research hypothesis assumed the pillars of the framework would enable people with different interaction needs to create and play games – both for themselves, and for people with different interaction needs than their own.
    - Evaluation:
      - The architecture was formally defined;
      - The algorithms of the architecture were implemented; we created two tech demos to demonstrate the tailoring.
      - The architecture, the collaborative work model, and Lepi were evaluated together.
        * Lepi implemented the architecture and generated games exploring it.
        * The framework was evaluated over four months and ten meetings, exploring full cycles of the creation process.
        * A not yet considered audience (people undergoing alcohol and drugs rehabilitation, including people with low literacy who had never used computers or played digital games before) was able to co-create games which satisfied their own interaction needs, as well as those of their peers (who might have had different needs).
    - We argue, thus, that we were able to confirm our research hypothesis. We are currently improving Lepi to suit the needs of new audiences, to achieve broader inclusion.
38. In our activities, we have also observed evidence that game creation can transform people over the development process.
    - Creation and use contributed to social and digital inclusion.
    - People became very proud and happy with their creations.
    - People enjoyed when playing their own games and those of their friends.
    - In serious contexts, game creation can be a journey on which the making is not the most valuable result.

- The journey can lead to self-growth, self-expression, self-transformation, increase the self-esteem, and provide opportunities for sharing experiences, perceptions and knowledge.

APPENDIX B

One (Meta-)Game, Infinite Ways to Play: Run-Time Tailorability for Tailorable Games

B.1 Introduction

Practicing the Universal Design(STORY; MUELLER; MACE, 1998) in digital systems may be challenging, especially when we consider Real-Time Interactive Systems (RIS), such as digital games. Interaction models and devices that work well for a group of users might not be suitable for others, as interaction needs and abilities may vary. Every user is unique.

It is, therefore, necessary to embrace differences. We should design and implement systems that respect and allow satisfying individual needs. Although it is impossible to satisfy every possible interaction need with a single set of interactions – especially in digital games[1] –, we can modify user interactions in software – even during use-time. Tailoring and tailoring-based approaches (KAHLER et al., 2000; NERIS, 2010) describe how to design use-time flexible software solutions[2]. Techniques and frameworks, such as the Unified Design (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011) for digital games, show how to design universally accessible games.

However, for digital systems, design alone may not be enough. The implementation should match the design flexibility to achieve run-time tailorability. Due to meta-design (FISCHER et al., 2004; FISCHER; FOGLI; PICCINNO, 2017) characteristics of tailoring and the Universal Design itself, the implementation should match the flexibil-

[1] <http://includification.com/>
[2] In this chapter, we adopt the term "tailoring" as defined by Garcia (2014): as a metaphor for something that is "custom made" to suit the abilities of a user.

ity of the design; thus, the architecture of the system must be extensible and support run-time modifications.

To achieve the desired flexibility, one approach is to make user interaction itself run-time extensible. For this chapter, enabling accessibility improvements is a benefit of the approach that we will describe. For digital systems, use-time happens at run-time. Therefore, provided that we could modify interaction functionalities when an application is running, we would be able to potentially tailor it to suit interaction needs of a user.

One approach towards run-time extensibility for user interaction is to abstract input and output (IO) functionalities of a system instead of pre-defining physical-level interactions. In this approach, developers define *what* a user can do when interacting instead of *how* she/he will perform an action. Therefore, developers define semantic interactions instead of specific actions. Instead of defining how a user should interact with the system, developers define what the user can do with the system (intents and meanings), and leave the user to decide how (the physical action) she/he will perform the interaction.

Semantic interactions, are, thus, related to intent, meanings, and goals of what one can do with a system (for instance, she/he can jump; afterwards, she/he can map a button press to perform the jump). To implement semantic interactions, an essential requirement is to map physical-level interactions into existing commands during run-time. That is, to redefine during how the user will perform IO by selecting and combining alternatives (keys, buttons, sticks, gestures...) to realize her/his intents (select an option, advance to the next option, return to the previous option, jump...).

From an implementation standpoint, the system, therefore, does not assume the existence of a user. The system is implemented to *no one*. It reacts to a finite set of pre-defined semantic interactions to compute its next state. This suggests that, to achieve universally accessible systems, it is first necessary to remove the user from the application to, at a later step, include the user into an ideal application, gifting her/him with accessible interaction that matches her/his needs. We called these games as tailorable games. At first, this might seem paradoxical, for an interactive system presupposes the existence of a user. There is no interactive system without a user – or is there?

In an interactive user-less system, there is no human. Thus, it is not useful for human beings – at least not practically. Rather, it is useful as theoretical result: if it is possible to create an interactive, abstract, IO-free system from the user perspective (thereafter called meta-system), and it is also possible to define all IO interactions during run-time, then it should be possible to create fully tailorable systems. The meta-system is used as the basis to build the exact same (logical) system. Derived systems differ from it by defining different physical-level interactions, transforming it into an interactive system for humans. Thus, the meta-system allows implementing the system logic *once* and to specialize the IO *multiple times*.

The meta-system approach with composition-able (composable) interaction can lead to the construction of tailorable systems, by considering finite sets of users interactions which should be met. An interactive system with no users is universal per se (GARCIA, 2014) – if there are no users for it, it trivially follows that it includes everyone. The same reasoning is valid to include a given set interaction needs. When we define the required IO to suit the interaction needs of a group of users and introduce them to the system, it becomes universal for no users (the base case) and for this new group of users. Thus, to reach the status of universal, we may introduce new groups of users iteratively, by considering their needs and implementing the required alternatives to enable physical-level interactions. At each iteration, the system will remain accessible to previously included users, as well as enable more people to use it. Provided that we can repeat this process for every interaction needs, the system would (theoretically) become universal. Otherwise, it would reach the most inclusion that is possible to achieve.

At a theoretical level, the meta-system could be a useful implementation artifact. Could we implement one? To answer this question, we could divide problem into two:

1. Is it possible to create an interactive system without user IO, that is, is it possible to implement a meta-system?

2. Is it possible to define all physical-level IO interactions for an application at run-time?

We address these questions in the remainder of chapter, exploring mathematical set theory on carefully selected elements to define meta-systems, create them, and convert them into user-interactive systems. The proposed approach extends the authors previous work in digital games domain (GARCIA; NERIS, 2014; GARCIA, 2014), by providing a general case theory and algorithms. As the original work explored games – and our chosen architectures are more common in game programming – the remainder of this chapter refers to systems as games. Nevertheless, although this chapter uses digital games to represent interactive systems, it could be extended to other digital applications built following the same process. This chapter will, thereafter, use the term Meta-Game for digital game meta-systems[3].

[3] Therefore, we use meta-systems and Meta-Games, and systems and games, interchangeably in this chapter, unless we state otherwise. The same applies to users, players, and humans; we use them as synonyms, unless stated otherwise to avoid ambiguities.

To define and implement meta-systems, we explore and combine three different software architectures in this chapter: Event-Driven Architectures (EDA), Data-Driven Architectures (DDA), and Entity-Component Systems (ECS) (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014; GARCIA, 2014; NYSTROM, 2014). The gaming industry has been exploring them to empower game developers, as they provided run-time flexibility to create complex gameplay. We, in turn, explore them to redefine the entire input and output (IO) of a digital system, decoupling semantic from physical-level interactions to define the meta-system and to tailor it into accessible systems afterwards.

Our resulting approach turns accessibility functionalities into run-time modifications, enabling run-time IO tailoring. With the approach, provided that there is a combination of entities, components, events, and event handlers which we combine to satisfy desired accessibility and interaction requirements, it becomes possible to create an accessible specialization of the Meta-Game for a given public. In this chapter, we will call specializations of the Meta-Game as Games (with capital 'G').

More importantly, provided that the digital system uses events, event handlers, and entities, components, it is possible to attach or to detach data and behaviors at run-time. This means that, for any given interaction required for an interaction need, if we can decompose the interaction into a finite set of events, event handlers, entities, and components, then we can create an accessible version of the application to that need. Therefore, if there are not enough elements to include a user yet, we can define and implement new accessible elements. As we can repeat this process infinitely, for all desired interaction needs, we could create a universal tailorable system – at least, for as far as existing technology and assistive technologies allows us. If we cannot achieve universal access, we would reach maximum inclusion with what is currently possible.

B.2 Related Work

This section presents related work and concepts explored in this chapter. As stated in Section B.1, we focus on digital systems in the game domain. We start with game interaction, following with Universal Design in general and in games, and game accessibility guidelines and strategies. Afterwards, we describe required programming architectures (ECS, DDA, and EDA) that we will employ to define meta-systems/Meta-Games.

B.2.1 Game Interaction

Yuan, Folmer & Harris (2011) described a generic interaction model for games (hereafter referred to as interaction model), relating accessibility and interaction problems that a user may face when playing digital games. According to the model, a user continuously performs three steps when she/he is playing a digital game:

1. receives stimuli,
2. determines response, and
3. provides input.

The user repeats these steps from the beginning of a play session until its end. As they divide the playing activity, they allow identifying possible interaction issues. For instance, at Step (1), a user receives sensory stimuli from the system. The received stimuli are usually visual, aural, or haptic; thus, to perform this step, the user must have the required visual, hearing, and motor abilities to perceive the content. In other words, she/he must not have the corresponding disabilities if there are not other ways to perceive the content.

At Step (2), the user needs to interpret and comprehend the stimuli that she/he was received, which requires cognitive abilities. Finally, at Step (3), the user has to interact with the input devices to translate her/his cognitive response into physical-level interactions – with the current technologies, this usually requires motor/speech abilities to provide input to a device.

B.2.2 Universal Design

Albeit the interaction model allows one to identify interaction problems in a game, developers are responsible for designing accessible experiences. One approach for inclusive design is the Universal Design (also known as Design For All) (STORY; MUELLER; MACE, 1998).

The Universal Design proposes the conception of solutions aiming to enable the largest possible extension of users, considering their physical, sensory, and cognitive abilities (NERIS, 2010; STORY; MUELLER; MACE, 1998). The principles of Universal Design are a good way to describe the practice (STORY; MUELLER; MACE, 1998):

1. equitable use;
2. flexibility in use;
3. simple and intuitive use;
4. perceptible information;
5. tolerance for error;

6. low physical effort; and

7. size and space for approach and use.

According to the principles, a universal solution encompasses the diversity in the design, aiming to be as inclusive as possible.

Tailoring (KAHLER et al., 2000) is one of the approaches to Universal Design, allowing changing a software application according to its use context. This enables us to "design for change" (NERIS, 2010), as it allows meeting contexts of use or scenarios not previously anticipated.

Tailoring is not restricted to aesthetics modifications on a system. It does also enable the inclusion of new functionalities. This is particularly useful for digital systems, as we can define multiple interfaces to suit interaction needs of a user. Instead of "one-size fits all", we can define different interaction alternatives to suit the needs of multiple audiences.

B.2.3 Universally Accessible Games

Different users may have different interaction needs. Due to accessibility barriers in games, there is not a single set of universally accessible interactions. It is, therefore, impossible to enable every one to interact with the very same game, without modifications. Due to this restriction, game accessibility approaches aim to enable persons with a specific disability to interact with a given game (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011) – instead of enabling every need at once, we can focus at improving the interaction for one disability at a time.

Universally Accessible Games (UA-Games) (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011) describe games designed according to the Universal Design principles. The goal of a UA-Game is the necessity is to make games more accessible and playable by a wider range of users. UA-Games can be tailoring-based – they can offer interaction alternatives to suit interaction needs of a user. In special, they can also allow users with different interaction needs to play together.

Grammenos, Savidis & Stephanidis (2011) defined the Unified Design, a participatory, iterative, and user centered framework on which developers, domain experts and users work together to design UA-Games. The Unified Design aims to define an abstract game, on which the activities of a user are defined without specific input or output (IO) interactions.

The abstract game design allows developers to define multiple IO interaction schemes, each of which aiming to address a specific interaction need. Multiple schemes allow a user to choose, according to her/his own interaction needs, the one that better suits her/his needs. This choice defines the specific IO for the game session.

B.2.4 Game Accessibility Guidelines and Strategies

Several studies describe guidelines and strategies to improve game accessibility, including (ARCHAMBAULT et al., 2008; GARCIA; NERIS, 2013b; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; International Game Developers Association, 2004; MELONIO; GENNARI, 2013; OSSMANN; ARCHAMBAULT; MIESENBERGER, 2006; OSSMANN; MIESENBERGER, 2006; SAVIDIS; STEPHANIDIS, 2006; YUAN; FOLMER; HARRIS, 2011). Institutions and foundations do also provide help and advice, such as The Game Accessibility project[4], The Able Gamers' Includification[5], Game Accessibility Guidelines[6], and the UPS Project Guidelines[7].

Based on the analysis of the interaction model and on game accessibility resources, Yuan, Folmer & Harris (2011) identified, generalized, and summarized techniques for improving game accessibility. Their summary groups the approaches in high and low-level strategies. High-level strategies aim enabling users to interact with a game in a specific step of their interaction model. Low-level strategies are possible implementations of a high-level strategy, used to address a specific disability.

For Step (1), Yuan, Folmer & Harris (2011) identified two high-level strategies: enhancing or replacing stimuli. Enhancing stimuli aims to augment visual cues. Possible low-level strategies include using accessible color themes, zoom, and increased font size. Replacing stimuli aims to convert the original stimuli into another; for instance, audio might be converted into textual (subtitles, closed captions) or non-textual conversions (visual cues, sign language). The same reasoning can be applied to replace visual content with audio or haptic stimuli.

For Step (2), they identified three high-level strategies: reducing stimuli, time constraints, or input. Those are aimed at cognitive impairments. Low-level strategies include limiting the number of game objects, simplifying the story line, increasing or decreasing times, and removing or automating game input.

Lastly, for Step (3), the high-level strategies explore reducing or replacing input, to aid on motor impairments. Low-level strategies include scanning, removing input, automating input, and alternative control schemes, potentially using different input devices (such as using voice or brain control).

4 <http://game-accessibility.com/>
5 <http://includification.com/>
6 <http://gameaccessibilityguidelines.com/>
7 <http://web.archive.org/web/20110724181620/http://gameaccess.medialt.no/guide.php>

B.2.5 Entity-Component Systems

Entity-Component Systems (ECS) (GARCIA; NERIS, 2014; MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014) allow the definition and characterization of entities by the composition (or aggregation) of components. An ECS is similar, although more extreme, variation of the Decorator pattern (GAMMA et al., 1994). In an ECS, an entity may be fully described by its components.

Strategies to implement entities and components may vary. For this chapter, one particularly important difference is how the ECS defines components' data and processes them. One option is to define data-only components. In this case, subsystems process certain types of components to define game behavior and mechanics. For this chapter, we assume data-only components, for it eases the required separation of logic and interaction functionalities. A second option is to process data in the component itself – in this case, subsystems are not needed. Regardless of strategy, it is possible to add to and to remove components from an entity at will and at any time, including at run-time. Thus, an ECS allows us to fully modify entities during run-time, should we desire so.

For this chapter purposes, thus, a component is raw data which add attributes and characteristic behaviors to an entity, when attached to it[8]. A component aggregates well-defined data and behaviors to entities, defining what it is able to do in a game world. As such, the approach eases the creation of game objects, increases the flexibility for the design, and promotes software reuse.

Although they were originally defined and used in games, the Academic Literature describes the use of ECSs for other purposes. When we performed a Systematic Review (KITCHENHAM, 2004) using the string ``entity component'' OR ``entity-component'' (at November of 2017) in the Association for Computing Machinery (ACM) Digital Library, Institute of Electrical and Electronics Engineers (IEEE) Xplore, and Scopus databases, we identified 10 studies which described ECSs or systems using ECSs. Alatalo (2011), Dahl et al. (2013), and Li, Wang & Zhou (2006) use ECSs to define interactive virtual worlds. Wiebusch & Latoschik (2015), Lange, Weller & Zachmann (2016), and Fischbach, Wiebusch & Latoschik (2017) describe enhancements for using ECSs in Real-Time Interactive Systems (RIS). Rein et al. (2017) describe how the approach can allow adapting objects and class memberships in Object-Oriented Systems. Fontana et al. (2017) and Orendt & Henrich (2015) describe the use of ECSs in non-gaming domains. The first applies an ECS to build a tool to create physical designs; the second employs an ECS in robotics.

[8] It is, thus, important to note that these components are not exactly similar to Commercial Off-The-Shelf components – they are much more flexible, as they are raw data. For instance, a transform matrix may become a component that, when attached, enables an entity to acquire a position, rotation, and orientation in a game world. A sprite, or model and texture may provide data for one to graphically view an entity; a sound effect may enable one to hear an entity.

The last study is ours. In a previous work, we (Garcia & Neris (2014)) outlined how ECSs can contribute to promote game accessibility. It was an improvement from Garcia & Neris (2013a), and completed in Garcia (2014), with UGE (available at <https://github.com/francogarcia/uge>). UGE fully decoupled game logic from IO, allowing one to fully redefine how to interact and perceive a game with the same rules (logic). In this chapter, in a way, UGE turned into case study serving as technology proof for the implementation of this chapter. However, instead of requiring a full engine to achieve our purposes, we simplified it to its core strategies. In the remained of this chapter, we discuss the architectures we have explored in UGE, and simplify the original approach to its core, resulting in theorems for run-time tailorability in digital games.

B.2.6 Data-Driven Architectures

Data-Driven Architectures (DDAs) (MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014) allow input data to define the content and the execution flow of a digital game. A DDA loads the required data of the game while its running, from external data resources (such as text files or databases). For example, instead of hard-coding variable and constant values, we can define them into an Extensible Markup Language (XML) file and load it when required. This allows the values to be modified without changes to the game source code, avoiding compiling and, potentially, the need for a software engineer to make a change.

DDAs are often present in game engines to increase their extensibility, ease the inclusion or modification of content, and allow the development of new games using the engine. Some additional benefits include the possibility of creating and using external tools to generate assets (such as image processing and modeling tools), developing specific game content creation tools (such as scenario and world editors), and allowing end-user created modifications (mods).

It is possible to combine ECS with a Factory pattern (GAMMA et al., 1994) to ease the creation of the entities (MCSHAFFRY; GRAHAM, 2012). In this case, an external data resource describes the components that an entity shall have. The application parses the resource, and, using a Factory, creates the desired entity by adding the specified components to it. In this approach, the data resource acts as a blueprint to create entities according to a data-driven definition.

B.2.7 Event-Driven Architectures

According to Gregory (2014), are “anything of interest that happens during gameplay”, and “games are inherently event-driven”. An event is a data structure used to convey the occurrence of something important to the game – something on which further functionalities, processing or detailing depends.

Event-Driven Architectures (EDAs) explore events to decouple the game logic from its implementation. By exploring strategies such as the Observer pattern (GAMMA et al., 1994), we can define dependency relationships between a point of interest (the event) and its processing (the event handler, also known as event listener). When the event happens, the event handlers registered for it are called to perform their work. This allows changing the execution flow of the application, decoupling the implementation, and easing the communication of different parts of the application.

Events and event handlers form many-to-many relationships. An event can have as many registered event handlers as we desire; an event handler may also be registered for as many events as needed. Registering and unregistering an event handler for an event are both run-time operations.

B.2.8 Input Mapping (Input Re-Mapping)

Input mapping (or re-mapping) (GREGORY, 2014) enable players to customize how they control a game. The degree of choice vary according how it is implemented. Simpler implementations assume a fixed input device – for instance, a controller or a keyboard –, and allow the player to change the bindings. More sophisticated implementations abstract the input device as well, granting greater flexibility. In this chapter, we follow the approach to abstract devices and bindings[9].

B.3 Implementing Tailorable Games

As previously stated in Section B.1, practicing the Universal Design may be challenging – requirements change according to abilities and capabilities of users. When we consider RIS, such as digital games, there may be further complexities, such as available IO devices and time constraints for user interaction.

For an inclusive game, accessibility functionalities will vary depending on interaction needs to address in order to enable people to play. When we consider a tailorable game, the ultimate goal for the design would be suiting needs at an individual level (that is, providing features to support the abilities of the user who is playing). Therefore, the implementation, ideally, should support individual levels of adaptability as well – flexibility, extensibility, and adaptability are important requirements –, via tailoring.

Moreover, the adaptability should occur at run-time. Thus, the system has to be extensible and modifiable at run-time, being able to change itself according to arbitrary

[9] A reference for implementation is available at <https://www.gamedev.net/blogs/entry/2250186-designing-a-robust-input-handling-system-for-games/>.

sets of specifications. This implies that the building blocks of the implementation must have these characteristics as well, both by themselves and when combined. Furthermore, they should allow the architecture to support run-time modifications of the execution flow with minimal changes to the existing source code.

The ideal situation, therefore, is to fully decouple the system's logic from user interaction. This decoupling would allow coding the system logic once, and to define multiple interaction alternatives. In particular, it would allow for adapting interactions to match abilities of a user as needed, case by case. For accessibility purposes, it is not the game logic that have to change. Rather, we should be able to define IO and physical-level interactions at run-time. The user who will interact with the game is not known before the game is running; as such, we cannot impose specific and immutable IO interactions on the game.

Paradoxically, from the technology side, one possibility would be to create interactive human-agnostic systems – systems without human IO. That is, to achieve a tailorable game, it is first necessary to take away the user from the game to, at a later step, include the user into an accessible game. Provided that it is possible to simulate external input without user interaction (with any programming mechanism), it might be possible to create an IO-free digital game that might be later tailored to include any desired IO interactions.

With an IO-free implementation, we could implement games at an abstract level, without any references to user physical-level interactions. Such non-human interactive game can be seen as a Meta-Game: a template for the creation of infinite similar Games with the same run-time logic[10]. When we add IO interactions to the Meta-Game, it

[10] This goes a step further than tiered implementation patterns, such as the Model-View-Controller (MVC) (GAMMA et al., 1994): both the input and the output can be changed at any time, without assuming any kind of specific interaction. MVC proposes making the model (logic, in this chapter) independent of views (which handles output) and controllers (responsible for handling input).

For implementation, views and controllers may be merged together – and frequently are people without any physical, cognitive, and emotional disabilities are assumed as average users. For this chapter, this is a limitation. For universality purposes, it is equally important not to assume any particular views nor controllers – in fact, not even assume the use of any single views and controllers at a time. Instead, the use of multiple controllers *at once* may be useful for motor impairments. Thus, a more flexible approach is to explore composition/aggregation to define IO interactions. In MVC terms, the aim is to define "single-responsibility" views and controllers. Not a single view to handle all output, or a single controller to handle all input. Instead, we can define individual views and controllers for each/important (sub-)models. In this case, a complete view would be the aggregation of smaller sub-views; the same reasoning applies for a complete controller. The "whole system" would be, thus, the combination of model, views, and controllers, which, combined, may generate multiple systems with different combinations of views and controllers. This enables the use of multiple IO devices and schemes, providing users with choices to suit their needs and interests. In this chapter, we name the "whole system" as a resulting "Game"; the "model" with software constructs to simulate interaction as "Meta-Game"

Although a sophisticated MVC with Command, Composite, Decorator, Observer, and Strategy patterns could be one option for the implementation considering Object-Oriented Systems, it is not the only one. In fact, Object-Oriented Systems are not even required for the implementation of a system transforms itself into a full-fledged Game, with which a user can interact.

We may explore the concepts of Meta-Game and Game for universal access purposes via tailorable games. Provided that it is possible to construct an entire digital game without physical-level references for user interaction, it becomes possible to tailor the IO during run-time to suit interaction needs of a user. In each specialization, we tailor the available abstracts elements by defining physical-level interactions aimed to enable a user to play. Different specializations may address different interaction problems. When a user wishes to play, she/he combines the most adequate specializations to her/his needs, creating a personal version of the Game for herself/himself.

Thus, a possible approach to implementing universal tailorable games begins with an IO-free implementation: the Meta-Game, which defines *what* users can do. Then, at use-time, it aggregates/composes functionalities to define human interactions – *how* the user provides input to her/his Game, and how the Game provides feedback for him/her.

How could we implement a Meta-Game? For this chapter, based on our experience with UGE, we chose entities, components, events, and event handlers as primitive elements to implement a tailorable game, as discussed following in Section B.5. Entities and components enable us to attach components to or to detach them from the entities during run-time; this grants the digital game with re-definitions of the data members from any entities during run-time. Events and event handlers allow decoupling the implementation, by separating an important happening from its processing.

Components and event handlers can make a game independent of user output. Events and events handlers can make the game independent of input – in this proposal, events serve as means of providing semantic interactions (*what* it is possible to do), whilst handlers provide the implementation (*how* to do it). Events are, therefore, the mechanism for game communication. In this way, it is possible to decouple the user interaction from the game implementation: the Meta-Game does not react to user input, but to events (thus, it does not care about the origin of the event, only about its occurrence). The combination of both parts enables us to implement an IO-free Meta-Game.

An interactive system without users is not practically useful, though. Therefore, it is necessary to introduce the user back to the Game: this is the role of the DDA. It allows us to define an external data source – defined as an Interaction Profile, discussed

sharing the same benefits. The algorithms described later in this chapter, for instance, may be implemented in Procedural, Imperative, Functional, Object-Oriented, or Aspect-Oriented Programming alike, as its requirements are software constructs and basic IO functionalities, such as file IO. Provided that developers follow the proposed logic and IO separation, IO-tailoring is guaranteed by design (due to the systematic nature of set constructs).Moreover, provided developers follow the principles of the set operations, they may be able to use other data structures (such as trees and hash tables) as sets. If they avoid repetition, they can define data hierarchies for their elements and still benefit from the approach.

in Section B.9 –, which represents and includes the necessary changes into the Game. It also enables the persistence of the changes for future uses, allowing the users to use or tweak the interaction in later sessions – or share interesting or useful configurations with a community.

B.4 Formalization of a Tailorable Game

In the remainder of this chapter, we aim to define a structured and systematic way to combine EDA, DDA, and ECS to provide an architecture to implement Meta-Games and Games.

To achieve this, our first goal is to provide a formalization to define concepts and operations for our architecture. We have chosen set theory to represent the formalization, for two reasons. First, it allows demonstrating the validity of the architecture a compact and non-ambiguous way. Second, because the final result allows for an interesting interpretation: if we can convert any functionality into a combination of the concepts of the architecture, then we will be able to insert this functionality into a digital game at run-time.

In this section, we present the formalization for the architecture. We will introduce and discuss how to develop software with the formalization from Section B.5 onward. The results from the formalization will serve as the basis for the construction of the Meta-Game (abstract system: interactive system without human-interaction) and its conversion into a Game (concrete system: interactive system with human-interaction). The goal is to create abstract Meta-Games without human-interactions, to, at use-time, combine existing IO functionalities to transform a Meta-Game into one (or more) interactive Games. As it will be discussed in Section B.7, if we can implement the Meta-Game, then we can create, by adding IO functionalities, the desired physical-level interactions with components, events, and event handlers to tailor the Meta-Game into a Game.

B.4.1 Components (C)

Definition 1 (Component)**.** A component is a data record that has a single, well-defined purpose. ◀

Definition 2 (Logic Component)**.** Logic components are components that store data to abstract the behaviors of a system. A logic component cannot be an input component, neither can it be an output component. ◀

Definition 3 (Input Component)**.** Input components are components that store input data. An input component cannot be a logic component, neither can it be an output component. ◀

Definition 4 (Output Component)**.** Output components are components that store output data. An output component cannot be a logic component, nor can it be an input component. ◀

Lemma 1 (Sets of Components)**.** *It is possible to join the disjoint sets of logic components, input components, and output components into a single set of unique components, which can be split back into the three original, disjoint component sets.* ★

Proof. Let C be a non-empty finite set of components. Let C^L a set of logic components, C^I be a set of input components, and C^O a set of output components. C is the union of the pairwise disjoint subsets C^L, C^I, and C^O.

By definition, C^L (Definition 2), C^I (Definition 3), and C^O (Definition 4) are all components that do not overlap.

$$\begin{aligned} C^L \cap C^I &= \emptyset \\ C^L \cap C^O &= \emptyset \\ C^I \cap C^O &= \emptyset \end{aligned} \tag{B.1}$$

Therefore, C^L, C^I, and C^O are pairwise disjoint. Furthermore, as C^L, C^I, and C^O are components, they belong to C.

$$\begin{aligned} C^L &\subseteq C \\ C^I &\subset C \\ C^O &\subset C \end{aligned} \tag{B.2}$$

As these are all possible categories for components, the union of C^L, C^I, and C^O results in the universe set C. □

$$C^L = \{c_1^L, c_2^L, \ldots, c_{ncl}^L\}$$ Logic Components

$$C^{I-H} = \{c_1^{I-H}, c_2^{I-H}, \ldots, c_{ncih}^{I-H}\}$$ Human Input Components

$$C^{I-AI} = \{c_1^{I-AI}, c_2^{I-AI}, \ldots, c_{nciai}^{I-AI}\}$$ AI Input Components

$$C^I = C^{I-H} \cup C^{I-AI}$$ Input Components

$$C^O = \{c_1^O, c_2^O, \ldots, c_{nco}^O\}$$ Output Components

$$C^{IO} = C^I \cup C^O \qquad \text{Input and Output Components}$$

$$C = C^L \cup C^{IO} = C^L \cup C^I \cup C^O$$
$$C = \{c^L_1, c^L_2, \ldots, c^L_{ncl}, c^{I-H}_1, c^{I-H}_2, \ldots, c^{I-H}_{ncih}, c^{I-AI}_1, c^{I-AI}_2, \ldots, c^{I-AI}_{nciai}, c^O_1, c^O_2, \ldots, c^O_{nco}\}$$
$$nc = ncl + ncih + nciai + nco$$

Components

B.4.2 Entities (E)

Definition 5 (Entity)**.** An entity is a non-empty finite set of components. ◀

$$E = \{e_1, e_2, \ldots, e_{ne} \quad | \quad e_i \subseteq C, \quad e_i \neq \emptyset, \quad 1 \leq i \leq ne\} \qquad \text{Entities}$$

Examples:

$$\begin{aligned} E &= \{e_1, e_2, e_3\} \\ e_1 &= \{c^L_1, c^O_1\} \\ e_2 &= \{c^L_1, c^L_2, c^{I-H}_1, c^O_1\} \\ e_3 &= \{c^L_3, c^L_4, c^{I-AI}_1\} \end{aligned} \tag{B.3}$$

B.4.2.1 Modifying Entities

B.4.2.1.1 Addings Components to an Entity

Lemma 2 (Adding a Component to an Entity)**.** *Adding a component to an entity results into a new entity, which introduces the behaviors from the new component to all existing ones from the original entity.* ★

Proof. Let e be an entity belonging to the set of entities E, and c be a component belonging to the set of components C. As entities are sets of components, the entity e' results from the union of e and $\{c\}$.

$$e' = e \cup \{c\} \tag{B.4}$$

This result holds even if c does already belong to e: in this case, the resulting entity e' is e itself. □

$$\begin{aligned} e_i^{'} &= e_i \cup \{c\}, \quad e_x \in E, c \in C, \\ E^{'} &= (E \setminus e_i) \cup e_i^{'} \end{aligned} \qquad \text{Adding a Component to an Entity}$$

Examples (entities from Equation B.3):

$$\begin{aligned} E &= \{e_1, e_2, e_3\} \\ e_1 &= \{c_1^L, c_1^O\} \\ e_1^{'} &= e_1 \cup \{c_2^O\} = \{c_1^L, c_1^O, c_2^O\} \\ E^{'} &= \{e_1^{'}, e_2, e_3\} \end{aligned} \tag{B.5}$$

B.4.2.1.2 Removing Components from an Entity

Lemma 3 (Removing a Component from an Entity)**.** *Removing a component from an entity results into a new entity, which retains all existing behaviors except to the ones aggregated by the former component.* ★

Proof. Let e be an entity belonging to the set of entities E, and c be component belonging to the set of components C. As entities are sets of components, e' is the entity resulting from the difference of e and $\{c\}$.

$$e' = e \backslash \{c\} \tag{B.6}$$

This result holds even if c does not belong to e: in this case, the resulting entity e' is e itself. □

$$\begin{aligned} e_i^{'} &= e_i \setminus \{c\}, \\ E^{'} &= (E \setminus e_i) \cup e_i^{'} \end{aligned} \qquad \text{Removing a Component from an Entity}$$

Example: (entities from Equation B.3):

$$\begin{aligned} E &= \{e_1, e_2, e_3\} \\ e_1 &= \{c_1^L, c_1^O\} \\ e_1^{'} &= e_1 \setminus c_1^O = \{c_1^L\} \\ E^{'} &= \{e_1^{'}, e_2, e_3\} \end{aligned} \tag{B.7}$$

B.4.2.2 Abstract Entities (E^L)

Definition 6 (Abstract Entity)**.** An abstract entity is an entity defined only by logic components. ◀

$$E^L = \{e_1 \cap C^L, e_2 \cap C^L, \dots, e_{ne} \cap C^L\} \qquad \text{Abstract Entities}$$

Examples (entities from Equation B.3):

$$\begin{aligned} e_1^L &= \{c_1^L\} \\ e_2^L &= \{c_1^L, c_2^L\} \\ e_3^L &= \{c_3^L, c_4^L\} \end{aligned} \tag{B.8}$$

B.4.2.3 Concrete Entities (E^{IO-H})

Definition 7 (Concrete Entity)**.** A concrete entity is an entity defined by any components, provided at least one is a logic component. ◀

$$E^{IO} = \{e_1^{IO}, e_2^{IO}, \dots, e_{ne}^{IO} \quad | \quad e^{IO} = e_i \cup X, \quad e_i \in E, \quad \forall X \in \mathcal{P}(C^I \cup C^O), \quad 1 \le i \le ne\}$$
Concrete Entities

Examples (entities from Equation B.3):

$$\begin{aligned} e_1 &= \{c_1^L, c_1^O\} \\ e_2 &= \{c_1^L, c_2^L, c_1^{I-H}, c_1^O, c_3^O\} \\ e_3 &= \{c_3^L, c_4^L, c_1^{I-AI}, c_2^O\} \end{aligned} \tag{B.9}$$

B.4.2.4 Converting Entities

Theorem 1 (Entity Reduction)**.** *Any entity can be converted to an equivalent (and unique) abstract entity, which retains the original logic behavior.* ★

Proof. According to Definition 5, an entity is a finite set of unique components. According to Definition 6, an abstract entity is an entity without any input or output components. Thus, intuitively, to convert an entity into a corresponding abstract entity, it is necessary to remove all original input and output components of an entity (Lemma 3).

From the definition of entity, an arbitrary entity e is a set of finite components belonging to the digital game set of components C. From Lemma 1, it is possible to decompose C into the sets C^L, C^I, and C^O, corresponding to the sets of logic, input and output components, respectively.

The corresponding abstract entity a is obtained from the intersection of e and C^L.

$$a = e \cap C^L \tag{B.10}$$

As the intersection results in a set of unique logic components, all references to input and output components are removed in the process. Thus, the resulting entity is abstract.

It is interesting to note that this is valid for any IO composition for a given entity: the intersection will result in the same set of logic components. Thus, the abstract entity is unique for this set of entities. □

Theorem 2 (Entity Construction)**.** *Any entity can be converted into a logically equivalent concrete entity, although with, potentially, different input and output components.* ★

Proof. According to Definition 5, an entity is a finite set of unique components. According to Definition 7, a concrete entity is an entity that may have input or output components. Thus, intuitively, to convert an entity into its corresponding concrete entity, it is necessary to add at least one input or output component to the original entity (Lemma 2).

From the definition of entity, an arbitrary entity e is a set of finite components belonging to the digital game set of components C. From Lemma 1, it is possible to decompose C into the sets C^L, C^I, C^O, corresponding to the sets of logic, input and output components, respectively. From the theorem enunciate, there is at least one input or output component; thus, the intersection of C^I and C^O is not empty.

As C^I and C^O are not empty, it is possible to create as many corresponding concrete entities as the number of possible subsets from the power set of the union of C^I and C^O. In fact, assuming the set C^I has cardinality n and the set C^O has cardinality m, it is possible to create 2^{nm} corresponding concrete entities from the original entity E[11].

Consider e, C, C^L, C^I, and C^O as defined above, $C^{IO} = C^I \cup C^O$, $\mathcal{P}(C^{IO})$ the power set of C^{IO}, c^{IO} a non-empty subset of $\mathcal{P}(C^{IO})$, then a possible logically equivalent concrete entity f generated from e is:

$$f = e \cup c^{IO}, \quad \forall c^{IO} \in \mathcal{P}(C^{IO}) \tag{B.11}$$

□

Theorem 3 (Entity Transformation)**.** *Provided that the Entity Construction conditions are valid, a concrete entity may be converted into an equivalent abstract entity, and vice-versa.* ★

Proof. Let e be an arbitrary entity. From the Entity Reduction, it is possible to reduce e to a corresponding abstract entity a. Afterwards, from the Entity Construction, it is possible to convert a back into e – e is one of the possible concrete entities resulting

[11] One of the possible results might be abstract entity provided that the initial entity is also abstract, one of the possible subsets is the empty subset.

from the union of a with a subset of the power set of the union of input and output components.

The reciprocal does hold as well. Starting with e, it is possible to obtain the same entity e from the Entity Construction – this time, the very same IO components of e. Afterwards, from the Entity Reduction, it is possible to reduce e to a corresponding abstract entity a. □

B.4.2.5 Agents

Definition 8 (Agent)**.** An agent is an entity with an input component, which is able to interact with the system. ◀

B.4.2.5.1 Human Agents (e^H)

Definition 9 (Human-Agent)**.** A human-agent is a human-controlled agent, that is, it has a human-controlled input component. ◀

$$e_i^H = (e_i \cap C^L) \cup X, \quad e_i \in E, \quad \forall X \in \mathcal{P}(C^I \cup C^O) \mid X \cap C^{I-H} \neq \emptyset \qquad \text{Human-Agent}$$

Examples (entities from Equation B.3):

$$\begin{aligned} e_1^L &= \{c_1^L, c_1^{I-H}, c_3^O\} \\ e_2^L &= \{c_1^L, c_2^L, c_1^{I-H}, c_3^{I-H}\} \\ e_3^L &= \{c_3^L, c_4^L, c_2^{I-H}, c_1^{I-AI}, c_1^O\} \end{aligned} \tag{B.12}$$

B.4.2.5.2 AI Agents (e^{AI})

Definition 10 (AI-Agent)**.** An AI-agent is an artificial-intelligence-controlled agent, that is, it has an AI-controlled input component. ◀

$$e_i^H = (e_i \cap C^L) \cup X, \quad e_i \in E, \quad \forall X \in \mathcal{P}(C^{I-AI} \cup C^O) \mid X \cap C^{I-AI} \neq \emptyset \qquad \text{AI-Agent}$$

Examples (entities from Equation B.3):

$$\begin{aligned} e_1^L &= \{c_1^L, c_1^{I-AI}, c_3^O\} \\ e_2^L &= \{c_1^L, c_2^L, c_2^{I-AI}, c_1^O\} \\ e_3^L &= \{c_3^L, c_4^L, c_1^{I-AI}\} \end{aligned} \tag{B.13}$$

B.4.2.5.3 Transforming Agents

Theorem 4 (Humanization)**.** *An AI-agent may be converted into a corresponding human-agent.* ★

Proof. Let $C^I = \{c_1^I, c_2^I, \ldots, c_i^I\}$ be the set of input components, $C^{I-H} = \{c_1^{I-H}, c_2^{I-H}, \ldots, c_j^{I-H}\}$ $(1 \leq j \leq i)$ the set of human-controlled components, and $C^{I-A} = \{c_1^{I-A}, c_2^{I-A}, \ldots, c_{i-j}^{I-A}\}$ the set of AI-controlled components, with C^{I-H} and C^{I-A} subsets of C^I, and with $C^{I-A} \cup C^{I-H} = C^I$ and $C^{I-H} \cap C^{I-A} = \emptyset$. Let a be an AI-agent.

Due to Definition 8, a is an entity that has an input component; Definition 10 ensures that the input component is an AI-controlled input component. As a is an entity, it has a set of components $\{c_1, c_2, \ldots, c_{n-y}, x_1, x_2, \ldots, x_y\}$, with x_n $(1 \leq n \leq y)$ one the AI-controlled input component, belonging to the set of the existing AI-controlled input components C^{I-A}.

To remove the AI-controlled components from a, we perform a set difference with C^{I-A}, resulting in the non-agent entity e. To promote e to a human-controlled entity h, we can perform the union with s, one of the possible non-empty subsets of the power-set of $\mathcal{P}(C^{I-H})$. Thus, h is:

$$h = e \cup s \tag{B.14}$$

□

Theorem 5 (Automation)**.** *A human-agent may be converted into a corresponding AI-agent.* ★

Proof. The proof is similar to Theorem 4.

Let $C^I = \{c_1, c_2, \ldots, c_i\}$ be the set of input components. Let $C^{I-H} = \{c_1^{I-H}, c_2^{I-H}, \ldots, c_j^{I-H}\}$ $(1 \leq j \leq i)$ be the set of human-controlled components, and $C^{I-AI} = \{c_1^{I-AI}, c_2^{I-AI}, \ldots, c_{i-j}^{I-AI}\}$ be the set of AI-controlled components, with C^{I-H} and C^{I-AI} subsets of C^I, and with $C^{I-H} \cup C^{I-AI} = C^I$ and $C^{I-H} \cap C^{I-AI} = \emptyset$. Let h be a human-agent.

Due to Definition 8, h is an entity that has an input component; Definition 9 ensures that the input component is a human-controlled input component.

As an entity, h has a set of components $\{c_1, c_2, \ldots, c_{n-x}, x_1, x_2, \ldots, x_x\}$, with x_n $(1 \leq n \leq x)$ a human-controlled input component, belonging to the set of the existing input components C^{I-H}.

To remove the human-controlled components from h, we perform a set difference with C^{I-H}, resulting in the non-agent entity e. To promote e to an AI-controlled entity a, we can perform the union with s, one of the possible non-empty subsets of the power-set of C^{I-AI}. Thus, a is:

$$a = e \cup s \tag{B.15}$$

$\square$

Theorem 6 (Agent Transformation)**.** *A human-agent may be converted into an AI-agent, and vice-versa.* ★

Proof. Let a be an arbitrary AI-agent. From the Theorem 4, it is possible to reduce a to a corresponding human-agent h. Afterwards, from the Theorem 5, it is possible to convert h back into a – a is one of the possible AI-agents resulting from the union of H with a subset of the power set of the union of AI-controlled input components.

The reciprocal does hold as well. Starting from h, it is possible to convert it into a with the Theorem 5. Then, with the Theorem 4, it is possible to convert a back into h – h is one of the possible human-agents resulting from the union of a with a subset of the power set of the union of human-controlled input components. $\square$

B.4.3 Events (V) and Commands (A)

Definition 11 (Event)**.** An event is anything (that designers consider) important that just has happened. ◀

Definition 12 (Command)**.** A command is an event issued with the purpose of enabling interaction between an agent (input provider) and the game (input reactor). ◀

$$A = \{a_1, a_2, \ldots, a_{nva}\} \qquad \text{Commands}$$

$$\begin{gathered} V = \{v_1, v_2, \ldots, v_{nvv}\} \cup A \\ V = \{v_1, v_2, \ldots, v_{nvv}, a_1, a_2, \ldots, a_{nva}\} \\ nv = nvv + nva \end{gathered} \qquad \text{Events}$$

B.4.4 Event Handlers (H)

Definition 13 (Event Handler)**.** An event handler is procedure with a single and well-defined purpose that processes an event. ◀

Definition 14 (Logic Event Handler)**.** Logic event handlers are event handlers that process an event (that it has been registered to) to implement the behaviors of the system. A logic event handler cannot be an input event handler, neither can it be an output event handler. ◀

Definition 15 (Input Event Handler)**.** Input event handlers are event handlers process an event (that it has been registered to) to provide input. An input event handler cannot be a logic event handler, neither can it be an output event handler. ◀

Definition 16 (Output Event Handler)**.** Output event handlers are event handlers process an event (that it has been registered to) to provide output. An output event handler cannot be a logic event handler, neither can it be an output event handler. ◀

Lemma 4 (Sets of Event Handlers)**.** *It is possible to join the disjoint sets of input event handlers, output event handlers, and logic event handlers into a single set of unique event handlers, which can be split back into the three original, disjoint component sets.* ★

Proof. Let H be a finite and non-empty set of event handlers. Let H^L be a set of logic event handlers, H^I a set of input event handlers, and H^O a set of output handlers. H results from the union of the pairwise disjoint subsets H^L, H^I and H^O, representing input, output and logic, respectively.

By definition, H^L (Definition 14), H^I (Definition 15), and H^O (Definition 16) are all event handlers that do not overlap.

$$\begin{aligned} H^L \cap H^I &= \emptyset \\ H^L \cap H^O &= \emptyset \\ H^I \cap H^O &= \emptyset \end{aligned} \tag{B.16}$$

Thus, H^L, H^I, and H^O are pairwise disjoint. As H^L, H^I, and H^O are event handlers, they belong to H.

$$\begin{aligned} H^L &\subseteq H \\ H^I &\subset H \\ H^O &\subset H \end{aligned} \tag{B.17}$$

As these are all possible categories for event handlers, the union of H^C, H^I, and H^O results in the universe set H. □

$$H^L = \{h^L_1, h^L_2, \ldots, h^L_{nhl}\} \qquad \text{Logic Event Handlers}$$

$$H^{I-H} = \{h^{I-H}_1, h^{I-H}_2, \ldots, h^{I-H}_{nhih}\} \qquad \text{Human Input Event Handlers}$$

$$H^{I-AI} = \{h_1^{I-AI}, h_2^{I-AI}, \dots, h_{nhiai}^{I-AI}\}$$ AI Input Event Handlers

$$H^{I} = H^{I-H} \cup H^{I-AI}$$ Input Event Handlers

$$H^{O} = \{h_1^{O}, h_2^{O}, \dots, h_{nho}^{O}\}$$ Output Event Handlers

$$H^{IO} = H^{I} \cup H^{O}$$ Input and Output Event Handlers

$$H = S^L \cup S^{IO} = S^L \cup S^I \cup S^O$$
$$H = \{h_1^L, h_2^L, \dots, h_{nhl}^L, h_1^{I-H}, h_2^{I-H}, \dots, h_{nhih}^{I-H},, h_1^{I-AI}, h_2^{I-AI}, \dots, h_{nhiai}^{I-AI}, h_1^O, h_2^O, \dots, h_{nho}^O\}$$
$$nh = nhl + nhih + nhiai + nho$$
Event Handlers

B.4.4.1 Modifying Event Handlers

B.4.4.1.1 Register Event Handlers

Lemma 5 (Registering an Event Handler to an Event)**.** *An event for which one registers a new event handler keeps all former processing functionalities, as well as incorporating new ones provided by the new handler.* ★

Proof. Let v be an event belong to the set of events V, and h be a set of events processed by an event handler to the set of event handlers H. As event handlers handle sets of events, h' is the resulting set from the union of H_1 and $\{v\}$.

$$h' = h \cup \{v\} \tag{B.18}$$

This result holds even if v does already belong to h: in this case, the resulting set of events handled by h' is h itself. □

$$h_i^{'} = h_i \cup \{v\}, \quad h_x \in H, v \in V,$$
$$H^{'} = (H \setminus h_i) \cup h_i^{'}$$
Registering an Event Handler to an Event

B.4.4.1.2 Unregistering Event Handlers

Lemma 6 (Unregistering an Event Handler from an Event)**.** *An event from which one unregisters a new event handler keeps all former processing functionalities, except the ones provided by the former handler.* ★

Proof. Let v be an event belonging to the set of events V, and h be a set of events processed by an event handlers belonging to the set of event handlers H. As event handlers process sets of events, h' is the resulting set from the difference of h and $\{v\}$.

$$h' = h \backslash \{v\} \tag{B.19}$$

This result holds even if v does not belong to h: in this case, the resulting set of events handled by h' is h itself. □

$$\begin{gathered} h_i^{'} = h_i \setminus \{v\}, \quad h_x \in H, v \in V, \\ H^{'} = (H \setminus h_i) \cup h_i^{'} \end{gathered} \qquad \text{Unregistering an Event Handler to an Event}$$

B.4.4.2 Abstract Event Handlers (H^L)

Definition 17 (Abstract Event Handler)**.** An abstract event handler is an event handler that handles an event without having input or output side-effects; only logic processing is allowed. ◀

$$H^L \qquad \text{Abstract Event Handlers}$$

B.4.4.3 Concrete Event Handlers (H^{IO-H})

Definition 18 (Concrete Event Handler)**.** A concrete event handler is an event handler that may handle an event using logic, input, or output side-effects. ◀

$$H^{IO} = \{h_1^{IO}, h_2^{IO}, \dots, h_{nh}^{IO} \quad | \quad h^{IO} = h_i \cup X, \quad h_i \in H, \quad \forall X \in \mathcal{P}(H^I \cup H^O), \quad 1 \le i \le nh\}$$

Concrete Event Handlers

B.4.4.4 Converting Event Handlers

Theorem 7 (Removing IO Event Handlers)**.** *For any event, it is possible to obtain a subset of abstract event handlers from all event handlers.* ★

Proof. According to Definition 13, Definition 14, Definition 15 and Definition 16, an event handler might serve three different purposes: handle logic, input or output. According to Definition 17, an abstract event handler does not perform IO operations. Thus, intuitively, to achieve the theorem result, it is necessary to remove all original set all event handlers related to IO operations (Lemma 6).

Assuming H is the set of all event handlers for a game, from Lemma 4, it is possible to decompose H into the sets H^L, H^I, and H^O, corresponding to the sets of logic, input, and output event handlers, respectively.

Assuming h_V is the set of components for an event v, the corresponding set of abstract event handlers a for v is obtained from the intersection of h_V with H^L.

$$a = h_V \cap H^L \tag{B.20}$$

As the intersection results in a set of unique logic event handlers, all references to input and output components are removed in the process. Thus, the resulting set contains only abstract event handlers. □

Theorem 8 (Adding IO Event Handlers)**.** *For any event, it is possible to obtain a logically equivalent superset of event handlers, albeit with different input or output handlers.* ★

Proof. According to Definition 13, Definition 14, Definition 15 and Definition 16, an event handler might serve three different purposes – handle logic, input or output. According to Definition 17, an abstract event handler does not perform IO operations. Thus, intuitively, to achieve the theorem result, it is necessary to add to the original set at least one event handler that performs IO operations (Lemma 5).

Assuming H is the set of all event handlers for a game, from Lemma 4, it is possible to decompose H into the sets H^L, H^I, and H^O, corresponding to the sets of input, output, and logic event handlers, respectively.

As H^I and H^O are not empty, it is possible to add as many corresponding concrete event handlers as the number of possible subsets from the power set of the union of H^I and H^O. In fact, assuming the set has cardinality n and the set O has cardinality m, it is possible to add up to 2^{mn} event handlers to the original set E[12].

[12] As with the Entity Construction, one of the possible results is the empty set.

Considering H, H^L, H^I, and H^O as defined above, $H^{IO} = H^I \cup H^O$, $\mathcal{P}(H^{IO})$ the power set of H^{IO}, h^{IO} a non-empty subset of (H^{IO}), v the considered event, h_V the initial set of event handlers for v and f_V is the final set of event handlers for v, then a logically equivalent set of event handlers f_v generated from h_v is:

$$f_v = h_v \cup h^{IO}, \quad \forall h^{IO} \in \mathcal{P}(H^{IO}) \tag{B.21}$$

□

Theorem 9 (Event Handler Transformation)**.** *Provided that the Adding IO Event Handlers conditions are valid, an event may have event handlers added to, or removed from, it to obtain subsets of abstract and supersets of concrete event handlers.* ★

Proof. The proof is similar to Entity Transformation.

Let v be an arbitrary event and h_v its set of event handlers. From the Removing IO Event Handlers, it is possible to remove all IO event handlers from h_v to obtain its corresponding set of abstract event handlers a. Afterwards, from the Adding IO Event Handlers, it is possible to convert a back into h_v – h_v is one of the possible sets resulting from the union of a with a subset of the power set of the union of input and output event handlers.

The reciprocal does hold as well. Starting with h_v, it is possible to obtain the same set h_v from the Adding IO Event Handlers – this time, the considered subset will be the one with the very same IO even handlers from h_v. Afterwards, from the Removing IO Event Handlers, it is possible to reduce h_v to a corresponding abstract set of event handlers a. □

B.4.5 Rules (R)

Definition 19 (Rule)**.** A rule is a relation depending on entities that can trigger an event. ◀

$$\begin{gathered} R = \{r_1, r_2, \ldots, r_{nr}\} \\ e_i \, r_j \, e_k \mapsto v_l \quad , \quad 1 \leq i \leq ne, 1 \leq j \leq ne, 1 \leq j \leq nr, 1 \leq l \leq nv \end{gathered} \qquad \text{Rules}$$

Designers can define other arities as needed.

B.4.6 Subsystems (S)

Definition 20 (Subsystem)**.** A subsystem processes a finite and arbitrary subset of all existing components according to the defined rules. ◀

Definition 21 (Logic Subsystem)**.** Logic subsystems are subsystems that handle logic components according to a finite set of pre-defined rules. They can read and write logic components. ◀

Definition 22 (Input Subsystem)**.** Input subsystems are subsystems that handle logic components and input components according to a finite set of pre-defined rules. They can read and write input components, and they may read (and must not write) logic components. ◀

Definition 23 (Output Subsystem)**.** Output subsystems are subsystems that handles logic components and output components according to a finite set of pre-defined rules. They can read and write output components, and they may only read (and must not write) logic components. ◀

$$S^L = \{s_1^L, s_2^L, \ldots, s_{nsl}^L\} \qquad \text{Logic Subsystems}$$

$$S^{I-H} = \{s_1^{I-H}, s_2^{I-H}, \ldots, s_{nsih}^{I-H}\} \qquad \text{Human Input Subsystems}$$

$$S^{I-AI} = \{s_1^{I-AI}, s_2^{I-AI}, \ldots, s_{nsiai}^{I-AI}\} \qquad \text{AI Input Subsystems}$$

$$S^I = S^{I-AI} \cup S^{I-H} \qquad \text{Input Subsystems}$$

$$S^O = \{s_1^O, s_2^O, \ldots, s_{nso}^O\} \qquad \text{Output Subsystems}$$

$$S^{IO} = S^I \cup S^O \qquad \text{Input and Output Subsystems}$$

$$S = S^L \cup S^{IO} = S^L \cup S^I \cup S^O$$
$$S = \{s_1^L, s_2^L, \ldots, s_{nsl}^L, s_1^{I-H}, s_2^{I-H}, \ldots, s_{nsih}^{I-H}, s_1^{I-A}, s_2^{I-A}, \ldots, s_{nsiai}^{I-A}, s_1^O, s_2^O, \ldots, s_{nso}^O\}$$
$$ns = nsl + nsih + nsiai + nso$$

Subsystems

B.4.6.1 Entities of a Subsystems (E^{s_i})

Lemma 7 (Entities of a Subsystem)**.** *A subsystem registers all entities that have all components that it processes.* ★

Proof. Let:

1. G be a digital game;
2. $C = \{c_1, c_2, \ldots, c_n\}$ be a non-empty finite set of components for G;
3. $E = \{e_1, e_2, \ldots, e_o\}$ be a non-empty finite set of entities, each defined by a non-empty finite set of components from C;
4. S_i be a subsystem of G, with $C^{S_i} \subseteq C$ the subset of components required by S_i.

The entities required by S_i are those whose corresponding set of components has each of its components in C^{S_i}. □

Lemma 8 (Entities of a Logic Subsystem)**.** *Logic subsystems can process only logic components.* ★

Proof. The result follows from Definition 21: as it only handles logic components, it is a particular case for Lemma 7. In this case, the set of logic components (C^L). □

$$\begin{gathered} s_i \in S, \\ C^{s_i} \subseteq C, \\ C^{s_i} = \{c_1^{s_i}, c_2^{s_i}, \ldots, c_{ncsi}^{s_i} \quad | \quad ncsi \leq nc\}, \\ E^{s_i} = \{e_1, e_2, \ldots, e_j, \ldots, e_{nesi} \quad | \quad e_j \subseteq C^{s_i}, \quad \forall e_j \in E, \quad nesi \leq ne\} \end{gathered}$$

Entities of a Subsystem

Examples (entities from Equation B.3):

$$\begin{gathered} S = \{s_1, s_2\} \\ C^{s_1} = \{c_1^L\} \\ E^{s_1} = \{e_1, e_2\} \\ C^{s_2} = \{c_2^L, c_1^{I-H}\} \\ E^{s_2} = \{e_2\} \end{gathered} \tag{B.22}$$

B.4.7 Meta-Games

Definition 24 (Meta-Game)**.** Let C_{Meta} be a set of components, V_{Meta} be a set of events, R_{Meta} be a set of rules, S_{Meta} be a set of subsystems, A_{Meta} be a set of commands, E_{Meta}

be a set of entities, and H_{Meta} be a set of event handlers. The Meta-Game M is defined as:

$$M = \{C_{Meta}, V_{Meta}, R_{Meta}, S_{Meta}, A_{Meta}, E_{Meta}, H_{Meta}\} \tag{B.23}$$

With A_{Meta} being a subset of V_{Meta}; E_{Meta} a set of abstract entities, each of which subset of C_{Meta}; H_{Meta} a set of abstract event handlers, each of which handles events from V_{Meta}; and with:

$$\begin{aligned} C_{Meta} &= C^{L}_{Meta} \cup C^{I-AI}_{Meta} \\ S_{Meta} &= S^{L}_{Meta} \cup S^{I-AI}_{Meta} \\ H_{Meta} &= H^{L}_{Meta} \cup H^{I-AI}_{Meta} \end{aligned} \tag{B.24}$$

For a non-interactive Meta-Game (there is not any input or output), $C^{I-AI}_{Meta} = \emptyset$, $S^{I-AI}_{Meta} = \emptyset$, and $H^{I-AI}_{Meta} = \emptyset$. Otherwise, for an interactive Meta-Game, there are AI-agents. Regardless of the case, the Meta-Game does not have any user related physical-level interactions. ◀

$$\begin{gathered} M = \{C^{Meta}, V^{Meta}, R^{Meta}, S^{Meta}, A^{Meta}, E^{Meta}, H^{Meta}\} \\ C^{Meta} \subseteq (C^{L} \cup C^{I-AI}), \quad C^{Meta} \neq \emptyset \\ V^{Meta} \subseteq V, \quad V^{Meta} \neq \emptyset \\ R^{Meta} \subseteq R, \quad R^{Meta} \neq \emptyset \\ S^{Meta} \subseteq (S^{L} \cup C^{I-AI}), \quad S^{Meta} \neq \emptyset \\ A^{Meta} \subseteq A, \quad A^{Meta} \neq \emptyset \\ E^{Meta} = \{e_1, e_2, \ldots, e_r \quad | \quad e_i \cap C^{Meta} = e_i, \quad e_i \cap e^{L}_{i} = e^{L}_{i}, e^{L}_{i} \in E^{L}\}, \quad E^{Meta} \neq \emptyset \\ H^{Meta} \subseteq (H^{L} \cup H^{I-AI}), \quad H^{Meta} \neq \emptyset \end{gathered}$$

Meta-Game

Non-interactive: $C^{I-AI} = \emptyset$, $S^{I-AI} = \emptyset$, and $H^{I-AI} = \emptyset$.

B.4.8 Games

Definition 25 (Interaction Profile)**.** An Interaction Profile is a data resource external to the application, which describes required logic, input, and output components and event handlers adaptations to convert a Meta-Game into a Game suitable for a human player. ◀

Definition 26 (Game)**.** Let C_{Game} be a set of components, V_{Game} be a set of events, R_{Game} be a set of rules, S_{Game} be a set of subsystems, A_{Game} be a set of commands, E_{Game} be a set of entities, H_{Game} be a set of event handlers, and P_{Game} be the Interaction Profile. The game G_{Game} is defined as:

$$G = \{C_{Game}, V_{Game}, R_{Game}, S_{Game}, A_{Game}, E_{Game}, H_{Game}, P_{Game}\} \tag{B.25}$$

With A_{Game} being a subset of V_{Game}; E_{Game} a set of entities, each of which subset of C_{Game}; H_{Game} a set of event handlers, each of which handles events from V_{Game}; and with:

$$\begin{aligned} C_{Game} &\supset C_{meta} \\ V_{Game} &\supseteq V_{Meta} \\ R_{Game} &\supseteq R_{Meta} \\ S_{Game} &\supset S_{meta} \\ A_{Game} &\supseteq A_{Meta} \\ E_{Game} &\supseteq E_{Meta} \\ H_{Game} &\supset H_{Meta} \end{aligned} \tag{B.26}$$

For convenience, we can consider C_{Game}, S_{Game}, and H_{Game} as all existing components, subsystems, and event handlers.

The Interaction Profile P_{Game} defines the sets of input and output components, subsystems, and events handlers to enable human interaction with the Game. ◀

$$\begin{gathered} G = \{C^{Game}, V^{Game}, R^{Game}, S^{Game}, A^{Game}, E^{Game}, H^{Game}\} \\ C^{Game} \supset C^{Meta}, \quad C^{Game} \subseteq C, \quad C^{Game} \neq \emptyset \\ V^{Game} \supseteq V^{Meta}, \quad V^{Game} \subseteq V, \quad V^{Game} \neq \emptyset \\ R^{Game} \supseteq R^{Meta}, \quad R^{Game} \subseteq R, \quad R^{Game} \neq \emptyset \\ S^{Game} \supset S^{Meta}, \quad S^{Game} \subseteq S, \quad S^{Game} \neq \emptyset \\ A^{Game} \supseteq A^{Meta}, \quad A^{Game} \subseteq A, \quad A^{Game} \neq \emptyset \\ E^{Game} \supseteq E^{Meta}, \quad E^{Game} \subseteq E, \quad E^{Game} \neq \emptyset \\ H^{Game} \supset H^{Meta}, \quad H^{Game} \subseteq H, \quad H^{Game} \neq \emptyset \end{gathered} \tag{Game}$$

B.5 Elements of a Tailorable Game

EDA, DDA, and ECS have some important similarities, including promoting decoupling, extensibility, and the ability to modify run-time execution. As each of the approaches is flexible on its own, when combined, the resulting architecture maintains these properties, allowing developers to modify a game in run-time without changing its source code. To achieve this goal, we start to discuss how to implement the formalization defined in Section B.4.

B.5.1 Component

Digital systems require data to manipulate. In an ECS, data come from components (Definition 1).

According to Definition 1, a component abstracts a single purpose for the game. It stores the required data to address a desired functionality of the system (that is, to implement a functionality and nothing else). A game has a finite set of components, which developers can combine to create entities (later defined in Definition 5). Subsystems (defined in Definition 20) process components according to the game purposes.

Types of Components

For the purposes of this formalization, it is convenient to differentiate three pairwise disjoint components types: logic components, input components, and output components.

Logic components do not store data describing *how* input or output will happen. Rather, these components describe *what* data show become information. They are, therefore, independent of physical-level interactions on their own. Thus, Definition 2 means that, from the perspective of a user, logic components are fully abstract components, as they do not have any IO-related information. In turn, this will, later, enable an IO-free simulation: instead of physical-level interactions, the simulation will define semantics.

An entity that has an input component is able to receive external input, such as the input provided by a human user to control the entity. The input component stores data to define *how* the user will control the system.

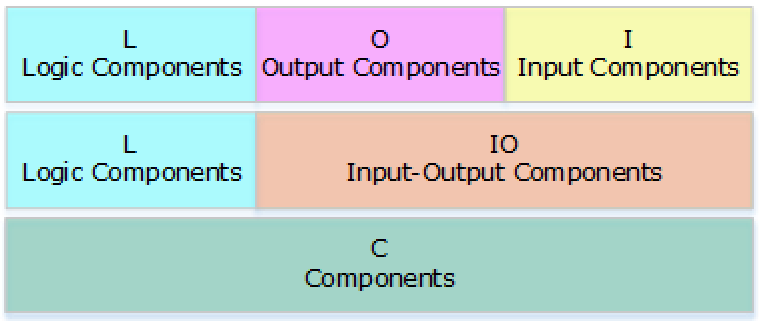
An entity that has an output component acquires output stimuli (for instance, graphical, aural, or haptic stimuli) to represent it. Similarly to input components, output components define data describing *how* to represent data as information to a user.

Considering the three types of components as well as Definition 1, it follows that it is possible to join them into a single component set without duplicates, as well as splitting this single set into three disjoint sets considering their types (Lemma 1). This is illustrated in Figure 17.

B.5.2 Entity

An entity (Definition 5) is a composition of existing components, each of which defined by following the Definition 1.

Figure 17 – Component types.



Source – Created by the author.

Note – Components are divided into three disjoint sets: logic (L), input (I), and output (O). The union for the input and output sets define the available components for human-interaction. The same applies to event-handlers.

Any arbitrary entity is a non-empty subset of existing components. This means that, given an entity E, and a set of components C, then:

$$\begin{aligned} E &\subseteq C \\ E \cap C &= E \\ E \cup C &= C \end{aligned} \tag{B.27}$$

are valid for any possible entity.

Definition 5 has an important, less obvious meaning – an entity is nothing by itself[13]. An entity is defined exclusively by its components, as they provide all its data. This is important for it implies that entities can become anything one wishes, provided that there exists required components, and they that are attached to them.

A game has a finite set of components (defined as in Definition 1) which stores its data. All game relationships derive from the data stored by components owned by the entities.

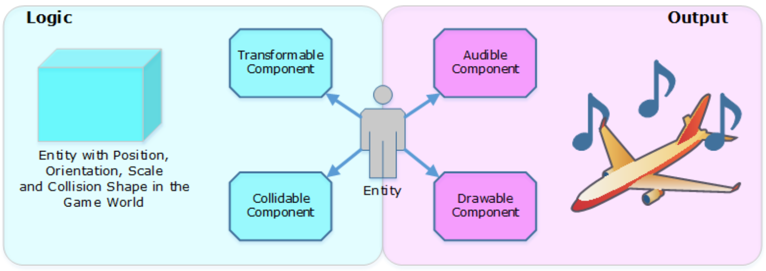
For instance, the entity in Figure 18 is defined by four components, namely: a `TransformableComponent`, a `CollidableComponent`, an `AudibleComponent`, and a `DrawableComponent`. The `TransformableComponent` stores the position, orientation and scale of an entity in the simulation world (thus, only the data required by a transform matrix). The `CollidableComponent` stores the data for physical-collection, such as physical shape and density (only the data required for the collision). The `DrawableComponent` and the `AudibleComponent` store, respectively, its graphical and aural output data (such as its

[13] Some ECSs implement an entity as an identifier (ID), for example.

corresponding image or sound, respectively; once again, only the required data to fulfill its purpose).

If one wishes to remove any of data records (alongside with the behaviors that they grant the entity with), she/he only needs to remove the corresponding component. Removing the `DrawableComponent` from the entity in Figure 18 would, in turn, remove the output graphical representation, leaving the entity with only aural representation for its output.

Figure 18 – An entity.



Source – Created by the author.

Note – An entity is a collection of components, defined as a subset from the existing components. As each component has a single, specific purpose, they are either related to logic processing, input handling, or stimuli output. As components enable an entity to perform activities, the entity in this image can: have a position, orientation, and scale in a game world (`TransformableComponent`); collide with other collidable entities (`CollidableComponent`); have a graphical output representation (`DrawableComponent`); have an aural output representation (`AudibleComponent`). However, it cannot be controlled component-wise, for it does not have an input-related component. For the Meta-Game, this entity is a box with which other collidable entities can collide; for a game (and, thus, for a user playing it), it may be a noisy airplane that other collidable entities can collide with.

B.5.3 Event

We define events in Definition 11.

In the same way a digital game has a finite set of components, it also has a finite set of events. The event itself does not alter the execution flow of a digital system. Instead, event handlers (Definition 13; functions or methods, for instance) subscribe to it, providing the required processing when it happens. As events define many-to-many relationships (Subsection B.2.7), it is not necessary to distinguish events into groups; an event might trigger, altogether, logic, input, and output functionalities, depending on its event handlers.

An event may have as many event handlers as desired. In theory, an event handler could process multiple events (provided that they are suitable and general enough). However, in practice, each event handler implemented with this formalization will be specialized to process a single event. Thus, in this formalization, an event may trigger a finite set of event handlers, and, in practice, each event handler will handle a single event[14].

Types of Event Handler

As with components, it is also convenient to differentiate three pairwise disjoint event handlers types: logic event handlers, input event handlers, and output event handlers.

As event handler types are also disjoint, they allow for a result similar to the one presented in Lemma 1, as states Lemma 4.

B.6 Operations on the Elements of a Tailorable Game

Elements cannot define a game by themselves, in isolation. Instead, they are the building blocks to allow developers to design simulations. Simulations often combine elements with each other, preferably in a non *ad hoc* way. This section details operations to change and process game data using game elements defined in Section B.5.

B.6.1 Modifying Entities

Operations such as union, intersection, and difference allow us to create new sets based on the original ones. We can extend this idea to entities and components. As an entity is a set of components (Definition 5), inclusion and removal of data members and behaviors can be implemented as simple set operations. These operations will create a new set with the properties that we want to achieve. We can add new components to an entity using set unions (then a set different to remove the original one), remove components from it by using set differences (once again, with a set different to remove the original one), or query the existence of an arbitrary component with set intersection (if the intersection is not the empty set, the entity has the queried component).

B.6.2 Modifying Event Handling

Operations to manipulate events are similar to those for manipulating components, except that they apply to event handlers instead of events themselves. Thus, we can register (add) an event handler for a given event, or unregister it (remove) from the event. Thus, for each event, we may define a set of event handlers to process it. Moreover, for each event handler, we may define a set of handled events that it processes.

[14] To define many-to-many relationships, we could define events and event handlers similarly to entities and components.

B.6.3 Defining Relationships Among Elements

At this point, we can create and modify entities, components, events, and event handlers. However, there is no way to combine them, nor there are ways of binding entities interactions to events. For this purpose, we may define rules (Definition 19). Rules govern and define the behaviors of digital games. They are akin to business rules that define the game elements that should behave in the simulation.

According to Definition 19, rules define resulting side-effects from the interaction of entities. As they result in an event, they allow we to build complex relationships among entities, and to trigger further asynchronous processing on their components.

Considering a finite set of entities $E = \{e_1, e_2, \ldots, e_n\}$ and a set of events $V = \{v_1, v_2, \ldots, v_m\}$, a binary rule has the form:

$$e_i \, R \, e_j \mapsto v_k \tag{B.28}$$

in which e_i, and e_j are entities belonging to E $(i, j = 1, \ldots, |E|)$, and v_k is an event which belongs to V $(k = 1, \ldots, |V|)$. The developer may define rules with different arities as needed (for instance, unary, ternary, or n-ary rules).

B.6.4 Data Processing for a Tailorable Game

It is necessary to process the game data stored in elements according to the defined rules. To divide types of processing according to their functionalities (input, output, or logic), we can define subsystems (Definition 20).

If an entity has a component that a subsystem processes, then it is registered on the subsystem (this will be defined in Section B.6.4). This guarantees that entities behaviors depend on their components[15].

Types of Subsystems

As with components and event handlers, it is possible to divide subsystems into three pairwise disjoint types, according to their functionalities: logic (Definition 21), input (Definition 22), and output (Definition 23).

Once again, as with components and event handlers, dividing subsystems by type ensures that logic subsystems do not process any IO data; thus, the logic subsystem is effectively abstract in terms of physical-level interactions.

[15] For ECSs that defines components with data *and* logic, subsystems are often merged into components themselves.

Subsystems Operations

A subsystem registers for itself only suitable entities, that is, entities composed by components that they process. The general registering process is the set that results from the intersection between the subsystem managed components and all existing entities (Lemma 7).

Lemma 7 ensures that each subsystem only processes entities that are supersets of components required by the subsystem. In particular, we may define Lemma 8 for logic subsystems:

A similar reasoning can ensure that the entities registered for the input or output subsystems have, at least, one input or one output component, respectively. The implication for this choice is that none of the IO subsystems can register nor process an entity composed exclusively of logic components. Effectively, the division makes logic processing disjoint of input and output ones. We can infer, thus, that a Meta-Game will use only logic components to its subsystem. To promote this idea, we might want to distinguish elements types into two types: related or non-related to user interactions.

B.7 Abstract and Concrete Elements of a Tailorable Game

After defining elements and operations for tailorable games, a necessary feature to formalization is the addition and removal of user-related interaction data and behaviors. This section introduces the concepts of abstract and concrete game elements, which act as building pieces of tailorable software that implements the formalization. Assuming that it is possible to convert abstract into concrete elements during run-time, then it becomes possible to create run-time extensible games that may consider the interaction needs of their users to provide a customized (and potentially accessible) experience.

Lemma 1 and Lemma 4 show it is possible to categorize components and event handlers according to their IO behaviors. This section uses the results presented in the previous sections to create new definitions aiming to ease comprehension and to demonstrate the validity of the proposed formalization.

B.7.1 Abstract Entity

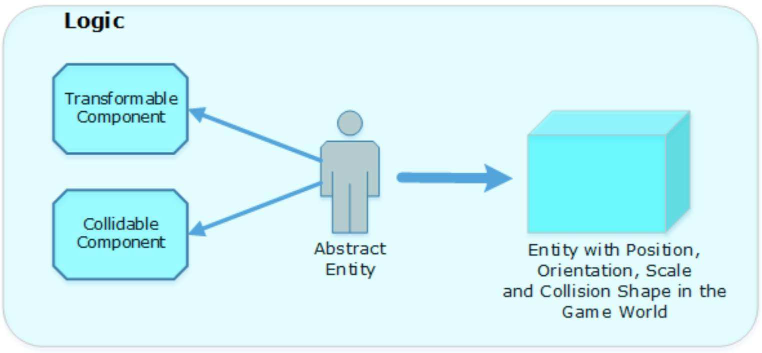
Definition 5 defines entities as sets of components. This means that it is possible to create and change an entity by altering its set of components, by adding new components to itself (Lemma 2) or removing some existing ones from it (Lemma 3). One may suggest that, provided that an entity only uses logic components, it is possible to build IO-free entities. This is indeed possible, and leads to the definition of abstract entities (Definition 6).

According to Definition 6, an abstract entity is entirely IO-free, as it does not have any input or output components. Furthermore, considering C^L as the set of logic components, C^I as the set of input components, C^O as the set of output components, and A is an abstract entity, then:

$$\begin{gathered} A \subseteq C^L \\ A \cap C^L \neq \emptyset \\ A \cap C^I = \emptyset \\ A \cap C^O = \emptyset \end{gathered} \tag{B.29}$$

For instance, the entity in Figure 19 is an abstract entity. It only has logic components – the `TransformableComponent` and the `CollidableComponent`. As such, there are no output stimuli to convey its existence to a user, neither can she/he control it, as it cannot receive input.

Figure 19 – An abstract entity.


Logic
Transformable Component
Collidable Component
Abstract Entity
Entity with Position, Orientation, Scale and Collision Shape in the Game World

Source – Adapted from Garcia (2014).

Note – An abstract entity does not have IO information. Although it exists in the game, a user cannot interact with it nor know it exists. However, as logic subsystems do not process IO data, they can process abstract entities to define the game simulation (as anticipated in Figure 18).

B.7.2 Concrete Entity

If there are entities without IO functionalities (abstract entities), we may define concrete entities with IO functionalities (Definition 7).

According to Definition 7, a concrete entity may or may not have input and output components – they are optional (thus, both entities in Figure 18 and Figure 19 are concrete entities, though only the latter is also an abstract entity). Considering I as the set of input components, O as the set of output components, L as the set of logic components, and F a concrete entity, then:

$$\begin{aligned} F &\subseteq L \\ F &\subset I \\ F &\subset O \end{aligned} \tag{B.30}$$

It is important to note that, for practical purposes in this formalization, a concrete entity has, at least, one logic component – otherwise, it would have no utility for the logic processing (in other words, its purpose would be purely aesthetic). Input and output have no practical meaning if they are not related to other data, as they would not convey nor modify any information.

B.7.3 Abstract Event Handler

The abstract event handler (Definition 17) is the event handler counterpart of the abstract component.

According to Definition 17, and following components types in Section B.5.1, should an abstract event handler need to process any data in a game, it shall only process logic components from entities.

B.7.4 Concrete Event Handler

The concrete event handler (Definition 18) is the event handler counterpart of the concrete entity.

Due to Definition 13, a concrete event handler will only perform one of the possible side-effects types. This is not an issue, as an event may have multiple event handlers – thus, when it is necessary to trigger more than one side effect for a given event, we should define a corresponding number of event handlers and register them for the event.

B.8 Conversions Between Abstract and Concrete Elements

This section is the basis for the creation of a Meta-Game. If we can create an abstract element from a concrete one, it becomes possible to convert a Game into a Meta-Game. Likewise, if we can add physical-level interactions to a Meta-Game, it is possible to convert it into, at least, one Game. The idea is to transform interactive systems with human-related interaction into simulations that abstracts human-related interaction with commands. This way, we will be free to arbitrarily re-define interaction to map how users will play a Game.

B.8.1 Converting Entities into Abstract Entities

Theorem 1 transforms a concrete entity into an abstract entity. Due to the use of sets to represent the entities, the operation is straightforward.

For instance, the entity in Figure 19 is the abstract entity representation of Figure 18. Logically, both entities are equivalent, as all logic data are present: the `TransformableComponent` and the `CollidableComponent` define its logical data. This means that its simulation in the Meta-Game is equivalent to the original one. However, the lack of IO components in the abstract entity makes it non-interactive – the user cannot perceive it, nor can she/he control it. Although it is there, the user has no way to know about it, or interact with it.

B.8.2 Converting Abstract Entities into Concrete Entities

Theorem 2 transforms an abstract entity into a concrete one. This does the inverse of the Theorem 1.

For instance, the entity in Figure 18 is a possible concrete entity for the abstract one in Figure 19. However, it is not the only: any combination of IO components will result in other equivalent representation, as suggest in Figure 20.

B.8.3 Transforming Entities

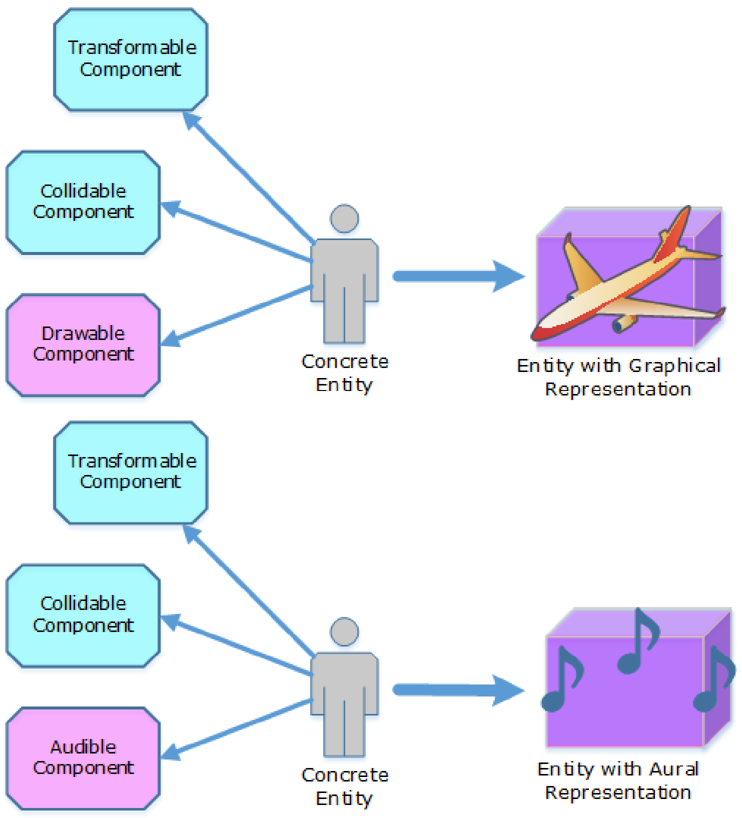
Theorem 3 combines Theorem 1 and Theorem 2 into one, showing that is possible to transform a concrete entity into an abstract one, and vice versa.

Due to Lemma 2 and Lemma 3, the Entity Transformation can be seen, informally, as the result of adding or removing required components to convert the entity.

B.8.4 Converting Event Handlers Sets into Abstract Event Handlers Sets

After transforming entities, it is necessary to transform event handlers. The reasoning and the procedures are the same. However, instead of changing an event, it will be necessary to add – or to remove – event handlers to – from – it.

Figure 20 – Concrete entities.



Source – Created by the author.

Note – Each of these entities has a different subset of IO components; thus, the way a user perceives them is different: for the top, she/he views the entity; for the bottom, she/he hears it. However, although they are different when one considers their concrete representation, they have the same abstract entity (the one in Figure 19). Thus, for the Meta-Game, they are the same.

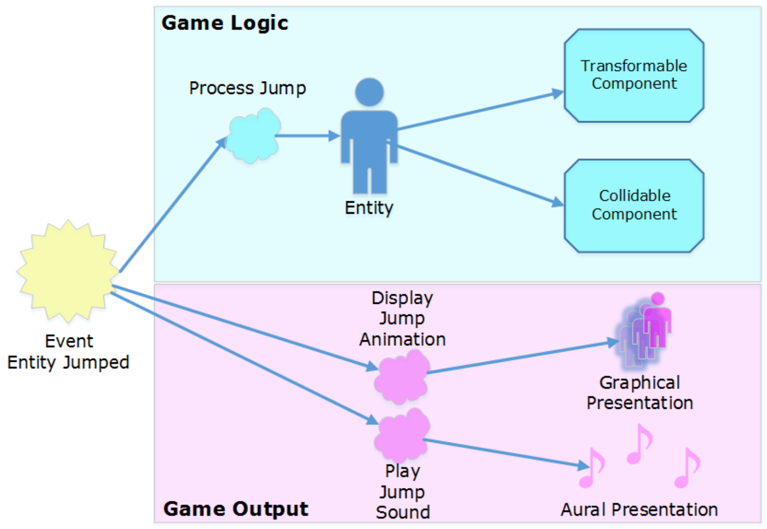
B.8.5 Converting Event Handlers Sets into Concrete Event Handlers Sets

B.8.6 Transforming Event Handlers Sets

This result is illustrated in Figure 21. The concrete version combines the top and bottom halves of the image, having both logic and output event handlers. The abstract version only has the events defined at the top half of the image. For the Meta-Game, both versions behave equally.

Due to Lemma 5 and Lemma 6, Event Handler Transformation can be seen, informally, as the result of adding or removing the required events to convert the event handlers.

Figure 21 – Events and Event Handlers.



Source – Created by the author.

Note – An event itself does not trigger further processing; it only conveys that something important happened. Event handlers to the processing, according to their goals. In the top half of the image, the game logic is updated due to the event. The data of components change as required. In the bottom half, event handlers dispatch user feedback for the event.

B.9 Architecture for Implementing Tailorable Games

Entity Transformation and Event Handler Transformation show that it is possible to add and to remove components to/from any existing entity, and event handlers to/from any existing event. Considering Definition 5 and Definition 11, alongside the proposed operations and theorems results, we can draw an interesting interpretation: it is possible to alter how entities and events are processed, and their behaviors.

According to Definition 5, the behaviors of an entity are defined by its components – as stated in Definition 1, a component holds all relevant data to allow the entity to achieve its purpose. The same reasoning applies to events: according to Definition 11 and Definition 13, an event by itself does not impose any side-effects; instead, side-effects are caused exclusively by their corresponding event handlers.

Combining the last two paragraphs, we may suggest that building a digital game using components and events can allow modifying the digital game itself. The modification only depends on which components and event handlers we add/remove to change behaviors. This statement is valid for logic, input, and output. Furthermore, it is a useful result for accessibility purposes. Provided that we implement game rules and logic processing using only logic components (Definition 2) and logic event handlers (Definition 14), the simulation will not depend on neither user input nor output. If we decouple the IO from the simulation, this could mean, in turn, that we could define suitable IO interactions by, later, choosing adequate IO-specific components and event handlers.

It is, thus, possible to define a system without user IO using entities, components, events, and event handlers. Next, it is necessary to make it interactive – both with and without users –, and to define a communication protocol for input. This section approaches each of these issues. First, we need to define an interactive system without users. The idea is letting the system to simulate itself without user interaction. In order to this, we should define an input protocol which users and machines could use alike. This would allow, at a later step, switching the interactions of "machines" to interactions of humans, defining the system input. Lastly, we have to bind the available actions to physical-level interactions, to enable the user to play. This effectively creates a human-controlled input subsystem.

The same reasoning applies to the system output. In this case, at a conceptual level, it is simpler. As machines do not need sensory stimuli to perceive the system, we only ought to add the desired output stimuli afterwards. Thus, the output is a form of sensory conveying the system current state of simulation to the user. In order to achieve this, we add the required output components to the entities, and define and register output event handlers for the events.

With the possibility of defining both input and output, it is, finally, necessary to tailor the system at run-time for the interaction needs of a user. This will use the last architectural choice for the approach – the DDA, which we can explore to transform Meta-Games into user-playable Games.

B.9.1 Input Handling

The first step to define an interactive Meta-Game is to enable non-human input. As entities, components, events, and event handlers are the only proposed elements, it is necessary to combine these elements to handle input. The interactive actor, for convenience, will be called agent (Definition 8).

As an agent has an input component, input subsystems will register it for themselves and process its input. Thus, for the system, an agent in an interactive entity. However, it is important to note that the definition is generic – it does not require the agent to be human. This branches the possible agent types into two, as it was the case with abstract and concrete, as defined in Definition 9 and in Definition 10.

The distinction is important as, in fact, the interactive Meta-Game simulation will use only AI-controlled agents. This agent will use an algorithm and AI techniques to play the game like a user, exploring the available game rules to interact with the game[16] [17] [18].

Thus, instead of a user, an AI-agent "plays" the interactive Meta-Game. An abstract simulation will have only AI-agents; on the other hand, a human playable Game will have, at least, one human-agent. The only remaining detail is how to convert between AI-agents and human-agents – each of these conversions in shown in Theorem 4 and Theorem 5, and generalized in Theorem 6 (Agent Transformation).

Input Protocol and Mechanism

At this point, when we use agents, it is possible to affirm that AI and human users are, at least theoretically, exchangeable for the implementation[19]. The next step is to use the elements to create an input protocol. The input protocol would mediate the input communication between the elements and the logic implementation.

One possible approach to implement the input protocol is to use events. With an EDA, we may define semantic commands for the input interactions (Definition 12). The Meta-Game logic, implemented using an EDA, processes the defined commands according the existing rules to change the flow of its simulation.

Thus, with commands, instead of defining how the user should interact with the game, the developers define what the user is allowed to perform when using the system. Commands define *semantic* inputs to represent *what* an agent should be able to perform in the game, instead of mapping this ability to *how* a user may perform it at physical-level. For instance, instead of binding a controller button to make an entity jump, the developers define a jump command[20]. The jump command might be performed

[16] We can see the AI-controlled input component as a logic component, as the system performs the input. In fact, it would be possible to convert it into pure AI implementation using techniques such as decision trees and finite-state machines.

[17] The AI-agent does not have to be sufficiently "intelligent" to play like a human. In fact, it *does not need* to play at all, *neither have* any sort of intelligence; rather, it only needs to *be able* to play. For this chapter, that means it only needs to be able to issue Commands (defined later in Definition 12), which we can implement as an algorithm to trigger an event. However, AI-Agents can act as computer-controlled opponents if the developers program input subsystems to do so. Likewise, the better the AI-agent, the better automation we can provide to support players with motor disabilities (or other disabilities). Moreover, if we desire to simulate specific instances of an AI-Agent acting as Human-Agent, it would be possible to record commands and time stamps on a file and replay them in a new instance of the game. In this case, the AI-Agent would not have any intelligence; however, it would be able to play the game.

[18] At times, it may be useful to consider Human-Agents as Agents in general, that is, potentially having human *and* AI-controlled input components – a hybrid. The idea is to reduce required game input, or to automate parts of the game, as potential implementations of input reduction and automation strategies described by Yuan, Folmer & Harris (2011).

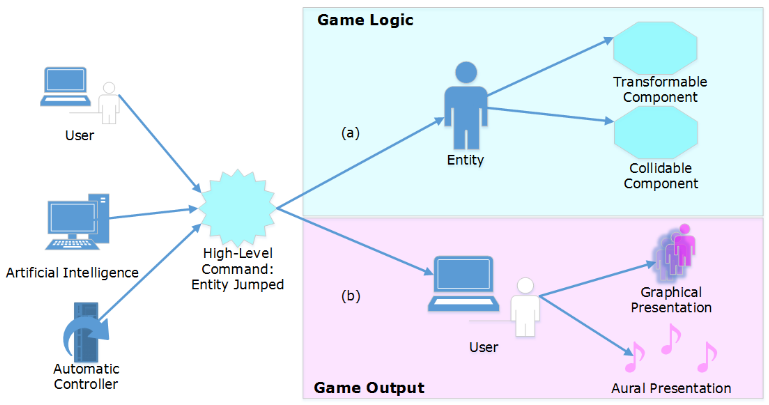
[19] Once again, only regarding what they are theoretically able to do – *semantically* – they are able to provide the game with any input required.

[20] It is important to notice that a command, in this chapter, is an event. This mean event handlers will process them after the command event occur. This also means that, potentially, many input devices or algorithms may be registered for a same command, as well as a same input mechanism (button, key, movement...) may be bound to one or more commands at different contexts.

by an agent; the game logic subsystems, in turn, process the event to allow its entity to jump.

The benefit of using event-driven input handling is that it allows different sources and devices to provide input to the system. This grants implementation flexibility, as, provided that the source is able to trigger an event, it is able to provide the system with input. As EDAs promote decoupling, it does not matter what originates the input event. It could be a human providing input via a device (or assistive technology), an AI-controlled agent, or even macros defined by a developer to play for or aid the user (Figure 22). The latter, in turn, allows the developers to define automated input schemes to change the interaction, and, if needed, aid the user in case of motor disabilities.

Figure 22 – Semantic commands with events.



Source – Adapted from Garcia (2014).

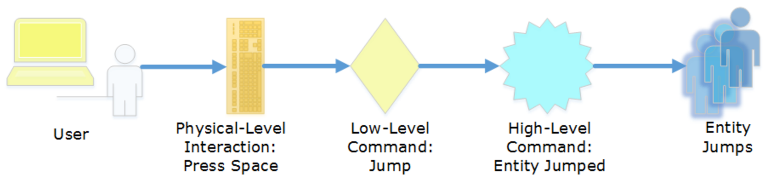
Note – Semantic commands allow developers to define *what* a user may do when interacting with the game, instead of *how* she/he should interact. Instead of forcing an arbitrary physical-level interaction for the input, this allows the definition of multiple input schemes, ranging from traditional input devices for users, to automated AI-controller to play against and for the user, promoting input automation when needed.

Input Mapping

The use of semantic commands allows developers to define what the user can do when interacting with the system. It does not impose a single way on how to do it, though. Rather, it grants the user liberty to define how to provide input to the system (with devices and mechanism) according to her/his interaction needs, via input mapping (also known as input re-mapping (MCSHAFFRY; GRAHAM, 2012; GREGORY, 2014)). Input mapping, thus, allow users to rebind existing game commands according to their preferences and needs. In this chapter, it is not enough to explore input mapping for a single device. Rather, we need to abstract the device as well.

To map a command to a specific physical-level interaction, we may define two different events per command (Figure 23). The first-event maps the physical action to a low-level event. This binds an action (such as pressing a specific key) to a possible game action (a command such as requesting the entity to jump). An input subsystem processes this event and generates corresponding high-level events (the semantic command itself) for the game logic subsystems. They process the events according to the existing rules and generate the output for the simulation (for this example, the controlled entity would jump).

Figure 23 – Input mapping.


User
Physical-Level Interaction: Press Space
Low-Level Command: Jump
High-Level Command: Entity Jumped
Entity Jumps

Source – Adapted from Garcia (2014).

Note – Semantic commands are the first step to decouple the physical-level interaction from the implementation. However, to enable run-time re-mapping, it is necessary to go a step further, using an indirect approach with two events. The first event converts the physical action into the desired input; the second event converts the input into a semantic command. This splits the input into two phases: a physical (physical-level interaction and low-level event) and a logical one (high-level event (the command) and subsystems' processing). This allows us to change the first phase without compromising or affecting the second.

We use two different events instead of a single one to allow run-time command re-mapping. As the high-level command does not depend on the physical-level interaction, it is possible to modify the input action or even the input device without affecting the remaining of the input sequence implementation. The new biding would still translate the low-level command into the respective high-level semantic command, without compromising the logic implementation.

At this point, with input mapping, the current result satisfies the input aspects of the two questions presented in Section B.1: they allow creating a system without user input, and they allow defining the user input interaction at run-time. This achieves half of the goals; next, it is necessary to tackle the output of the system.

B.9.2 Output Handling

As stated, at a conceptual level, it is simpler to tailor the output of a Meta-Game than its input. Assuming input handling is available, the simulation of the Meta-Game is already complete, orchestrated by logic subsystems processing existing rules, and with commands (events) originating the input. This makes the output a sort of digital storyteller – the functionality responsible for conveying sensory stimuli to the user; that is, what is happening at a given moment in the simulation.

As all data belong to components, non-sensory information which should be conveyed to users already exist in the system – it is stored on logic components. Thus, the goal of output systems is to translate data into accessible stimuli for presentation, exploring the available output devices (potentially including assistive technologies) and the senses of a user.

As with input, this approach defines only entities, components, events, and event handlers as primitive elements. Thus, we should define the presentation using output components and output event handlers. These operations explore Entity Transformation and Event Handler Transformation – they share the same abstract concepts in the Meta-Game, and add concrete resources to transform it into a Game. We add output components to existing entities to provide output stimuli (Figure 20), and add output event listeners to provide feedback on relevant events (Figure 21). These additions shall provide required output data for presentation; the usage of output components will allow output subsystems to register the required entities to convey the desired media.

The output data generated by output components and event handlers can complement each other. Components can hold data for continuous presentation, which we may wish to present on every update frame. This includes, for instance, images, text, and music. On the other hand, event handlers provide (usually instant) feedback when an important event happens. This may include, for instance, graphical or sound effects, and haptic stimuli such as vibration.

Output subsystems will use available output data to present the chosen representation to the user. It is important to note that, as output data do not modify nor intersect with logic data, the presentation does not change logic data at all – the data are disjoint by construction (Lemma 1 and Lemma 4), respecting output subsystems definition (Definition 23). Thus, the simulation is the same, regardless of the chosen presentation and of explored output strategies.

It is also important to note that rules, representing the entities interactions, may result in events. Thus, it is possible to provide feedback when anything of interest happens – in particular, at the exact time after it happens. Moreover, input commands are events themselves. This makes trivial to add input feedback to the system: it requires registering desired output event handlers for a corresponding command.

Therefore, *the simulation is semantic and the input is semantic; the output can be shown according to purposes defined by developers and needs of users*. As such, we can define different output schemes to see, hear, smell, touch, or taste the moment of the simulation – it only depends on available output components, subsystems, event handlers, and output devices. The system logic implementation is not, at this point, the limitation for the output. Rather, the available output devices, technologies, components and event handlers determine how it is possible to tailor for the presentation.

At this point, the results satisfy the two questions presented in Section B.1 – they allow creating a system without user input and output, and they allow defining the IO interaction at run-time. It is time to include the user back into action, using the DDA.

B.9.3 Run-Time Tailoring

The previous results allow one to affirm that it is possible to build an interactive Meta-Game. Entities, components, events, and event handlers define a fully extensible and interactive run-time architecture, which allows us to modify logic, input, and output behaviors.

The interactive Meta-Game itself does not have user interaction: it is a game for machines. Rather, it simulates the interaction using AI-agents providing input commands, without the compromise of outputting stimuli (because the machine does not need the output). A real user, however, has specific interaction abilities and capabilities; it is necessary to address them adequately.

From the previous subsections, we can create, implement, and add IO components and events to define the interaction. The goal of the additions is to convert the abstract elements into concrete ones, that is, to transform the Meta-Game into a Game.

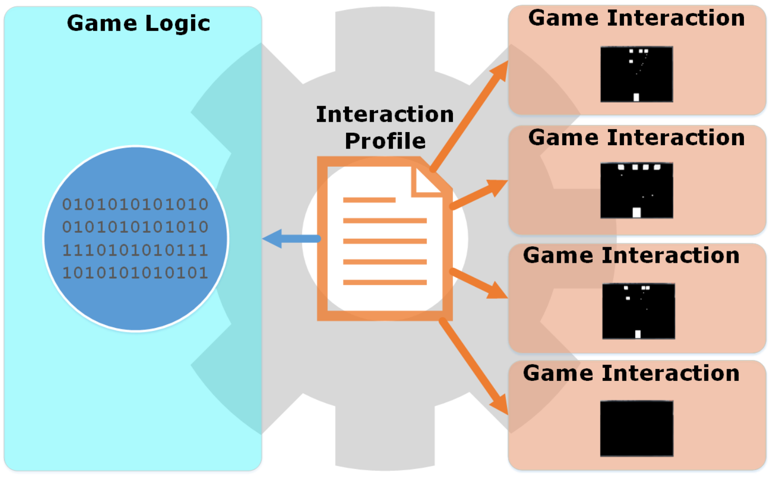
This justifies the choice of a DDA. We can use an external data resource – the Interaction Profile (Definition 25) – to represent required IO specializations, that is, necessary IO inclusions to adapt the system to the user (Figure 24).

The Interaction Profile is an external data resource, such as a plain-text file, a structured text-file, such as XML or JavaScript Object Notation (JSON), or a database. Although the format may vary, its goal is to list, for each entity and event present in the Meta-Game, which components or event handlers, respectively, should be added to define the system IO. The Interaction Profile, therefore, represents the necessary set-up to transform the Meta-Game into a potentially accessible game, allowing a user to interact with a full Game (Figure 25).

Listing 1 outlines an example of Interaction Profile – it is based on the author's UGE game engine (GARCIA, 2014). In addition to tailoring data, this Profile stores options to customize the game logic. It groups the tailoring information into a few groups:

General Settings. This group may store general data, such as the desired language for the game.

Figure 24 – Interaction Profile.



Source – Created by the author.

Note – The Interaction Profile represents the necessary adaptations to convent a Meta-Game into a Game, aiming to meet interaction needs or a user (or group of users).

Input Settings. This group defines desired input subsystems (provided that there are multiple possibilities for a given set of input components), and defines the desired mapping between physical-level interactions for input and game commands. Here the user can specify, therefore, desired input devices, keybindings, and actions to the game with input. These changes do not affect the game logic.

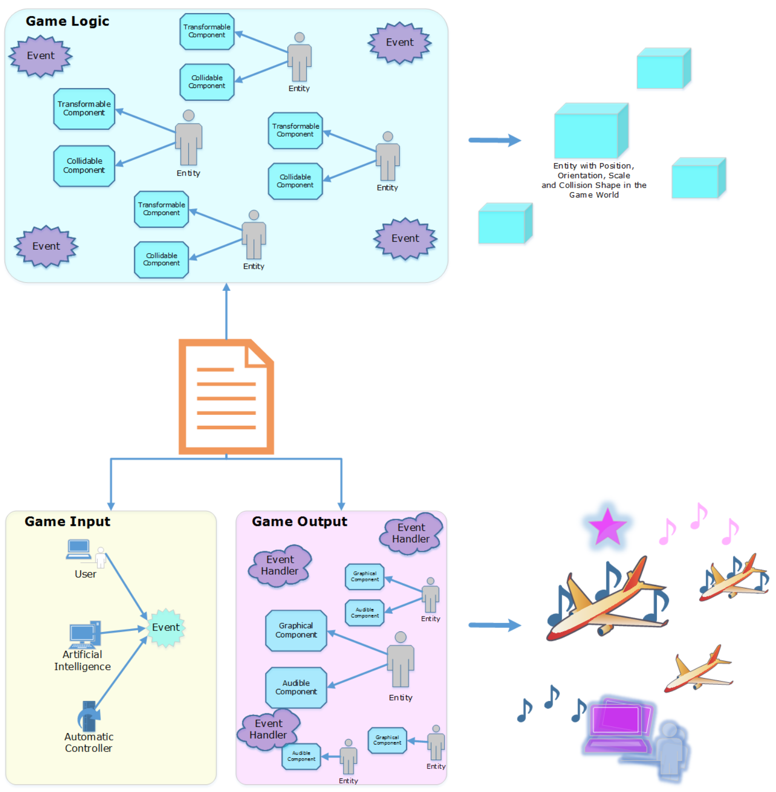
Output Settings. This group represents desired output subsystems (provided that there are multiple possibilities for a given set of output components), output components to add to each entity, output event handlers to register for the events. It may also allow further customization, such as typefaces and sizes of fonts, and window size. As with the input settings, these changes do not affect the game logic.

Gameplay Settings. This allows modifying game logic settings for gameplay, such as speed, game constants, and difficulty level.

Listing 1 – XML Interaction Profile.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PlayerProfile name="Default"
               resource="profiles/default/profile.xml">
 <GeneralSettings>
  <PlayerPreferences
```

Figure 25 – Profile based tailoring.



Source – Created by the author.

Note – As logic components and logic event handlers do not depend on IO, the Meta-Game simulation can be the same, regardless of physical-level interactions. The interaction profile allows the conversion from a Meta-Game into a system, describing required IO data and *how* a user should interact with the system. Thus, the logic is always the same (left side of the figure); however, they physical-level interactions may vary according to the users' interaction needs (some possibilities are outlined at the right side of the figure).

```
  resource=
   "profiles/default/general/player_preferences.xml"/>
 </GeneralSettings>

 <InputSettings>
  <InputSubsystems
  resource=
   "profiles/default/input/input_subsystems.xml"/>
  <InputMapping
  resource=
  "profiles/default/input/input_mapping.xml"/>
 </InputSettings>

 <OutputSettings>
  <OutputSubsystems
  resource=
  "profiles/default/output/output_subsystems.xml"/>
  <EntitySpecialization
  resource=
  "profiles/default/output/entity/entities.xml"/>
  <EventSpecialization
  resource=
  "profiles/default/output/events/events.xml"/>
  <TextSettings
  resource=
  "profiles/default/output/text.xml"/>
  <WindowSettings
  resource=
  "profiles/default/output/window.xml"/>
 </OutputSettings>

 <GameplaySettings>
  <AutomationSpecialization
  resource=
  "profiles/default/gameplay/automation/automation.xml"/>
  <EntitySpecialization
  resource=
  "profiles/default/gameplay/entity/entities.xml"/>
  <EventSpecialization
  resource=
  "profiles/default/gameplay/events/events.xml"/>
  <ProjectionSpecialization
  resource=
  "profiles/default/gameplay/projection.xml"/>
  </GameplaySettings>
</PlayerProfile>
```

Note – The Interaction Profile describes how to adapt the Meta-Game to convert it into a Game. For accessibility related tailoring, it requires how to add physical-level input and output interactions to the Game. However, it may also be used to parameterize the game logic, should the developers wish to do so.

As the Profile is used as an external input resource for the system, it is possible to define many of them, such as in Listing 2. The variability is important, as they allow the user to choose the best one for her/his interaction needs. The complexity and granularity of a Profile may vary from a general use case, such as trying to include the largest extension possible of users, to specific interaction needs, ranging from a given disability to a user-level Profile.

Listing 2 – XML Profile List.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PlayerProfiles resource="profiles/profiles.xml">
 <PlayerProfile
  name="Default"
  resource="profiles/default/profile.xml"/>

 <PlayerProfile
  name="Cognitive Impairment"
  resource="profiles/cognitive_impairment/profile.xml"/>

 <PlayerProfile
  name="Motor Impairment"
  resource="profiles/motor_impairment/profile.xml"/>

 <PlayerProfile
  name="Visual Impairment: Blind"
  resource="profiles/blindness/profile.xml"/>

 <PlayerProfile
  name="Visual Impairment: Low Vision"
  resource="profiles/low_vision/profile.xml"/>

 <PlayerProfile
  name="Franco's Personal Profile"
  resource="profiles/users/franco/profile.xml"/>
</PlayerProfiles>
```

Note – Developers may define as many Interaction Profiles as they wish. Each Profile will adapt the game according to the defined IO specifications.

When a game application parses the Interaction Profile, it adds the chosen elements to their respective targets – a Factory may be useful pattern for this. As entities and events hold sets, the addition is a simple algorithm (Listing 3) – the chosen architectures and the combined approach make run-time tailoring a relatively simple task. This results in data-driven defined (and tailored) entities and events.

Listing 3 – Run-time tailoring algorithm.

```
Open the Interaction Profile with reading permissions

# Tailoring the Meta-Game input.
Read the Input Settings Element
  Read the Input Subsystems Element
    For each Subsystem in the Input Subsystems Element
      Instantiate the corresponding Subsystem implementation
      Add the instance to the available Input Subsystems

  Read the Input Mapping Element
    For each Mapping in the Input Mapping Element
    For each Entity in the Entity Specialization Element
      For Input Component in the Entity
        Instantiate the corresponding Input Component implementation
        Add the Input Component to the Entity

# Tailoring the Meta-Game output.
Read the Output Settings Element
  Read the Output Subsystems Element
    For each Subsystem in the Output Subsystems Element
      Instantiate the corresponding Subsystem implementation
      Add the instance to the available Output Subsystems

  Read the Entity Specialization Element
    For each Entity in the Entity Specialization Element
      For each Output Component in the Entity
        Instantiate the corresponding Output Component implementation
        Add the Output Component to the Entity

  Read the Event Specialization Element
    For each Event in the Event Specialization Element
      For each Output Event Handler in the Event
        Instantiate the corresponding Output Event Handler implementation
        Register the Output Event Handler for the Event

```

```
# At this point, the Meta-Game was converted into an interactive Game.

# Initialize other settings, such as general and gameplay settings.
↪  Albeit not required for IO tailoring, this is important for
↪  accessibility purposes.
Close the Interaction Profile
```

Note – The application parses the Interaction Profile, adding the specified components and event handlers to convert the Meta-Game into a Game.

As the tailoring algorithm operates at run-time, this process results in run-time tailoring. The tailored Meta-Game results in a full-fledged Game. For the user, it behaves like any other game, except it may suit her/his needs perfectly. Developers, however, may create new IO components and event handlers to continuously improve the Game – both for new users and interaction needs and for old users, as the new elements can also be used to improve older Profiles.

B.9.4 Meta-Game and Game

At this point, it is possible to define Meta-Games (Definition 24) and Games (Definition 26). They group all elements, concepts, and results described in this chapter.

The conversions between each definition use the theorems to convert abstract elements into concrete elements, and to convert AI-agents into human-agents. Converting abstract into concrete results in the Game; converting between concrete into abstract results in the Meta-Game.

B.9.5 Two Steps to Universal Access Implementation

When we analyze the mathematical part of Definition 24 and of the Definition 26, we can note that differences between them are the existence of (or lack thereof) the Interaction Profile and of IO components, event handlers, subsystems. The IO is, therefore, fully decoupled from the implementation in the approach.

The creation of the Meta-Game can benefit from the same approaches used for the run-time tailoring. This brings the benefits of the DDA to the initialization of the logic; it is outlined in Listing 4. It is not necessary, though; we can skip the reading from the external data resource and hard-code the initialization into the application.

Listing 4 – Meta-Game to Game algorithm.

```
# Initialization.
Register all available Components
Register all available Events
Register all available Event Handlers
Register all available Subsystems

# Create the Meta-Game simulation.
# Data-driven Logic.
Open the Logic Resource with reading permissions

Read the Logic Settings Element
  Read the Logic Subsystems Element
    For each Subsystem in the Logic Subsystems Element
      Instantiate the corresponding Subsystem implementation
      Add the instance to the available Logic Subsystems

  Read the Abstract Entity Definitions Element
    For each Abstract Entity in the Abstract Entity Definitions Element
      For each Logic Component in the Abstract Entity
        Instantiate the corresponding Logic Component implementation
        Add the Logic Component to the Abstract Entity

  Read the Event Definitions Element
    For each Event in the Event Definitions Element
      For each Logic Event Handler in the Event
        Instantiate the corresponding Logic Event Handler implementation
        Register the Logic Event Handler for the Event

  Read the Command Definitions Element
    For each Command in the Command Definitions Element
      Find the required Event for the Command
      Bind the Command to the Event

Close the Logic Resource

# Create the Game.
# Add physical-level interactions to transform the Meta-Game into a Game.
# The application parses the Interaction Profile to define the desired
↪  IO.

Run the Tailoring Algorithm
```

```

# At this point, the Meta-Game was converted into an interactive system:
↪  the Meta-Game became a Game with which the user can interact.
```

Note – The first step is to instantiate the Meta-Game. Afterwards, in the second step, the application parses and instantiates the Interaction Profile, using Listing 3.

The first step defines the Meta-Game: the entire Game simulation is created. The simulation is the same for any possible physical-level interactions. Thus, the application logic is coded once. Afterwards, by creating the implementation for the IO and applying it to the Meta-Game with the tailoring algorithm (Listing 3), the second step creates an interactive Game for the user.

As the approach is based on sets, it is also possible to define a development method to the implementation, as outlined in Listing 5.

Listing 5 – Development method algorithm.

```
# Use the appropriate constraint to the end.
Repeat until Developers and Users are Happy

Brainstorm a new Functionality
# (Idea, Mechanic, Target User, Interaction Need...)

Decompose the Functionality into Elements
# (Entities, Components, Events and Event Handlers)

For each Element in the Functionality
  Classify the Elements into a Category
  # (Logic, Input, or Output)

For each Element in Logic
  If the Element already Exists
    # It was already implemented before.
    Reuse the existing Element
  Else
    Develop the Element
    Register the Element into the Application
    Create (or Modify) Logic Subsystem(s) to Handle the Element
    # Modify if it is related to existing Functionalities.
    Register the Subsystem into the Application

```

```
# Use the same steps for IO.
# The category ensures the new Functionality will be disjoint from the
↪   others.

For each Element in Input
  If the Element already Exists
    Reuse the existing Element
  Else
    Develop the Element
    Register the Element into the Application
    Create (or Modify) Input Subsystem(s) Handle the Element
    Register the Subsystem into the Application

For each Element in Output
  If the Element already Exists
    Reuse the existing Element
  Else
    Develop the Element
    Register the Element for the Application
    Create (or Modify) Output Subsystem(s) to Handle the Element
    Register the Subsystem into the Application

# Test the new created content.
# If it involves IO, include accessibility and usability tests, etc.
```

Note – The set definition of the proposed approach allow for the definition of some steps for the development.

By applying the method in Listing 5, in every iteration of the development, the number of available entities, components, events, and event handlers will increase; it is important to note that, as input, output, and logic functionalities are categorized, the Meta-Game is defined implicitly. As every iteration reuses existing elements, it may take less effort to develop a new functionality. However, when a functionality is entirely new, it may require implementing all elements – this may happen, unfortunately, to accessibility functionalities.

However, due to reusing functionalities, it is possible that every new functionality may improve the experience for more users than the target ones. This will become clearer in the next section, as it illustrates the development method.

B.10 The Game to the Player: Iterative, Incremental, and Mutualistic Universal Access

Interaction Profiles enable developers to design multiple interaction schemes to convert a Meta-Game into (potentially several) Games. Provided the implementation follows the approach described so far, developers code the logic – Meta-Game – only once. Afterwards, they are able to define accessible versions of the Game by creating new and/or combining existing IO functionalities. Users may pick an existing Interaction Profile to play a given Game, or even select IO functionalities to play a personalized, self-tailored Game. Moreover, as the adaptation occurs at run-time, users are also able to customize IO whilst the application is running – combine and recombine elements to tweak the interaction, evaluating the results in real-time, without restarting the application.

If developers follow Listing 5, the first version of their product can be, simultaneously, a conventional Game and a Meta-Game[21]. Particularly, this implies that creating a Meta-Game can be an opportunity cost of the traditional development process. Following Listing 5, provided they decouple IO implementation from the game logic using components and event handlers, the Meta-Game is the conventional game stripped of its IO components and event handlers. Due to Theorem 1 and Theorem 7, this is as simple as removing IO-related elements to convert concrete into abstract elements.

The simplicity of obtaining the Meta-Game does not imply that accessibility functionalities are free, though: accessible specializations requires fully implementing new IO-related functionalities; therefore, improving accessibility still requires (considerable) efforts. Fortunately, this chapter's approach may ease the process. Although it does not automate it, all chosen development architecture promote software reuse by themselves and combined. After implementing components and event handlers, any Interaction Profile – new and exiting alike – may benefit from the new and improved functionality to improve the overall game accessibility and quality. Developers – and even users themselves – may combine existing elements to change how they play and perceive the resulting Game. In other words, this makes improving accessibility an iterative, incremental, and mutualistic process: small improvements may be (re-)combined to achieve greater improvements.

In this section, we illustrate the suggested development approach and accessibility improvement process using a simple clone of Atari Inc's Pong® as a viability study. Pong® is a game in which two players controls their paddles, hitting a ball back and forth, trying to make the ball avoid the paddle of their opponents to score a point. The game occurs

[21] Conceptually, a Meta-Game always exist for a Game. However, if IO functionalities are not decoupled from logic, decoupling it may require significant refactoring efforts – including architectural changes.

in a virtual field; when the ball hits the extremes orthogonal to paddles, it rebounds, changing direction.

Although mechanically simple, Pong is a good viability study. First, it is a well-known game – thus requiring little explanation. Second, and more importantly, it provides typical game functionalities, including real-time game loop, graphics, sounds, and user input, which allows showing how to include interactions to tailor a Meta-Game. The source code for the implementation is available at <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong>[22]. To avoid confusion with the original game of Atari, in this chapter we will call, hereafter, the prototype Ping-Pong. Ping-Pong will pass through several IO specializations in the following subsections; Figure 26 illustrates some of them.

We have also provided a second viability study exploring a more complex game – a clone of Access Invaders (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011) –, which is available at <https://gitlab.com/francogarcia/GamesRunTimeTailorability>[23] [24]. Both prototypes were created using the Godot game engine[25] without modifying it. Rather, we implemented the approaches described in this chapter to create the prototypes using the scripting language (`GDScript`) of Godot[26]. This choice illustrates that all tailoring operations can convert Meta-Game into Games (and back to Meta-Game) during run-time[27] [28].

[22] The implementation of the tailoring algorithm is available at <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong/blob/master/game/engine/tailoring/tailoring.gd>.

[23] This prototype has a few game states, including a menu to illustrate the approach in non-gaming digital systems.

[24] The tailoring implementation is available at <https://gitlab.com/francogarcia/GamesRunTimeTailorability/blob/master/game/scenes/tailoring/tailoring.gd>. A simplified, albeit functional version of the code is available choosing the commit `b017ac41bcf8a868a49cf6f426220518f4b833dc` of the repository (<https://gitlab.com/francogarcia/GamesRunTimeTailorability/tree/b017ac41bcf8a868a49cf6f426220518f4b833dc>). For instance, the tailored game scene can be accessed at <https://gitlab.com/francogarcia/GamesRunTimeTailorability/blob/b017ac41bcf8a868a49cf6f426220518f4b833dc/game/scenes/usual_game/usual_game.gd>

[25] <https://godotengine.org/>

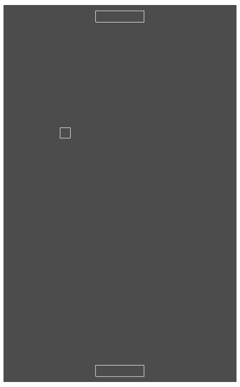
[26] It is important to note that Godot scene hierarchy is tree-based instead of set-based; thus, components are added to entities as leaf nodes at the same level (therefore, an entity is their root), avoiding duplicates to assure each component is unique in a given entity. The same reasoning allow for using other data structures (such as hash-tables) and successfully applying the approach described in this chapter.

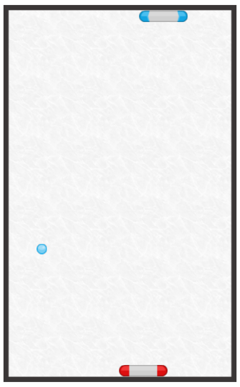
[27] Our Access Invaders clone includes a graphical user interface to build custom Interaction Profiles, instead of requiring manual changes of the profile resources.

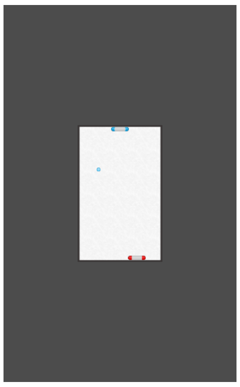
[28] A third example is our original work implementing an C++ engine, UGE, cited at Subsection B.2.5. This chapter approach, however, simplified UGE core tailoring strategies into the proposed definitions and theorems.

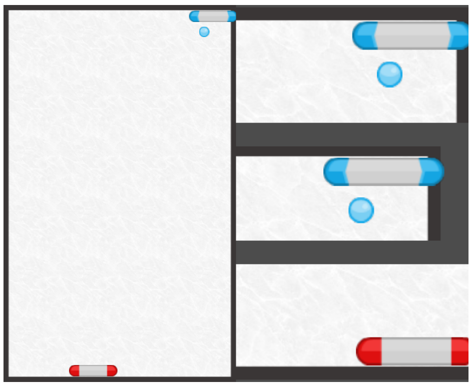
Figure 26 – Meta-Game and Games (with graphical effects) created in this section.

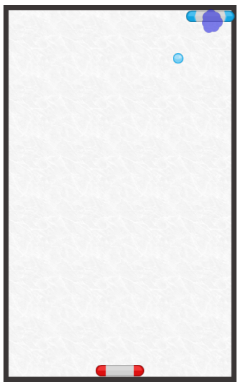
(a) Meta-Game.

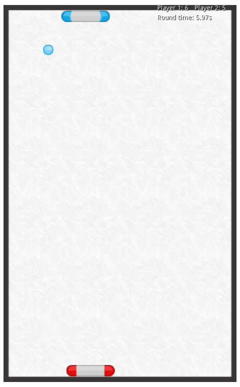

(b) Meta-Game (debug drawings).

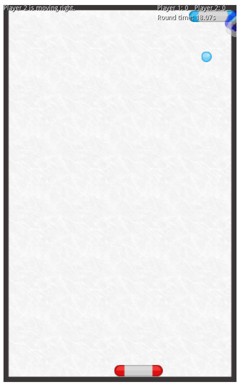

(c) Usual Game.


(d) Zoom.


(e) Zoom with multiple viewports.


(f) Graphical effects.


(g) Head-up display.


(h) All specializations.

Source – Created by the author.

Note – The Meta-Game (Figure 26a) is present in all games, regardless of specialization. Even if no human player can interact or perceive it, it is running behind the scenes (Figure 26b). Interaction Profiles add input and output functionalities to the Meta-Game, defining how one sees, hears, smells, touches, speaks, and interacts with a possible Game derived from the Meta-Game. The Game with sound effects was not included as sounds cannot be print – the graphical effects one (Figure 26f) replaces it –, nor the Games with input automation, as it would take several images to suggest the movement of the paddle.

B.10.1 First Iteration: Creating the Original Game (Conventional Game) and Obtaining the Meta-Game

In a conventional implementation, the first iteration defines the (only) game. Game and Meta-Game are the same: the Meta-Game lies hidden into the Game, as developers defined IO suitable for their intended audience. However, if developers followed the steps of Listing 5, they can untangle Game and Meta-Game from each other. In this pass, developers defined rules, events (including game commands), event handlers, components, and entities. Due to the proposed division, elements can be split into three disjoint groups: logic, input, and output. Thus, instead of a Game, after removing IO functionalities, the first iteration defined the Meta-Game and its first specialization: the conventional Interaction Profile of the Game.

Removing IO Elements from the Conventional Game: A Generic Introduction

The Meta-Game gathers the logic of the implementation: namely logic components, event handlers, and subsystems. Rules define existing relationships between entities. Events include the available command for game input. Definition 24 proposed the Meta-Game as:

$$M = \{C_{Meta}, V_{Meta}, R_{Meta}, S_{Meta}, A_{Meta}, E_{Meta}, H_{Meta}\} \tag{B.31}$$

The Meta-Game is unique, the common logic core of the application, serving as basis for every possible playable game derived from it. The first Game extends the Meta-Game with input and output elements, and an Interaction Profile, as in the previous Definition 26:

$$G^1 = \{C^1_{Game}, V^1_{Game}, R^1_{Game}, S^1_{Game}, A^1_{Game}, E^1_{Game}, H^1_{Game}, P^1\} \tag{B.32}$$

Supposing a typical 2D game, some examples of output components could include:

`DrawableComponent` stores a 2D image (such as a sprite) to visually convey how an entity should look like.

`WritableComponent` stores a string to describe an entity with words.

`AudibleComponent` stores a monaural sound for continuous playback to describe how an entity sounds.

These components would require output subsystems for processing:

`DrawingSubsystem` draws entities with `DrawableComponents` into the screen. It reads the position data from a `TransformableComponent` and the graphical data from a `DrawableComponent` to draw an image to the screen.

`TextSubsystem` draws entities with `WritableComponents` to the screen. It reads the text from the data of the component, and draws it to the screen with a default font typeface and size.

`AudioSubsystem` playbacks entities with `AudibleComponents` to the speakers. It fetches the sound to reproduce from the data of the component.

For human input, possibilities could include:

`HumanInputComponent` human input from a game controller;

`InputSubsystem` receives input from a game controller attached to the `HumanInputComponent` of a human-agent and converts it to commands.

Thus, the components and subsystems for the first Game include, respectively:

$$\begin{gathered}C^1_{Game} = C_{Meta} \cup \{\texttt{DrawableComponent}, \texttt{WritableComponent},\\ \texttt{AudibleComponent}, \texttt{HumanInputComponent}, \ldots\}\\ C_{Meta} = \{\texttt{TransformableComponent}, \ldots\}\\ S^1_{Game} = S_{Meta} \cup \{\texttt{DrawingSubsystem}, \texttt{TextSubsystem},\\ \texttt{TextSubsystem}, \texttt{AudioSubsystem}, \texttt{InputSubsystem}, \ldots\}\end{gathered} \tag{B.33}$$

As this example suggests, the Game contains the Meta-Game; thus $G^{1'} \cap M = M$, after the conversion of concrete into abstract elements $(G^{1'})$. Therefore, all elements of G^1 either are a super-set or are equal to their respective elements in M.

Instancing the Generic Introduction: the First Iteration of Ping-Pong

The first step to create Ping-Pong is identifying existing game elements and rules, which provides the game mechanics. The brief description of the original Pong® in the beginning of this section contains four entities:

1. Two paddles, which are the game agents;
2. A court (hereafter called field), on which parallel extremities to the paddles are goals (hereafter called goals), and orthogonal parts are rebounding areas (hereafter called walls);

3. A ball, which moves on the game field, rebounding when hitting either a paddle or a wall.

The second step is decomposing entities into entities and components of ECS. Analyzing the entities and their behaviors, we may identify the following functionalities abstracting behaviors:

1. Positions, orientations, and velocity in the field (the game world);
2. Controllers to move the paddles parallel to the goals;
3. Images to represent the entities.

Using the components defined previously in this chapter, `TransformableComponent`, `CollidableComponent`, `HumanInputComponent`, and `DrawableComponent` would cover the data for all functionalities except velocity. For velocity, we could define a new component – hereafter `KinematicComponent` –, storing a velocity vector (as well as, perhaps, an acceleration vector) to address the shortcomings. Thus, set-wise, concrete entities for a conventional Ping-Pong game are:

- A paddle is a set of {`TransformableComponent`, `CollidableComponent`, `KinematicComponent` `DrawableComponent`, and `HumanInputComponent`}.
- A ball is a set of {`TransformableComponent`, `CollidableComponent`, `KinematicComponent`, and `DrawableComponent`}.
- The field (including the goal and walls) is a set of {`TransformableComponent`, `CollidableComponent`, and `DrawableComponent`}.

The third step is identifying, gameplay-wise, what moments are important during a play session of the game. These moments will define game mechanics, and, thus, its rules, commands, and events. For Ping-Pong, these moments are:

1. When the ball is released (`EventBallReleased`, moment which the rule *goal* and the rule *game start* triggers);
2. When a player (either human or AI) scores a goal (`EventGoal`, triggered by the rule *ball hits goal* – which, in turn, triggers the rule *goal*);
3. When a ball hits a paddle (`EventBallHitPaddle`), which the rule *ball his paddle* triggers;
4. When a ball hits a wall (`EventBallHitWall`, triggered by the rule *ball hits wall*);

5. Whenever the ball moves (`EventBallMoved`, which occurs from the logic subsystem, due to the rule *ball moves whenever it has velocity*);
6. Whenever the paddle moves (`EventPaddleMoved`, triggered when the game logic subsystem accepts a valid input command, due to the rule *paddle moves towards an orientation whenever it is within – and not colliding with – the walls*).

Next, in the fourth step, it is necessary to determine *what* users can do when playing: the game commands, that is, semantics of what they intend to do, along with their choices. For Ping-Pong, the only interaction is moving the paddle parallel to the goals. Assuming the goals are at north and south sides of the field (hereafter, we assume that north is top and south is bottom), gameplay commands are:

1. Move paddle west (left, for simplicity);
2. Move paddle east (right, for simplicity).

As the player intent for both inputs is the same, We could generalize them into a single parameterized game command: `EventMovePaddle(paddle, direction)`. For a simple Pong® clone, those components and events should suffice to move towards implementing the Meta-Game and the (first) Game.

Implementing Ping-Pong in Godot

Whenever using a programming framework, it is useful to explore its resources instead of "re-inventing the wheel". For instance, Godot offers `Position2D`, which we can use as the `TransformableComponent`, and `Sprite`, suitable to be a `DrawableComponent`, as the rendering system of Godot does not modify position (logic) data. Creating non-existing components is not an issue, though, as they are data-only – Listing 6 shows the definition of the logic component `KinematicComponent`.

Listing 6 – Kinematic Component.

```
var m_Direction = Vector2(0, 0)
var m_Acceleration = Vector2(0, 0)
var m_Speed = Vector2(0, 0)
```

Note – For this chapter, ECSs components are data only; thus, to define new components, it is only necessary to identify and group together relevant data to perform a single purpose.

The fifth step is creating subsystems to handle the game elements, to implement the rules, and, thus, the game mechanics. To implement Ping-Pong logic, we identify its logic operations, processing suitable logic elements into one (or more) logic subsystem(s) to implement the rules. Due to the simplicity of the logic of Pong®, a single subsystem may implement the core mechanics – collisions handling, real-time movements, and *responding* (*reacting*) to game commands. Listing 7 provides examples of how to implement some of these functionalities. The responding part is essential – the game logic is reactive: it is not providing input to the Meta-Game, only responding to it. Consequently, *logic subsystems only react to game commands, not to user input directly*. Although subtle, this choice decouples input from the logic by letting input subsystems dispatch game commands whenever they detect an input from whatever source. This was explored in Section B.9.1 to allow mapping input to different physical-level interactions.

Listing 7 – Logic Subsystem.

```
func _process(delta):
    # ...

    # Fetch data from the components.
    var fieldSize = Vector2(get_node("Field/Collidable").m_Width, get_node("Field/Collidable").m_Height)# get_viewport().get_rect().size

    var ballPosition = get_node("Ball/Transformable").get_pos()
    var ballSize = Vector2(get_node("Ball/Collidable").m_Width, get_node("Ball/Collidable").m_Height)

    var player1Position = get_node("Player1/Transformable").get_pos()
    var player1Size = Vector2(get_node("Player1/Collidable").m_Width, get_node("Player1/Collidable").m_Height)

    var player2Position = get_node("Player2/Transformable").get_pos()
    var player2Size = Vector2(get_node("Player2/Collidable").m_Width, get_node("Player2/Collidable").m_Height)

    var ballRectangle = Rect2(ballPosition - 0.5 * ballSize, ballSize)
    var player1Rectangle = Rect2(player1Position - 0.5 * player1Size, player1Size)
    var player2Rectangle = Rect2(player2Position - 0.5 * player2Size, player2Size)

    # Rule: move ball whenever it has non-zero velocity.
    ballPosition += m_BallDirection * m_BallSpeed * delta
    # Rule: detect ball collision with walls.
    if (ballPosition.x > fieldSize.x):
        ballPosition.x = fieldSize.x
        m_BallDirection.x *= -1
        g_MetaGameEvents.emit_signal("EventBallHitWall", k_WallRight,
        ↪ ballPosition)
    elif (ballPosition.x < 0):
        ballPosition.x = 0
        m_BallDirection.x *= -1
        g_MetaGameEvents.emit_signal("EventBallHitWall", k_WallLeft,
        ↪ ballPosition)

    var bBallHitPlayer1 = player1Rectangle.has_point(ballPosition)
    var bBallHitPlayer2 = player2Rectangle.has_point(ballPosition)
    # Rule: detect ball collision with paddles (rebound).
    if ((bBallHitPlayer1) or (bBallHitPlayer2)):
        m_BallDirection.y *= -1
        m_BallSpeed *= 1.1
        if (m_BallSpeed > k_MaxBallSpeed):
            m_BallSpeed = k_MaxBallSpeed
        if (bBallHitPlayer1):
            g_MetaGameEvents.emit_signal("EventBallHitPaddle",
            ↪ k_Player1, ballPosition)
        else:
            g_MetaGameEvents.emit_signal("EventBallHitPaddle",
            ↪ k_Player2, ballPosition)
    else:
        get_node("Ball/Transformable").set_pos(ballPosition)
        g_MetaGameEvents.emit_signal("EventBallMoved", m_BallDirection)

    # Rule: detect ball collision with walls (goal).
    if (ballPosition.y > fieldSize.y):
        reset_positions()
        reset_kinematics()
        m_Player1Score += 1
        g_MetaGameEvents.emit_signal("EventGoal", k_Player1)
    elif (ballPosition.y < 0):
        reset_positions()
        reset_kinematics()
        m_Player2Score += 1
        g_MetaGameEvents.emit_signal("EventGoal", k_Player2)
```

Note – This snippet shows how a logic subsystem processes and updates its entities. Whenever a rule occurs, it dispatches an event to signal its occurrence.

The sixth step is creating IO subsystems[29]. For input, each paddle receive an input component. An `HumanInputComponent` is attached to the paddle entity of human players, whilst paddles controlled by AI-agent receive an `AIInputComponent`. Input subsystems handle those entities, either receiving input from a device of the user and converting it to a game command (assuming it handles the `HumanInputComponent`), or running an algorithm to provide the game command. Listing 8 illustrates these possibilities: in both cases, the input subsystem does not (and must not) modify logic data. Although the AI-agent, for instance, may access entities relevant data (such as positions from `TransformableComponent`) to perform suitable calculations, it must not modify them. The logic subsystem in Listing 7 will modify the data of the entities to reflect the request movement on its next update.

Listing 8 – Game command.

```
# From Human-Input Subsystem:
func human_input():
    var metaGame = get_node("/root/MetaGame")
    if (metaGame.get_node("Player1").has_node("HumanInput")):
        if (Input.is_action_pressed("player1_move_left")):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player1, metaGame.k_PlayerMovementLeft)
        if (Input.is_action_pressed("player1_move_right")):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player1, metaGame.k_PlayerMovementRight)

    if (metaGame.get_node("Player2").has_node("HumanInput")):
        if (Input.is_action_pressed("player2_move_left")):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player2, metaGame.k_PlayerMovementLeft)
        if (Input.is_action_pressed("player2_move_right")):
```

[29] As Pong® is a simple game, the description could separate each subsystems implementation in order – logic, input, and output. In real world scenarios, this would not be the case. However, this is not an issue: provided each subsystems respects their definitions, the separation is ensured.

```
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player2, metaGame.k_PlayerMovementRight)

# From AI-Input Subsystem:
func ai_input():
    var metaGame = get_node("/root/MetaGame")
    var ballPosition = metaGame.get_node("Ball/Transformable").get_pos()

    if (metaGame.get_node("Player1").has_node("AIInput")):
        var player1Size =
        ↪  Vector2(metaGame.get_node("Player1/Collidable").m_Width,
        ↪  metaGame.get_node("Player1/Collidable").m_Height)
        var player1Position =
        ↪  metaGame.get_node("Player1/Transformable").get_pos()

        var player1MoveDirection = Vector2(0.0, 0.0)
        if (metaGame.m_BallDirection.y < 0.0):
            player1MoveDirection = player1Position - ballPosition
        if (player1MoveDirection.x > 0.0):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player1, metaGame.k_PlayerMovementLeft)
        elif (player1MoveDirection.x < 0.0):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player1, metaGame.k_PlayerMovementRight)

    if (metaGame.get_node("Player2").has_node("AIInput")):
        var player2Size =
        ↪  Vector2(metaGame.get_node("Player2/Collidable").m_Width,
        ↪  metaGame.get_node("Player2/Collidable").m_Height)
        var player2Position =
        ↪  metaGame.get_node("Player2/Transformable").get_pos()

        var player2MoveDirection = Vector2(0.0, 0.0)
        if (metaGame.m_BallDirection.y > 0.0):
            player2MoveDirection = player2Position - ballPosition
        if (player2MoveDirection.x > 0.0):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player2, metaGame.k_PlayerMovementLeft)
        elif (player2MoveDirection.x < 0.0):
            g_MetaGameEvents.emit_signal("EventMovePaddle",
            ↪  metaGame.k_Player2, metaGame.k_PlayerMovementRight)


# From the Meta-Game Logic:
func on_event_move_paddle(playerIndex, directionIndex):
    if (playerIndex == k_Player1):
        if (directionIndex == k_PlayerMovementLeft):
            m_bPlayer1MoveLeft = true
        else:
            m_bPlayer1MoveRight = true
    elif (playerIndex == k_Player2):
        if (directionIndex == k_PlayerMovementLeft):
            m_bPlayer2MoveLeft = true
        else:
            m_bPlayer2MoveRight = true


# From the Meta-Game Logic:
func _process(delta):
    # Process game commands.
    if (m_bPlayer1MoveLeft):
        get_node("Player1/Transformable").translate(delta *
        ↪  -k_PaddleSpeed)
        g_MetaGameEvents.emit_signal("EventPaddleMoved", k_Player1,
        ↪  k_PlayerMovementLeft)
    if (m_bPlayer1MoveRight):
        get_node("Player1/Transformable").translate(delta *
        ↪  k_PaddleSpeed)
        g_MetaGameEvents.emit_signal("EventPaddleMoved", k_Player1,
        ↪  k_PlayerMovementRight)
    if (m_bPlayer2MoveLeft):
        get_node("Player2/Transformable").translate(delta *
        ↪  -k_PaddleSpeed)
        g_MetaGameEvents.emit_signal("EventPaddleMoved", k_Player2,
        ↪  k_PlayerMovementLeft)
    if (m_bPlayer2MoveRight):
        get_node("Player2/Transformable").translate(delta *
        ↪  k_PaddleSpeed)
        g_MetaGameEvents.emit_signal("EventPaddleMoved", k_Player2,
        ↪  k_PlayerMovementRight)
    m_bPlayer1MoveLeft = false
    m_bPlayer1MoveRight = false
    m_bPlayer2MoveLeft = false
    m_bPlayer2MoveRight = false

    # Logic subsystem continues here...
```

Note – Whenever an input subsystem detects an input (either from a Human- or an AI-agent), it triggers a game command.

For output, the engine uses the sprite stored in the `DrawableComponent` alongside its world position stored in the `TransformableComponent` to draw each entity to the screen. Logic-wise, this does not change any game data: it only conveys a representation of the current internal data of the game world in sensory ways.

Adding IO completes the first version of the game. The seventh step is extracting the Meta-Game from the created Game. As logic implementation does not process any input or output (it only reacts to game commands), obtaining the Meta-Game from the current implementation is straightforward: the only requirement is striping IO components and event handlers from the Game. Thus, resulting abstract entities are[30]:

- A paddle is a set of {`TransformableComponent`, `CollidableComponent`, `KinematicComponent`, `AIInputComponent`}[31].
- A ball is a set of {`TransformableComponent`, `CollidableComponent`, `KinematicComponent`}.
- The field is a set of {`TransformableComponent`, `CollidableComponent`}.

Data-Driven Customization: Interaction Profiles

If we explore Factory-related patterns together with the DDA, stripping IO from the Game becomes a trivial operation. Developers would be able to perform this operation without code. This is one benefit of using an Interaction Profile: as it describes necessary IO elements to aggregate into the game, changing IO elements implies in

[30] At this point, there are two options for paddles controlled by humans: leave it without an input component (it does not play anymore), or create and attach an `AIInputComponent` which an input subsystem handles to generate `EventMovePaddle` commands. At this moment, instead of discussing the AI counterpart, let us move to new iterations of development to enable new audiences to play. The reason for this is that we could use a variation of the `AIInputComponent` as an aid to enable people with motor impairments to play (Section B.10.5).

[31] The `AIInputComponent` allows an AI-agent to "play" the Meta-Game – an "AI-interactive" game. Omitting it would result into a non-interactive simulation.

adding/removing entries from the Profile. Listing 9 (Figure 26c) and Listing 10 (Figure 26a) show this possibility: the differences between them refer exclusively to IO components, event handlers, and subsystems[32]. Combinations of these elements change how one interacts with a Game, both for input and for output. Listing 11 applies or removes the specializations defined in an Interaction Profile to/from the Meta-Game.

Listing 9 – Interaction Profile for the usual version of the Game.

```json
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
```

[32] In this section, Interaction Profiles use JSON instead of XML (Listing 1), as Godot provides high-level functionalities to parse JSON into its dictionary data structures. For a more concise resource, the Profiles used in this section only contains fields used to tailor the Meta-Game in the prototype.

```
                    ],
                },
                "InputSubsystemSpecialization": {
                    "human_input":
                    ↪  "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "graphical_output": "res://scenes/game/subsystems/gr⌋
                    ↪  aphical_output.tscn"
                }
            }
        }
    }
}
```

Note – The regular version of the game is the first product of the implementation. This Profile allows two human users to play the Game.

Listing 10 – Interaction Profile for the Meta-Game.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                },
                "EventSpecialization": {
                },
                "ComponentSpecialization": {
                },
                "InputSubsystemSpecialization": {
                },
                "OutputSubsystemSpecialization": {
                }
            }
        }
    }
}
```

Note – Assuming the use of a Factory pattern to aggregate IO elements to the Meta-Game, obtaining the Meta-Game is simply removing these components from the resulting Game. With this Profile, although the simulation is still running, human users are unable to perceive it (as there is no output) and to interact with it (as there is no input).

Listing 11 – High-level tailoring algorithm.

```
func tailor_to_profile(interactionProfile, sceneName):
    if (m_bTailored):
        untailor_to_profile(interactionProfile, sceneName)

    # Input mapping.
    tailor_input_mapping(interactionProfile, sceneName)

    # Register event handlers.
    tailor_event_handlers(interactionProfile, sceneName)

    # Tailor existing entities.
    tailor_entities(interactionProfile, sceneName)

    # Add input and output subsystems.
    tailor_input_subsystems(interactionProfile, sceneName)
    tailor_output_subsystems(interactionProfile, sceneName)

    m_bTailored = true


func untailor_to_profile(interactionProfile, sceneName):
    if (not m_bTailored):
        return

    # Remove input mapping.
    untailor_input_mapping(interactionProfile, sceneName)

    # Remove event handlers.
    untailor_event_handlers(interactionProfile, sceneName)

    # Untailor existing entities.
    untailor_entities(interactionProfile, sceneName)

    # Remove input and output subsystems.
    untailor_input_subsystems(interactionProfile, sceneName)
    untailor_output_subsystems(interactionProfile, sceneName)

    m_bTailored = false
```

Note – This algorithm applies or remove the Interaction Profile tailoring to/from the Meta-Game, converting to a Game (or back to a Meta-Game).

B.10.2 Second Iteration: Visual Impairments (Low Vision)

For simplicity, the remainder of this section assumes that the logic implementation does not change any longer, unless stated otherwise[33]. This way, the following subsections will include accessible functionalities for different interaction needs, based on high and low-level strategies defined by Yuan, Folmer & Harris (2011) as possible modifications. For brevity, we omit discussions about the strategies themselves, as our focus is showing how this approach of this chapter enables adding/removing IO to tailor the application.

Yuan, Folmer & Harris (2011) cite enhancing the output stimuli to aid users with low vision. For this, low-level strategies include accessible color schemes, zoom options, and font size – all output-related functionalities. Developers should, therefore, define new output elements: that is, they should create new (or improve existing) output components, event handlers, and subsystems to improve the solution.

As new functionalities can be presented continuously, one approach would be to extend image and text components with a scale value and the new font properties to their existing data. This would redefine the following:

`DrawableComponent` stores a sprite (2D image), with a scaling value, to draw.

`WritableComponent` stores a string to write, alongside with a font typeface, and a size.

`DrawingSubsystem` draws entities with `DrawableComponents` to the screen. It reads position data from a `TransformableComponent` and graphical data from a `DrawableComponent` to draw a scaled image to the screen.

`TextSubsystem` draws entities with `WritableComponents` to the screen. It reads the text from the string stored in the component, and draws it to the screen with the size and typeface stored in the component.

[33] In other words, this assumes the Meta-Game was fully implemented in the first version. It would not be a problem if it changed, however. In this case, it would be necessary to add new functionalities and update the Interaction Profile with those inclusions, promoting iterative development.

For the remainder of the implementation, it is possible to fully reuse existing logic and input functionalities of G^1, and, possibly, some output functionalities (at least partially). After a second iteration, this would result in:

$$G^2 = \{C^2_{Game}, V^2_{Game}, R^2_{Game}, S^2_{Game}, A^2_{Game}, E^2_{Game}, H^2_{Game}, P^2\} \tag{B.34}$$

With:

$$\begin{gathered}
C^{2^L}_{Game} = C^{1^L}_{Game} = C_{Meta}, \\
C^{2^I}_{Game} = C^{1^I}_{Game} \\
C^{2^O}_{Game} \supseteq C^{1^O}_{Game}, \\
V^2_{Game} = V^1_{Game} = V_{Meta} \\
R^2_{Game} = R^1_{Game} = R_{Meta} \\
S^{2^L}_{Game} = S^{1^L}_{Game} = S^L_{Meta}, \\
S^{2^I}_{Game} = S^{1^I}_{Game} \\
S^{2^O}_{Game} \supseteq S^{1^O}_{Game}, \\
A^2_{Game} = A^1_{Game} = A_{Meta} \\
E^2_{Game} \supseteq E^1_{Game}, \quad E^2_{Game} \cap E^1_{Game} = E_{Meta} \\
H^2_{Game} \supseteq H^1_{Game} \\
P^2 \neq P^1
\end{gathered} \tag{B.35}$$

It is possible to make the same comparisons for any new Profile. It is important to note that the logical implementation does not change. Developers create and implement new accessibility functionalities for game interaction, adding them to the system, and updating Interaction Profiles to apply new IO to the Meta-Game. For this first case, We could achieve an accessible version of the game for users with low vision. However, improvements are not restricted to them. Previously included users may modify the original Profile to benefit from scaling (for instance, better display in higher resolution screens) and text customization (size and typeface) improvements. After implementing new IO functionalities, adding them to an Interaction Profile is enough to merge them into a Game[34].

The Second Iteration of Ping-Pong: Adding Zoom

Zoom options are a possible example of functionality for enhancing output stimuli. In game programming, one approach to implement zooming is using a camera. Godot offers a camera node which can be aggregated to an entity, acting as component.

[34] That is, assuming they are not incompatible with each other.

We explore it in this example to provide a "Zooming Output Subsystem" (Listing 12), including it into the existing Profile from Listing 9 to aggregate to it basic scaling (Listing 13, resulting into Figure 26d).

Listing 12 – Zooming Output Subsystem.

```
func _process(delta):
    if (Input.is_action_pressed("zoom_in") or
    ↪  Input.is_action_pressed("zoom_out")):
        var camera = self
        var zoom = get_zoom()
        if (Input.is_action_pressed("zoom_out")):
            zoom += Vector2(0.25, 0.25)
        if (Input.is_action_pressed("zoom_in")):
            zoom -= Vector2(0.25, 0.25)
            if (zoom.x <= 0.0):
                zoom.x = 0.25
                zoom.y = 0.25
        set_zoom(zoom)

        var metaGame = get_node("/root/MetaGame")
        var fieldPosition =
        ↪  metaGame.get_node("Field/Transformable").get_pos()
        var fieldSize = metaGame.get_node("Field/Collidable")
        set_pos(fieldPosition + 0.5 * Vector2(fieldSize.m_Width,
        ↪  fieldSize.m_Height))
```

Note – Using a camera is an easy way to implement graphical stimuli enhancing to games. As every output subsystem, this listing does not modify data from logic elements – it only reads from them to acquire relevant data to achieve its goal. In this case, it uses the position and size of the field to center the camera into the center of the field.

Listing 13 – Interaction Profile with graphical enhancement.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                },
                "InputSubsystemSpecialization": {
                    "human_input": "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn",
                    "camera_output": "res://scenes/game/subsystems/camera_output.tscn"
                }
            }
        }
    }
}
```

Note – Comparing this Profile to Listing 9, the only difference was the inclusion of the Zooming Output Subsystem, by means of instancing the "`camera_output`".

The functionality could be refined, for instance, including multiple zoomed view ports of the Game (Figure 26e).

B.10.3 Third Iteration: Visual Impairments (Blindness)

To improve the game for blind users, Yuan, Folmer & Harris (2011) highlight strategies focusing on stimuli replacement. The current application only supports basic visual stimuli. To further improve it, we could create audio stimuli by defining new output components and subsystems aiming for stimuli replacement. The overall strategy is similar to the one outlined in the previous subsection. Component-wise, developers could, for instance, add cue to the `AudibleComponent`, improve sonification in existing output subsystems, or define multiple subsystems, exploring different technologies or approaches:

`AudibleComponent` stores a stereo sound for binaural playback.

`BinauralAudioSubsystem` playbacks entities with `AudibleComponents`, exploring binaural sounds for depth and position. It fetches sounds to reproduce from the `AudibleComponent` and the position from the `TransformableComponent`.

`HRTFAudioSubsystem` playbacks entities with `AudibleComponents`, exploring binaural sounds for depth and position. It fetches sounds to reproduce from the `AudibleComponent` and the position from the `TransformableComponent`; it calculates head-related transfer functions (HRTF) from the `TransformableComponent` of the human-agent to provide spatial positioning.

Event-wise, however, they could define new output event handlers as well. As there are already events in the game-logic, developers would implement new handlers to provide audio cues to suitable occasions, registering them for events corresponding to the occasions. The same approach could be explored for haptic cues, to provide haptic/tactile feedback.

When defining new handlers, developers may notice important events they had not considered before: in this case, they could define new events, and create/register event handlers for them. These new events are not important to the Meta-Game: they are related to human interaction with Games. Thus, the game logic may dispatch the events to enable interaction improvements – that is, these events are provided to allow IO event handlers to improve the feedback and overall aesthetics of the Game. Once again, this may benefit other users: perhaps with better sound playback and quality, perhaps with better audio cues or haptic feedback.

The Third Iteration of Ping-Pong: Adding Sound Effects

To illustrate a new approach, this section adds sound effects as output event handlers (Listing 14). The goal is twofold – providing flexibility to enable/disable sound effects to play for events using the Interaction Profile (Listing 15), and to show an output subsystem may register output event handlers to itself (Listing 16).

Listing 14 – Event handlers providing sound effects.

```
func on_event_ball_released_audio():
    g_SceneManager.get_user_interface_overlay().play_sound_effect(k_Start)


func on_event_goal_audio(playerIndex):
    g_SceneManager.get_user_interface_overlay().play_sound_effect(k_Goal)


func on_event_ball_hit_wall_audio(wallIndex, position):
    g_SceneManager.get_user_interface_overlay().play_sound_effect(k_PingPongWall)


func on_event_ball_hit_paddle_audio(playerIndex, position):
    g_SceneManager.get_user_interface_overlay().play_sound_effect(k_PingPongPaddle)


func on_event_ball_moved_audio(directionVector):
    g_SceneManager.get_user_interface_overlay().play_sound_effect(k_Ball)
```

Note – For simplicity, these handlers play a sound when triggered by an event.

Listing 15 – Interaction Profile with graphical enhancement.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                    "EventBallReleased": [
                        "on_event_ball_released_audio"
                    ],
                    "EventGoal": [
                        "on_event_goal_audio"
                    ],
                    "EventBallHitWall": [
                        "on_event_ball_hit_wall_audio"
                    ],
                    "EventBallHitPaddle": [
                        "on_event_ball_hit_paddle_audio"
                    ],
                    "EventBallMoved": [
                        "on_event_ball_moved_audio"
                    ],
                    "EventPaddleMoved": [
                    ]
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/Meta-Game/components/ai_input_component.tscn"
                    ],
                },
                "InputSubsystemSpecialization": {
                    "ai_input": "res://scenes/Meta-Game/subsystems/ai_input.tscn",
                    "human_input": "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn"
                }
            }
        }
    }
}
```

Note – Starting from the Profile of Listing 9, this listing adds the name of desired event handlers to play sound effects when an event happens. For instance, when the Meta-Game triggers the event `EventBallReleased`, the game will dispatch the event handler `on_event_ball_released_audio`.

Listing 16 – Subsystem for sound effects.

```
func _ready():
    # Register event handlers.
    g_MetaGameEvents.connect("EventBallReleased", g_EventHandlers, "on_event_ball_released_audio")
    g_MetaGameEvents.connect("EventGoal", g_EventHandlers, "on_event_goal_audio")
    g_MetaGameEvents.connect("EventBallHitWall", g_EventHandlers, "on_event_ball_hit_wall_audio")
    g_MetaGameEvents.connect("EventBallHitPaddle", g_EventHandlers, "on_event_ball_hit_paddle_audio")
    g_MetaGameEvents.connect("EventBallMoved", g_EventHandlers, "on_event_ball_moved_audio")
```

Note – This output subsystem registers event handlers to play sound effects when an event happens. Registering it to an Interaction Profile would be similar to adding zoom in Listing 13.

B.10.4 Fourth Iteration: Hearing Impairment

For hearing impairment, Yuan, Folmer & Harris (2011) comment that strategies involve stimuli replacement. This time, however, one possibility is converting audio into visual media. We may improve the text component and create a derivative `CaptionableComponent` with a `SubtitleSystem`. Another approach include exploring event handlers to create onomatopoeia, or graphical effects and animations.

Once again, benefits are not restricted to hearing impaired users. For instance, subtitles may aid foreign users to play, if there is no option for their native language. Graphical effects, on the other hand, may improve the playing experience to sighted users, providing further feedback to the Game.

The Fourth Iteration of Ping-Pong: Adding Graphical Effects

As, currently, Ping-Pong does not have voice-over for subtitles, this section adds graphical and animations to convert sound effects into graphical stimuli. Graphical effects implementation resembles sound effects as both provide instantly feedback for when something important happens. Thus, as discussed in Subsection B.9.2, creating output event handlers work well for them. For brevity, we omit the implementation of the handlers and have added them to an output subsystem (“`graphical_output`”), showing only changes in the new Interaction Profile (Listing 17, resulting into (Figure 26f)). As with sound effects, for more granularity, event handlers for graphical effects could be globally available to allow an Interaction Profile to enable/disable it individually.

Listing 17 – Interaction Profile with graphical effects.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/Meta-Game/components/human_input_component.tscn"
                    ],
                },
                "InputSubsystemSpecialization": {
                    "human_input": "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn",
                    "graphical_effect_output": "res://scenes/game/subsystems/graphical_effect_output.tscn"
                }
            }
        }
    }
}
```

Note – Starting from the Profile of Listing 9, this listing adds graphical effects to specialize the Meta-Game using a subsystem. To combine graphical effects with sound effects, we could add event handlers (or the subsystem) from Listing 15 to create a new Profile with both sound and graphical effects.

B.10.5 Fifth Iteration: Motor Impairment

When we consider motor impairment, the focus of accessibility functionalities change from output to input. Yuan, Folmer & Harris (2011) provide two high-level strategies for motor impairments: input reduction and input replacement. Input replacement requires the creation of new input components, event handlers, and subsystems.

Input reduction, on the other hand, is greatly simplified from the choice of using events to implement semantic game commands. To automate parts of the interaction, developers define input components and subsystems to send game commands on behalf of the user. This effectively implements input reduction – AI-agents perform part of what the human-agent would to aid her/him. This can further help users who use assistive technologies – the AI-agents may become like the user's co-pilot.

Developers may go further, offering input queues to allow users to create their own input macros. With this approach, a macro can be a queue of game commands, with a specific time in between triggering them.

Lastly, it is possible to simulate the scanning technique with a combination of components, events and event handlers. We could create `OrdenableComponent` to assign order to components. `Next`, `Previous`, `Select` events allow cycling between adjacent entities, and to select the desired one. Cycling can be manual or automated; the latter uses the same idea of the AI-agent: a subsystem send a `Next` or `Previous` event at regular time intervals, allowing the Game to cycle the options.

The Fifth Iteration of Ping-Pong: Automating User Input

Every listing up to this point defined Ping-Pong Games with two human users instead of one human-agent and one AI-agent (or even two AI-agents). Input automation address this shortcoming, provide both a non-human opponent and an accessibility functionality for motor impairment. Listing 8 provided the implementation for the core of the AI-Input Subsystem in function `ai_input()`. However, as all agents use game commands to provide input to the Meta-Game, human-agents may also benefit from it to acquire full or partial input automation. Listing 18 illustrates an Interaction Profile for both cases into a single one. "`Player1`" is a human-agent, meaning a human user control the paddle. "`Player2`", in the other hand, has both an `HumanInputComponent` and an `AIInputComponent`, meaning that a human-agent and an AI-agent are able to control it[35].

Listing 18 – Interaction Profile with graphical effects.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
```

[35] For an AI-Agent, we could remove the `HumanInputComponent` from "`Player2`".

```
            "player1_move_left": ["Keyboard", "A"],
            "player1_move_right": ["Keyboard", "D"],
            "player2_move_left": ["Keyboard", "LEFT"],
            "player2_move_right": ["Keyboard", "RIGHT"]
        },
        "EventSpecialization": {
        },
        "ComponentSpecialization": {
            "Field": [
                "res://scenes/game/entities/drawable_field.tscn"
            ],
            "Ball": [
                "res://scenes/game/entities/drawable_ball.tscn"
            ],
            "Player1": [
                "res://scenes/game/entities/drawable_player1.tscn",
                "res://scenes/game/components/human_input_component.tscn"
            ],
            "Player2": [
                "res://scenes/game/entities/drawable_player2.tscn",
                "res://scenes/Meta-Game/components/ai_input_component.tscn",
                "res://scenes/game/components/human_input_component.tscn"
            ],
        },
        "InputSubsystemSpecialization": {
            "ai_input": "res://scenes/Meta-Game/subsystems/ai_input.tscn",
            "human_input": "res://scenes/game/subsystems/human_input.tscn"
        },
        "OutputSubsystemSpecialization": {
            "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn"
        }
```

```
                    }
                }
            }
        }
```

Note – Input reduction with the approach explores game commands and AI-agents to partially or fully automate human interaction with the Meta-Game.

Due to the current implementation from Listing 8, the AI-agent is able to fully automate interaction of the user. For Ping-Pong, the one option for input reduction would be letting the AI-agent move the paddle into a single direction (left, for instance), leaving the user with the other one (right, for the previous case). A second option would be using a timer, letting the AI-agent provide input for the user only after a pre-defined time without interaction had elapsed.

B.10.6 Cognitive Impairment: Parameterizing the Meta-Game

At this point, the remaining high-level strategies from Yuan, Folmer & Harris (2011) regard cognitive impairments. Input reduction is the same strategy aimed at motor impairments. Stimuli reduction, and time constraints' reduction, on the other hand, are more related to game design (as they relate to game logic). It is possible to partially satisfy them with parameterization; in this case, it could be useful to include values of the parameters in the Interaction Profile. This could lead to derived simpler versions of the Meta-Game.

However, modifying the logic would modify the Meta-Game directly, which would go against the goal of this chapter (tailor IO) and could cause confusion. Therefore, we avoid it here[36].

B.10.7 Sixth Iteration: A Simple Head-Up Display

Although the previous iterations all focused on how to convey game information and provide interaction to entities, subsystems may have important data that developers might want to provide to users. For graphical games, one popular approach is using head-up displays. For aural games, an alternative is providing an "on request" auditory interface – use a game command to request speech information.

In this chapter, we provide an example of graphical head-up display using an

[36] An alternative to address this situation would be using the elapsed time as a *parameter* to the Meta-Game via a command. In this way, we would be able to arbitrarily slow down and/or pause/resume the Meta-Game simulation.

output subsystem in Listing 19[37]. As with any IO subsystem, an Interaction Profile may add the head-up display into a Game (Listing 20, which results into Figure 26g).

Listing 19 – Simple Head-Up Display.

```
# Time elapsed since this interface was created.
var m_TimeElapsed = null


func _ready():
    set_process(true)

    on_event_goal_score_text_output(-1)

    # Register event handlers.
    g_MetaGameEvents.connect("EventGoal", self,
    ↪  "on_event_goal_score_text_output")


func _process(delta):
    m_TimeElapsed += delta
    update_score()


func update_score():
    var metaGame = get_node("/root/MetaGame")
    get_node("RoundTime").set_text("Round time: %.02fs" % m_TimeElapsed)


func on_event_goal_score_text_output(playerIndex):
    var metaGame = get_node("/root/MetaGame")
    get_node("Player1Score").set_text("Player 1: %d" %
    ↪  metaGame.m_Player1Score)
    get_node("Player2Score").set_text("Player 2: %d" %
    ↪  metaGame.m_Player2Score)

    m_TimeElapsed = 0.0
```

[37] The prototype has a second one, proving text description for the last event of interest that happened in the game. It is called "`text_output`", and appears in the Profile of Listing 21.

Note – An output subsystem to display the score stored in the Meta-Game into the screen, and the time elapsed since the head-up display subsystem was created.

Listing 20 – Usual version of the game with head-up display.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                },
                "InputSubsystemSpecialization": {
                    "human_input": "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn",
                    "score_text_output": "res://scenes/game/subsystems/score_text_output.tscn"
                }
            }
        }
    }
}
```

Note – Creating (non-gaming) user interfaces for the Meta-Game follows the same principles of the general approach.

B.10.8 Seventh Iteration: Combining All Profiles into One

IO specialization do not modify the Meta-Game. The logic implementation of the game is not affected by whom is interacting with it, nor for whom the output is conveying stimuli to represent it. Thus, we can combine IO specializations as “unit functionalities” to improve the accessibility of the game. Listing 21 provides an example grouping all specializations from the previous subsections into a single Interaction Profile (illustrated in Figure 26h). The resulting Games, thus, have sound and graphical effects, zooming, input reduction, and game information from the head-up display.

Listing 21 – A game with sound and graphical effects, zoom, input reduction, and a head-up display.

```
{
    "InteractionProfile": {
        "Scenes": {
            "PingPong": {
                "InputMapping": {
                    "player1_move_left": ["Keyboard", "A"],
                    "player1_move_right": ["Keyboard", "D"],
                    "player2_move_left": ["Keyboard", "LEFT"],
                    "player2_move_right": ["Keyboard", "RIGHT"]
                },
                "EventSpecialization": {
                    "EventBallReleased": [
                        "on_event_ball_released_audio"
                    ],
                    "EventGoal": [
                        "on_event_goal_audio"
                    ],
                    "EventBallHitWall": [
                        "on_event_ball_hit_wall_audio"
                    ],
                    "EventBallHitPaddle": [
                        "on_event_ball_hit_paddle_audio"
                    ],
                    "EventBallMoved": [
                        "on_event_ball_moved_audio"
                    ],
                    "EventPaddleMoved": [
                        "on_event_paddle_moved_audio"
                    ]
                },
                "ComponentSpecialization": {
                    "Field": [
                        "res://scenes/game/entities/drawable_field.tscn"
                    ],
                    "Ball": [
                        "res://scenes/game/entities/drawable_ball.tscn"
                    ],
                    "Player1": [
                        "res://scenes/game/entities/drawable_player1.tscn",
                        "res://scenes/Meta-Game/components/ai_input_component.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
                    ],
                    "Player2": [
                        "res://scenes/game/entities/drawable_player2.tscn",
                        "res://scenes/Meta-Game/components/ai_input_component.tscn",
                        "res://scenes/game/components/human_input_component.tscn"
```

```
                    ],
                },
                "InputSubsystemSpecialization": {
                    "ai_input": "res://scenes/Meta-Game/subsystems/ai_input.tscn",
                    "human_input": "res://scenes/game/subsystems/human_input.tscn"
                },
                "OutputSubsystemSpecialization": {
                    "text_output": "res://scenes/game/subsystems/text_output.tscn",
                    "audio_output": "res://scenes/game/subsystems/audio_output.tscn",
                    "graphical_output": "res://scenes/game/subsystems/graphical_output.tscn",
                    "graphical_effect_output": "res://scenes/game/subsystems/graphical_effect_output.tscn",
                    "camera_output": "res://scenes/game/subsystems/camera_output.tscn",
                    "score_text_output": "res://scenes/game/subsystems/score_text_output.tscn"
                }
            }
        }
    }
}
```

Note – This Profile combines all specialization into a single one. To create other possible Games, one could remove elements from it.

Although, in this case, all functionalities are compatible, some specializations might be inherently incompatible with others. For instance, a specialization which adds music to the game may hinder spatial audio for visual impairment. However, the possibility to mix and match specialization exists, and users may benefit from combinations. Extending specializations, thus, may, indirectly, allow combinations to improve the accessibility for more users.

B.10.9 Further Iterations: User-Level Accessibility?

A new profile can combine any existing IO components and event handlers defined so far. Accessibility improvements aimed at a specific public tend to benefit users without the disabilities of the original public as well. This chapter approach allows one to combine any existing element and add it to her/his Interaction Profile. It makes it possible to combine components and event handlers, mixing-and-matching combinations to define new interaction profiles.

Assuming that there are enough components and event handlers implemented, can this lead to the construction of an accessible game at the user-level? This is what the approach aims towards.

B.11 Discussion and Future Work

Although digital systems can be flexible, implementation decisions might impose predetermined interaction to users. This may work well for some users, but not for everyone. Everyone is unique. Abilities, capabilities, knowledge, culture, beliefs do vary according to individuals. These differences reflect on expectations and needs of a person. This is valid for all aspects of life – digital system interaction included.

Practices such as the Universal Design and tailoring offer a different approach, trying to include the user and allowing her/him to have a more suitable interaction according to her/his user needs. The system should adapt itself to suit users needs, not force the other way around. The implementation should, therefore, allow tailoring modifications to improve the use experience.

This chapter presented an approach to enable run-time tailorability in digital games. We called the result as tailorable games. The approach combines a few elements (entities, components, events, and event handlers) using set operations to decouple the logical implementation of the game from its interaction. The decoupled implementation results in a system without user interaction (here named meta-system – or Meta-Game, in the game domain), on which we can include different input and output interactions by adding IO-specific elements at run-time. This allows developers to conceive different interaction models to the Game, effectively enabling the creation of multiple schemes for physical-level interaction.

One can combine existing input and output elements to define her/his own way of interacting with the system, by creating an Interaction Profile to convert a user-free Meta-Game into a (potentially) user-accessible Game. If the desired interactions does not already exist in the system, developers can create them. As long as it is possible to convert a new functionality into entities, components, events, and event handlers, it should be possible to include the functionality on the game at run-time.

Other than decoupling the implementation and providing run-time flexibility, the described approach has the benefit of promoting software reuse. We may reuse existing elements to create new functionalities, or define news ones to extend the original possibilities. The result is a mutualistic development process, in which improvements may benefit a wider number of users than originally conceived. By creating accessibility functionalities, we improve the system for new and old users alike. Every new interaction alternative provides better accessibility and variety and choice for physical level interactions are functionalities that may enhance the quality for the human interaction.

It is important to note that the presented approach is based on set theory. As such, we can implement the approach using any programming language which can implement a set data structure (or a data structure equivalent to sets to enable run-time manipulation) and has file IO (for the DDA). As we can implement entities, components, events, and event handlers on software, the approach has low requirements for implementation. For the prototype that motivated this work – a game engine –, a C++ implementation was discussed in (GARCIA, 2014), and is available at <https://github.com/francogarcia/uge>. Two newer prototypes, created with the open-source Godot engine, are available at <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong> and at <https://gitlab.com/francogarcia/GamesRunTimeTailorability>. These prototypes do not change the engine. Rather, they implement the approach described in this chapter in implementation of the games itself, performing the proposed operations during run-time.

Although the approach used games as an example, it could be extended to other interactive systems, provided that the implementation followed the same principles. Games are a class of interactive systems.

Once again, the mutualistic improvements raise a question. Assuming that there are enough components and event handlers implemented, can this lead to the construction of an accessible game at the user-level? If it does, the approach described approach in this chapter provides a step towards the implementation of universally-accessible digital systems.

It is important to note, however, that the approach has its drawbacks. Computing always have trade-offs. There are four main issues. First, there is a trade-off between raw performance versus run-time flexibility. The implementation is dynamic, requiring memory allocations and manipulation of data structures at run-time. For instance, entities could have static definitions in conventional systems, allowing for compile time optimization and, possibly, exploring better memory model. With the approach, they are created during run-time. A user interacts with a running system, not with the code. There are possibilities around this problem, although they are not in the scope of this chapter[38].

[38] One example is loading compiled dynamic link libraries (DLLs) during run-time, avoiding scripting languages and the use of slower strategies (such as polymorphism and reflection). This re-

Second, the approach does require deep knowledge of the underlying architectures and programming concepts. The proposed definitions and theorems explore fundamentals of the concepts to improve run-time flexibility and customizability[39]. The theoretical work and algorithms presented here aims to avoid ambiguities and reduce required knowledge and programming expertise. Even though, working with undergraduate students, we found they are not used to think of systems in terms of the elements of this chapter, which required time and willingness to learn.

Third, the proposed interaction (IO) separation from logic is very systematic and rigid. The separation makes prototyping slower, as the programmer must consciously avoid mixing these functionalities. As components, event handlers, and subsystems process different functionalities (potentially cross-cutting concerns), developers have to be careful, assuring the correct order for the operations. Besides, although it is possible to refactor an implementation to achieve it, it requires significant efforts for non-trivial applications. The result for this trade-off is always having to choose among simple code, optimizations, and convenience. Prototyping game mechanics and new ideas, in special, benefit from sloppy code, which the approach discourages. Specialized tools may aid in this regard (and we are working on one, mentioned at the end of this section). Functionalities for refactoring and extracting elements are also helpful. The scene model explored for Godot, for instance, allows developers to extract new components and entities with ease.

Fourth, the creation of input and output functionalities require design and implementation efforts. This chapter has not discussed design issues, as it focused on the implementation. The prototype was also more focused on tailoring the IO than providing truly accessible game experiences. Designing accessible systems is challenging – the existing Literature is proof. However, provided that we can design an accessible interaction and decompose it into the elements of this approach, we will be able to tailor this functionality to the Meta-Game, as we illustrated with the prototypes. The approach, thus, considers universal access as an iterative, incremental, and mutualistic approach, including more people when new elements are created or improved. In a way, it resembles set theory. At first, there is the Meta-Game, which is inaccessible for all. Considering an empty set of possible users, however, it is universal. Next, there is the conventional game, which is accessible to users belonging to a normal distribution of abilities. For this group and for no one, the game is universal. Then, for each new accessible version, a new public can be introduced to the game. Following this reasoning,

duces the performance penalty to the first library load. For more information on this topic, <http://runtimecompiledcplusplus.blogspot.com.br/>, <https://github.com/RuntimeCompiledCPlusPlus/RuntimeCompiledCPlusPlus>. A second example is exploring Data-Oriented Design within an ECS implementation.

[39] An instance is the use of event handlers themselves, which are registered in a data-driven way. We provide a self-contained example in <https://github.com/francogarcia/Data-Driven-Event-Handlers>.

possibilities for specialization become gradually more accessible.

It is, however, only an initial step towards a greater goal. There are interesting alternatives in to improve and extend the possibilities. For one example, we are working on combining the presented approach with end-user development practices. The goal is to allow users with different (dis)abilities to create and modify games themselves, improving them to better suit their needs and of their family and friends. To support this goal, a (meta-)game creation tool is under development as a meta-system, following the same principles of Meta-Games and Games to provide different IO specializations for game creation. The creation meta-system provides abstract functionalities to create game, offering specializations to enable people with different abilities to contribute on game creation. Instead of games, the tool create Meta-Games and aid specializing them to Games. This enables people with different (dis)abilities to create games for people with, potentially, different (dis)abilities. Therefore, it changes creation from what people are unable to do to what they are able to do: individuals are able to use their knowledge and abilities to enable more people to play. Combined with a text-based Interaction Profile and with a sharing network, we intend to allow users to create, improve, and share their modifications to enable more people to interact with their favorite games. We have selected two public as case studies, which should start this year and provide feedback to improve this approach.

APPENDIX C

Real-Time Tailorable Games: Advancements in Run-Time Tailorability

C.1 Introduction

This appendix complement the Appendix B, with our latest findings regarding run-time tailorability.

C.2 Run-Time Tailorability is Turing Complete

If we implement run-time tailorability into a Turing Complete (SIPSER, 2005; HOPCROFT; MOTWANI; ULLMAN, 2000) language, it immediately follows that it is Turing Complete. Therefore, provided that we can find a solution and implement it into an algorithm that runs in suitable time (acceptable time for use), we can introduce any computationally possible accessibility feature at run-time – either at specific points (event handlers), or for permanent presentation (components). We can affirm this due to the Church-Turing thesis (SIPSER, 2005), which states that any Turing machine can run any algorithm that any other Turing machine could.

C.3 Full Automation in Real-Time Interaction Is Not Always Possible

For a truly universal game, an AI-agent would have to be able to beat any game alone. This is impossible, as it is not computationally possible to solve all classes of computational complexity theory (SIPSER, 2005; HOPCROFT; MOTWANI; ULLMAN, 2000). with current programming approaches in suitable time for games. For instance, many interesting games are NP-complete(DEMAINE; HOHENBERGER; Liben-Nowell, 2003; CLIFFORD et al., 2012); full automation would not be possible within suitable time limits.

However, in practice, it is not necessary to simulate a human playing games. There are some alternatives to create tailorable games.

C.4 Run-Time Tailorability in Practice

Can we reach user level accessibility, for any user? Currently we cannot, as we do not have the technology to do so.

Could run-time tailorability enable the creation of universal tailorable games? As it is Turing Complete, as we can introduce any computationally possible with run-time tailorability at specific moments, we are able to use any existing assistive technology at the correct moment with events. As we can read any meta-game data, it follows that, provided that we could translate the data to the assistive technology, we could enable use. Therefore, run-time tailorability is a potential candidate for the implementation of universal tailorable games with everything that is currently available – and everything that could be made available in the future. It allows for arbitrary code execution at the exact times that they are needed.

For simple user interfaces – like from conventional applications –, automation is possible. If we can reduce the game to a parameterized application (for instance, a command-line program), we can automate it. At least in this way, we argue that we could step towards universal tailorable games – for as many people as we can with the currently existent technology.

For complex applications, we cannot yet automate solutions. We have found an interesting alternative. The first is considering inclusion both as independence and interdependence, and people both as individuals and communities (BENNETT; BRADY; BRANHAM, 2018; LIU; DING; GU, 2016a).

Instead of a single human playing, we could enable multiple people to play at once. As the input is decoupled from the meta-game, each people could provide a subset of commands, everyone could provide all commands – or any combinations in between.

Then, we can slowly move towards independence, as we start figuring out how to convert core game loops into composable commands. Although an AI-agent may not be able to beat a full game, it can perform smaller tasks within games.

C.5 Run-Time Tailorability Makes Accessibility a Use-Time Problem

If we implement games for no one, we design for everyone. As we do not assume any interaction abilities, we are free to redefine them at use-time. As there only exists data, we are free to define as many ways to represent the data as we want. In the same ways, as commands are the only input, we can define as many ways to provide input to the game as we wish. In particular, we can explore this for accessibility.

We are free to explore assistive technologies (software and hardware) and any other accessibility features without anticipating their use. Rather, we can attach them at run-time. Once we implement software to introduce them into the solution, we are free to recombine them.

The underlying assumption is that software can be re-configurable within the limits of Turing completeness. In this way, we abstract the interaction, and become able to attach it later. If it helps, you may thing it as an add-on/extension/plugin; however, instead of task-related features, it introduces interaction features to the game.

This way, run-time tailorability makes interaction a composable part of digital games. We can redefine everything, because we have access to all inner data. There can co-exist potentially infinite variations of an application that shares the same logic. We are free to define multiple.

Thus, we can use tailoring as a strategy towards universal access. We can explore "one core fits mosts" alongside "accessible" versions. We can define multiple interfaces, multiple ways to interact, and allow the user the build her/his own.

C.6 Run-Time Tailorability Makes Accessibility a Community Problem

As information results from interpretation of data, it is up for designers to figure out how to present them is as many ways as they can, they introduce alternatives to the game. This makes accessibility an opportunity cost for developers – if they follow the approach, they can enable the community to improve accessibility even if they do not. In the same way, they do not have to assume a particular accessibility feature in advance; they can add them over time, as needed.

C.7 Concluding Remarks

With our results from Appendix B, we can implement games in expansive and expressive ways. The only limits to what we can introduce at use-time with run-time tailorability are what we can implement as algorithms that runs in suitable time. If we can find a solution to map data into representation within the limits of what we are able to implement, we can attach it to the solution to enable use.

The Meta-Game to digital games, the Game to the player. The Meta-System to applications, the System to the user.

We cannot assume that every game may become universal, but accessibility can always be improved. If we are correct, we can move towards maximum inclusion to universal access in digital games – as far as technology, programming knowledge, and creativity allows us. Let us move towards tailorable creation of tailorable games. Once people can use and create digital games, everyone is equal.

APPENDIX D

One Small Step From a User, a Giant Leap for Universal Access: A Collaborative Work Model for Co-Creation of Digital Tailorable Games

D.1 Introduction

Information and Communication Technologies (ICTs) are part of modern life. They extend human abilities and capabilities to improve productivity, reduce distances, connect people, and enhance quality of life. However, despite their importance, ICTs are not yet for everyone. ICTs are commonly designed for average users instead of for individuals (POZZI; BAGNARA, 2013). As a result, people who deviate from the intended audience of an ICT may struggle to use it. At best, they may face poor usability. At worst, the ICT might be inaccessible for them.

When we consider inclusion, there are multiple barriers hindering possibilities of use. As a broad term, inclusion may encompass physical, cognitive, and emotional (dis)abilities, age, language, education and knowledge, culture, socioeconomic factors, computer literacy and skills, and even time and context of use. When these factors affect the quality of interaction[1], people may suffer from usability and accessibility issues.

Although there are approaches to design towards universal access (for instance, Inclusive Design, and Universal Design / Design For All (SAVIDIS; STEPHANIDIS,

[1] Even if temporarily, as in situational disabilities (for instance, while driving).

2006; SAVIDIS; STEPHANIDIS, 2004; NERIS; BARANAUSKAS, 2009)), their intended audience are professional developers. End-users are the people who suffer from the limitations of inaccessible systems. Nevertheless, their abilities enable them to overcome everyday difficulties – either alone or assisted (BIGHAM; LADNER; BORODIN, 2011; BRADY; BIGHAM, 2015). What if end-users could contribute to improve accessibility and usability of digital systems?

Collaboration is a key concept in Computer-Supported *Collaboration* Work (CSCW). If we focus on abilities over disabilities, and individuals as well communities, and if we provided computational support to empower end-users, people could collaborate to co-create digital systems.

In this perspective, we could explore abilities of end-users to address accessibility issues from other end-users – universal access become a community problem. Instead of considering people as average users, we should adopt two perspectives: people alone as individuals, and people together as communities. Individuals provide their abilities to the community. From each one, what she/he is able to do. The community provide support for the individuals. For each one, what she/he needs to become able to do. Although recent work (for instance, (BENNETT; BRADY; BRANHAM, 2018; LIU; DING; GU, 2016a)) share this perspective in real world scenarios, they examine real world "hardware" scenarios. Software, however, has a unique feature of being able to redefine its features for human-computer interaction at run-time, and therefore, at use-time (GARCIA, 2014; GARCIA; NERIS, 2014). With this possibility, every time we enable someone new to use and create, she/he can potentially enable someone else to use and create. Therefore, rather than a static process, inclusion becomes a dynamic process – incremental, collaborative, and iterative. Software accessibility comes from within the software itself; instead of adaptations at use-time, people provide interaction alternatives to minimize the need for external adaptation (for instance, assistive technologies).

This approach can be particularly useful for systems on which traditional accessibility solutions are hard. Digital games are in this situation; they are complicated applications for accessibility and universal access, due to their real-time interactive constraints, poor compatibility with assistive technologies, and costs and efforts for development *versus* potential revenue (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; PORTER; KIENTZ, 2013; PORTER, 2014; Aguado-Delgado et al., 2018; YUAN; FOLMER; HARRIS, 2011).

We have been working towards improving game accessibility – especially towards enabling as many people as possible to create and play digital games. Our lemma is "games *by* everyone, *for* everyone", emphasizing the importance of inclusion and of respecting (dis)abilities of creators and players, as well as recognizing that creators may have different interaction needs than those of players (and vice-versa)[2]. To support the lemma, we have defined a framework with three pillars – a run-time tailorable architecture which enables full input and output definition during use time, which is an evolution from (GARCIA, 2014)[3]; a game creation platform (Lepi) that implements that architecture itself and to generate the games made with it; and a collaborative work model – our Collaborative Co-Creation of Inclusive Digital Games (C3-IDG) – to define and guide an end-user game development process.

The C3-IDG conducts inclusive game co-creation of tailorable games. It aims to transform skills and abilities into opportunities to improve accessibility and usability in games. We aim to change the focus of use *and* creation from what people are *unable* to do to what they are *able* to do. Then, with support from the other pillars, people become able to introduce their contributions into a game. This way, they become able to co-create content and accessibility – potentially at the same time.

In the collaborative work model, contributions provide content redundancy and alternatives of use to enable more people to use a digital system. Each small contribution from a person combines to every other previous contributions from the community. As a result, a combination of small individual steps result into giant leaps towards universal access, for they address interaction needs successively.

Our intended audiences are not professional programmers, neither accessibility experts. However, in our evaluations on non-average users created and played digital games[4], we observed that participants managed to co-create accessible games for people with different interaction needs than their own.

Although the C3-IDG was conceived for games, many concepts also apply to interactive digital systems in general. As a result, we generalized a smaller contribution model from it – the Collaborative Co-Creation of Inclusion (C3-I). In this chapter, we describe the C3-I and the C3-IDG. This chapter is organized as follows. Section D.2 discusses related work. Section D.3 discuss how we can expand the current Literature towards collaborative end-user development of universal access – especially for digital games. Section D.4 and Section D.5 introduces the C3-I and the C3-IDG, respectively, through our study scenarios. They serve as introductions to the models. Section D.6 presents the C3-I. Subsection D.6.1 describe examples of end-user collaborations to

[2] When reading games created with the framework, remember to think about small, simple games. For non-real time games, conversion of mechanics are not a problem. For real-time games, our current research suggests that, if we can automate it, we can generate accessibility. Regardless, this work acts an alternative timeline with re-start of game development, focusing on accessible creation and accessible play.

[3] We defined the underlying theory for the run-time interaction Tailoring is described in Appendix B.

[4] As end-users – including people with disabilities and/or situations of vulnerability – participated actively in our models, we followed research ethics protocols throughout the entire processes. Certificado de Apresentação de Apreciação Ética from Plataforma Brasil: CAAE: 89477018.5.0000.5504.

provide interaction alternatives to digital systems. In Section D.7, we describe the C3-IDG. Section D.8 and Section D.9 discusses the results and our conclusions.

D.2 Related Work

D.2.1 Inclusion

The World Wide Web Consortium (2016) defines inclusion as:

> Inclusive design, universal design, and design for all involves designing products, such as websites, to be usable by everyone to the greatest extent possible, without the need for adaptation. Inclusion addresses a broad range of issues including access to and quality of hardware, software, and Internet connectivity; computer literacy and skills; economic situation; education; geographic location; and language — as well as age and disability.

We adopt this definition in this chapter because it is broad and coined for the Web, which has traditional accessibility standards – which is not the case for every domain, such as digital games.

According to Pozzi & Bagnara (2013), software developers often make assumption regarding the abilities of their end-users. As such, people who deviate from these abilities might be unable to interact with resulting systems. In fact, Mankoff, Hayes & Kasnitz (2010) affirm that “the person designing a piece of software is, in some sense, defining who is disabled with respect to that software”. As systems are commonly designed for average users instead of individuals, this means that many people may be unable to interact with the resulting system.

Unlike hardware or physical solutions, software is a construct of instructions and data. With techniques such as tailoring, developers can design software flexible to its context of use (NERIS; BARANAUSKAS, 2009; NERIS; BARANAUSKAS, 2012; PIPEK; KAHLER, 2006). Tailoring explores software flexibility to promote use-time adaptation; with tailoring, end-users may modify interfaces to suit their preferences and needs.

In this chapter, we argue later (in Section D.3) that, if we applied tailoring towards universal access, we could start changing the preamble of the definition of inclusion from products “without the need of adaptation” to products designed *for* adaptation. This is a solution we have explored previously for digital games in (GARCIA; NERIS, 2013a; GARCIA, 2014; GARCIA; NERIS, 2014).

D.2.2 Digital Game Accessibility

Albeit game accessibility is an emerging study topic, there are studies providing advice on how to design and implement suitable solutions for broader public. They may range from guidelines (YUAN; FOLMER; HARRIS, 2011; International Game Developers Association, 2004; ELLIS et al., 2013; BARLET; SPOHN, 2012; GARCIA; NERIS, 2013b) to approaches promoting more accessible game design and implementation (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011; GARCIA; NERIS, 2014; GARCIA, 2014; TORRENTE et al., 2015).

However, there is a divide between what the academic community recommends, and what the industry effectively needs and uses; as a result, games have not benefited so far from accessibility standards (PORTER; KIENTZ, 2013; PORTER, 2014; Aguado-Delgado et al., 2018). Most games are not compatible with traditional accessibility solutions; in the rare cases when games support assistive technologies, interaction quality and usability are usually poor (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; PORTER; KIENTZ, 2013)[5].

These issues persist to this day. In a recent systematic review, Aguado-Delgado et al. (2018) found that:

> None of the studied initiatives can guarantee universally accessible video games:
>
> 1. Approaches that involve particular interfaces and custom developments may lead to a lack of quality in the generated/adapted video games and the segregation of disabled players.
> 2. The frameworks often depend on the use of specific technologies. This dependence hinders accessibility, since it necessarily involves the use of elements that can be beyond the control of the developer (they belongs [*sic*] to third parties) and limit the available options.
> 3. The guidelines, techniques, strategies, etc. that facilitate the development of accessible video games are ignored by many developers (due to ignorance, lack of comprehension, impossibility of application, etc.).

Nevertheless, these problems did not stop people from playing games; for example, there are more than 100 million players with disabilities worldwide (BARLET; SPOHN, 2012). There are accessible games, developed for specific disabilities – examples include audio games for visual disabilities, and one switch games for motor disabilities (YUAN; FOLMER; HARRIS, 2011; International Game Developers Association, 2004) and there are a few Universally-Accessible Games, aiming for inclusion of a broader public (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2011). There are also the rest of existing games, that people may want to play regardless of their abilities. In these cases, a strategy is to request assistance from family and friends, who offer their abilities to remove accessibility barriers temporarily (PORTER; KIENTZ, 2013).

[5] There are efforts to improve this situation. For example, Microsoft's Xbox Adaptive Controller (<https://news.xbox.com/en-us/2018/05/16/xbox-adaptive-controller/>) might become a viable alternative to mediate the use of assistive technologies to provide input in games.

Another is to play against all the odds; people use their abilities – and good design features which provide “accidental” accessibility features – as well as they can[6]. People will find a way to play.

There are, however, opportunities for improvements. Aguado-Delgado et al. (2018) concluded their systematic review stating that:

> a specific development methodology arising from the adaptations and/or modifications of existing solutions could have a notable impact on the level of accessibility of the video games.

Later, we further argue that the methodology should be for developers and for players. If developers and the industry do not follow the methodology, the situation will not change. In our opinion, digital inclusion is only truly complete once people can use *and* create. This way, they could potentially improve solutions even if the original developers do not plan to.

D.2.3 End-User Development and End-User Game Development

End-User Development (EUD) enables end-users to act as non-professional software developers with activities including parameterization, macros, visual programming languages (VPL), programming by demonstration (PbD), and scripting (LIEBERMAN et al., 2006). These activities try to lower technology barriers to enable end-users to create or modify digital systems. To map EUD studies for EUGD, we performed a systematic literature review in 2017[7]. We wanted to find out what were the approaches (and respective audiences) that enabled end-users to create their games. We selected 63 out of 542 unique studies matching our inclusion criteria.

[6] The interested reader may refer to <https://www.youtube.com/watch?v=nmmqarQRSSE&list=PLXcBFfRlLcpiGcMcdCpJ_L1Okqm2uQvrL> for an example on which a blind player describes how he plays a traditional *video* game.

[7] Search string: `("End-User Development" OR "EUD" OR "End-User Programming" OR "EUP" OR "Modding") AND ("Digital Game" OR "Game").`

Databases: Association for Computing Machinery (ACM) Digital Library, Institute of Electrical and Electronics Engineers (IEEE) Xplore, Science Direct, and Scopus.

Inclusion criteria:

1. Study described challenges related to EUGD;
2. Study described approaches related to EUGD;
3. Study described end-user creation process;
4. Study described activities on which modification explored programming;
5. Study described activities on which end-user modification did not require programming;
6. Study described how an end-user approached game creation.

For development activities, we found three important recommendations. The first were four desirable criteria for EUGD tool by Resnick & Silverman (2005) and Burke & Kafai (2014), stating that tools should:

1. Be intuitive and accessible for new users.
2. Allow the creation of complex applications.
3. Allow the creation of a wide range of applications.
4. Promote developing in communities – encourage discussions, help, collaboration, and learning among users.

Yet, in our review, we noticed that, although there existed multiple approaches, they had similar intended audiences – usually young anglophone students. No study addressed accessibility towards broader inclusion for heterogeneous interaction needs and abilities. For instance, Earp (2015) noticed in his systematic literature review that inclusion in game making for learning is, usually, "a strategy for addressing the under-representation of girls and women in computing". This accounts for an important factor (gender inclusion); we should include other audiences as well.

The second was the need for "gentle slopes" – gentle and incremental increases of creation complexity over time (PANE; MYERS, 2006; IOANNIDOU; REPENNING; WEBB, 2009; REPENNING; IOANNIDOU, 2006a; IOANNIDOU; REPENNING; WEBB, 2008; KAUHANEN; BIDDLE, 2007). One approach to achieve them was defining layers for creation, exploring different EUD activities (TONON; BAECKER, 2010; van Herk; VERHAEGH; FONTIJN, 2009). Following this approach, VPLs, PbD, macros, scripting, and traditional programming languages can co-exist into a same tool, to suit needs of different creators. As a result, layers can define a path for learning programming and to develop more complex games over time. Another approach was transitioning from consuming (use) to creation practices – the Create-Consume Spectrum defined by Basawapatna et al. (2014) proposes that a "gentle slope" spectrum could explore, in order: animation, interactive simulation, collective simulation, construction set simulation, pattern-based authoring, end-user programming, to, finally, reach traditional programming.

Finally, the third recommendation was the importance of community, which provides benefits including collaboration, sharing, learning and teaching, and co-creation (BURKE; KAFAI, 2014; REPENNING; AMBACH, 1997; AHMADI; JAZAYERI, 2014; JAZAYERI; AHMADI, 2011; AHMADI; JAZAYERI; REPENNING, 2012a; AHMADI; JAZAYERI; REPENNING, 2012b; PENA, 2011; UZUNBOYLU; BAYTAK; LAND, 2010, 2010; POOR, 2014; COMUNELLO; MULARGIA, 2015; HONG; CHEN, 2014; REPENNING; IOANNIDOU, 2006b). As collaboration was mentioned multiple studies, the next step was identifying how CSCW and EUD could promote inclusion; this way, people could become able to create and play regardless of (dis)abilities.

D.2.4 CSCW and EUD for Accessibility

The CSCW community has since been involved in accessibility research. However, to the best of our knowledge, there seem to be few studies considering people with disabilities as creators of inclusion in software. Rather, they are traditionally portrayed – at time even unfairly – as recipients of assistance (BENNETT; BRADY; BRANHAM, 2018). In particular, it seems that most studies consider a perspective centered on individuals, with the goal of promoting independence for people with disabilities (BENNETT; BRADY; BRANHAM, 2018; LIU; DING; GU, 2016a). As a result, although there are design principles, recommendations, and guidelines to improve accessibility through crowdwork (BIGHAM; LADNER; BORODIN, 2011; BRADY; BIGHAM, 2015; KANE, 2007; VOYKINSKA et al., 2016), people without disabilities (often professionals) usually assist people with disabilities (LIU; DING; GU, 2016a; SHIRAISHI et al., 2017) – for instance, in (Mahelaqua et al., 2013; PIPER; WEIBEL; HOLLAN, 2013; FOGLI; COLOSIO; SACCO, 2010; TAKAGI et al., 2008; BIGHAM; LADNER; BORODIN, 2011; BRADY; BIGHAM, 2015; LASECKI; KUSHALNAGAR; BIGHAM, 2014; SHIRAISHI et al., 2017; MAZAYEV; MARTINS; CORREIA, 2016; CARDONHA et al., 2013; HARA; LE; FROEHLICH, 2013; RICE et al., 2013)

Yet, with suitable approaches, people with disabilities can also contribute to include their peers. For instance, Shiraishi et al. (2017) described an approach on which deaf and hard of hearing (HDD) people provided Japanese sign language alternatives support their peers. Buehler et al. (2015), Bennett et al. (2016), and Hurst & Tobias (2011) explored crowdsourcing for end-user creation of (hardware) assistive technologies. Piper et al. (2006) described Participatory Design activities on which they co-designed a cooperative computer game for social therapy with game with adolescents with Asperger's Syndrome and adult moderators. In a case study from Bennett, Brady & Branham (2018), two participants defined the expression "cross-ability cooperation" to refer to how they benefited each other from their abilities – on participant was blind, the other was a wheelchair user. Together, they helped each other overcome issues to perform daily life and work activities.

To extend these success stories, we should consider individuals and communities, as well as interdependence and independence (BENNETT; BRADY; BRANHAM, 2018; LIU; DING; GU, 2016a).

For accessibility as communities, Liu, Ding & Gu (2016a) recommends "communistic interaction": "from each according to their abilities, to each according to their needs". They argue that we should focus on groups of people instead of individuals, and on abilities instead of disabilities. In this way, people can contribute to help themselves and their peers. This is supported by Zyskowski et al. (2015) – a "possible avenue for research is to understand how crowdworkers with disabilities are currently interacting together and helping one another" –, and by Bennett, Brady & Branham (2018) – "access is not only a solution to a disability-related barrier; it is a way of being together and helping one another". Similarly, Habicht, Oliveira & Shcherbatiuk (2012) described that, in a healthcare context, people who helped themselves to overcome a problem could help others in similar conditions.

For accessibility as interdependence, Bennett, Brady & Branham (2018) affirm that interdependence as frame:

- helps to see connections and relations between people and things;
- helps to observe "simultaneous forms of assistance in action", that is, in a given event, there might occur multiple assistance, potentially with people providing and receiving access simultaneously;
- highlights contributions from people with disabilities;
- destabilizes "traditional hierarchies that rank abilities", that is, it asserts that "people with and without disabilities are equal".

Combining inclusion, EUD, and CSCW, systems could provide features to improve their accessibility (KANE, 2007). However, assistance can go beyond technology. Friends, family, acquaintances, and even strangers can help people with disabilities to overcome accessibility barriers at use-time (BIGHAM; LADNER; BORODIN, 2011; BRADY; BIGHAM, 2015; BENNETT; BRADY; BRANHAM, 2018).

As final digressions, Bennett et al. (2016) states that co-creation reinforces the sense of identity in people with disabilities. In the other hand, Liu, Ding & Gu (2016a) mentions that people with severe motor disabilities avoided requesting assistance for the fear of becoming indebted to those who helped them. Finally, for distributed teams, Salehi & Bernstein (2018) describes Hive, an approach to support collective design. Hive intermixes people working towards a common goal with the intent of helping participants understand perspectives from each other. In particular, Hive defines the role of facilitators to manage that workflows and contributions achieve their goal.

D.3 The Pillars of the Framework: Supporting Games by Everyone, For Everyone

In Subsection D.2.1, we highlighted the need for a broad concept of inclusion to support universal access. This could be particularly important in domains such as digital games, (Subsection D.2.2), which does not yet benefit from traditional accessibility solutions.

From Subsection D.2.1, developers define "who is disabled with respect to that software" when they design it. This a harsh truth, because traditional software development restricts interaction to specific input and output devices and interactions. Physical-level interactions defined at design-time results into fixed ways for human-computer interaction at use-time. Fixed means immutable; this is the reason that the preamble for the definition of inclusion specifies "without the need for adaptation". This quote has two underlying assumptions – immutability and passiveness.

First, immutability. Although there exists the possibility for external adaption (for instance, with assistive technologies), its potential is limited. External adaptation cannot access the internal state of a process[8]. In best case scenarios for output – for instance, interpreted languages on which the source-code is available – the tool could inspect the source code and attach to the process to examine it[9]. At worst case scenarios, the tool would have to act as a human perceiving the system – potentially in real time, as with digital games[10]. It would try to extract information from users interfaces, transform it to another media, and convey it to the user – in a digital game, even milliseconds could be too long. Similar reasoning applies to input. In this case, at worst case, the tool would have to act as a human providing input to the system[11]. Although applications such as the Web have standard input mechanisms and interactions, other systems – such as digital games – do not. If the process does not inform where and how to interact, the tool cannot guess.

Within this context, our answer is that we should design software *for* adaptation – not only in the traditional task-oriented EUD perspective, but also for interaction adaptation. This would solve the problem described by Mankoff, Hayes & Kasnitz (2010) – if developers do not assume specific audiences, they do not define "who is disabled

[8] Even in cases that it could scan and monitor the machine's memory, a general purpose tool would not able to understand the data without knowing variables addresses and meanings. The same reasoning applies to provide input; although the tool could modify the memory directly, it would need to know variable addresses before modifying them.

[9] For instance, screen readers benefit from patterns in Hypertext Markup Language (HTML) elements. The interested read may refer to <https://thepaciellogroup.github.io/AT-browser-tests/> for examples of how a screen reader parses a document.

[10] Games use their own technologies and middleware for implementation, which typically do not support assistive technologies.

[11] In this case, automation software can help.

with respect to that software". Instead of designing for an audience, we should develop interactive systems for *no* particular audience, then we enable people iteractively to it (GARCIA, 2014). In this approach, although we have a fully interactive logical system from the start, everyone starts unable to interact with it. The system has semantics of use, commands for interaction, and primitives representing the information to convey. For each semantic, developers define multiple, composable interaction alternatives to define physical-level interactions for input and output. In turn, end-users become able to build their own way to interact with the system, based on alternatives they combine. In other words, there is no interaction until a end-user defines her/his own. Once this happens, she/he have the interaction that she/he has chosen. This may sound paradoxical, as it is not a conventional approach; we provide two prototypes for demonstration at <https://gitlab.com/francogarcia/RunTimeTailorability-PingPong> and at <https://gitlab.com/francogarcia/GamesRunTimeTailorability>.

The approach relies on the principle that, unlike physical products, software can change indefinitely at run-time within the limits of Turing completeness (and the Halting problem) (SIPSER, 2005; HOPCROFT; MOTWANI; ULLMAN, 2000). It is the underlying (also software) architecture that limits software flexibility. Therefore, with suitable architectures, software can change at run-time to redefine interaction with accessibility purposes (GARCIA; NERIS, 2013a; GARCIA; NERIS, 2014; GARCIA, 2014). A refinement of this work resulted into the first pillar of our framework — an architecture which enables software able to fully redefine human-computer interaction at use-time. Following the approach, to build systems for everyone, we can start designing *interactive* systems for *no one*. As we work with games, we called this system a Meta-Game. The fully simulation runs on it, interactively, without human-intervention. All the interactive elements exist there; input exists as a semantic abstraction (command). Output does not exist, for machines do not need it; there is data. The task of developers, thus, is to define physical-level interactions mapping abstract input to human-input, and computer data to human-output. In particular, they can create multiple interaction alternatives. As a result, players can choose among existing alternatives to define their own interaction with a system – which we called a Game. A Game is a Meta-Game attached with elements to enable human interaction – *the Game to the Player*. As interaction alternatives are composed to the system, detaching them all reverts a Game to its unique Meta-Game. The process, thus, is reversible. The architecture, therefore, provides a way to implement games *for* everyone – provided that we can define interaction alternatives to support their needs[12].

[12] Although the framework is for games, most approaches apply to any interactive digital system. Our game creation platform Lepi, is a system implementing the architecture. As strategies for the architecture originated from game programming, we opted to focus the framework on games. Our reasoning is that we can extract generic parts later, to extend the approach for other domains – as we did with the C3-I, which resulted from the C3-IDG. Moreover, we found digital games suitable for the framework

At this point, we can reach the second assumption – passiveness. If we design for no end-user adaptation, we assume that end-users are passive; they are recipients of technology. However, if we design for adaptation, we design for change – for activeness. From Subsection D.2.4, most studies assume an individual perspective to accessibility. When there is a mismatch between the interaction design of a system and the real abilities of its end-users, the system might become inaccessible. Hitherto, there are three main approaches to solve this mismatch:

1. End-users try to use the system as it is – with an assistive technology, when possible;
2. End-users request developers to fix it;
3. End-users request assistance from a community (for instance, close persons or via crowdwork) to overcome the problem.

In the first two approaches, the underlying premise is that digital systems are static entities. Developers deploy new versions, end-users interact with them. This matches traditional software approaches. In the third approach, traditional software approaches do not consider that the contributions might become part of the original system.

The CSCW and EUD communities, in particular, recognize that people cooperate to co-create digital systems, for end-users can modify and improve them. In this perspective, digital systems are dynamic entities. They can change, end-users can change them; developers do not have to be the sole responsible for changes.

When systems allow EUD, end-users for whom the system is currently usable join developers as potential creators. Inclusion, by definition, "address a broad range of issues" (Subsection D.2.1). When end-users are potential creators, they become able to provide their own skills and abilities to improve systems. In particular, this can help with accessibility – Item (3); the state of art.

To extend it, we should consider inclusion as a dynamic, iterative, and collaborative process. Inclusion should not remain a dichotomy of accessible/usable for some, inaccessible/unusable for others. Rather, it should be dynamic – accessible/usable for some, *currently* inaccessible/unusable for others. Although subtle, the difference is meaningful, because inclusion becomes iterative. Once included, people become potential co-creators as well. This is valid for the whole "broad range of issues". Once included, people *with disabilities, different computer skills, knowledge, and socioeconomic situations* become potential co-creators. Once included, anyone can potentially help to further improve the system to themselves and for others. In special, one can enable someone

(Subsection D.2.2). More constraints lead to more robust solutions; it is easier to generalize a solution from more to fewer constraints than the contrary.

else to use a digital system – from currently unusable, the system may become usable to her/him. This newly-included person becomes another potential co-creator. If she/he – or anyone else already included – can further contribute, the dynamic process continues.

We can see this as a possible application of "communistic interaction" described by Liu, Ding & Gu (2016a) applied to software. People contribute with their abilities, with what they are able to do and to provide. Besides communistic, we argue that it is also a mutualistic[13] process on which individuals benefit the community (with new contributions), and the community benefits individuals (from previous contributions). Contributions add to each other over time; every new collaboration complements every previous one. This could be especially useful for purposes of universal access – that is, provided people could collaborate, regardless of their interaction needs.

In Subsection D.2.3, we moved our focus to digital games. The Literature provided strategies to enable EUGD – desirable criteria, the need for "gentle slopes", and the importance of community. If we implemented an EUGD approach which built Meta-Games and Games exploring the architecture, we could enable people to make games *for* everyone – provided that there were people who could create interaction alternatives. For everyone is good, but it is half the battle towards universal access; more people need to create.

If we consider inclusion as a dynamic process, people could co-create interaction alternatives. If the prior EUGD approach implemented the architecture itself, we could enable game creation *by* everyone. Current audiences would be limited by currently available interaction alternatives[14].

Still, end-users are not professional developers. We have to help them to create their games and define interaction alternatives. Although they could not develop tailorable games alone, we assumed that, with suitable computational support – the game creation platform –, they could co-create together. Communities as well as individuals, and interdependence as well as independence. With these focuses, we defined the last pillar of the framework: the C3-IDG. With the work model, cooperation and assistance become part of development processes. Inclusion becomes a dynamic, iterative process. Rather than passive recipients, end-users become active software co-creators, who can include the community and be included by the community; they can promote accessibility and improve usability.

Although the proposed perspective defies the *status quo*, our current (small) study scenarios (with a small subset of possible interaction needs) suggests that people could

[13] Mutualism is no longer a new term, it has first been used in Biology to describe a relationship in which two organisms benefit from each other.
[14] This is a longitudinal, long-term goal. A smaller step is enabling one audience at a time. We learn from them to enable them to enable others.

co-create once they had computational and community support – and enable everyone in the community to play their creations. We start this chapter from the scenarios. Our intention is inducing possible collaborations and workflows, hoping to show how daily life skills can support end-user contributions towards digital accessibility, and that end-users can extend systems with those contributions.

We start with the C3-I (Section D.4), to highlight how people can contribute based on their abilities and how inclusion is an iterative, dynamic process. Next we advance to the C3-IDG (Section D.5), with a real world study on which people with low literacy and who had never used computers created games for themselves and their peers. In these sections, it is important to note that:

- Accessibility and inclusion become iterative, incremental, and dynamic processes;
- The workflows are generic enough to support other collaborative workflows, if needed;
- The workflows are comprehensive accessibility-wise, that is, they support diversity and multiple (dis)abilities instead of particular ones;
- People are empowered and active. Besides informing their needs, they can actively contribute to co-create digital inclusion;
- Any included people may collaborate based on what they are able to do – abilities before disabilities;
- Inaccessibility is a result from a lack of interaction alternatives in a digital system. Once we introduce suitable alternatives to the system, they might enable people who needed them to use and create[15];
- Inclusion becomes transient – included *versus* to be included, instead of a dichotomy included *versus* excluded;
- Upon inclusion, people for whom the system was previously inaccessible might become potential enablers of others;
- As changes are incorporated to the system (software), interaction alternatives may minimize dependence upon assistive technology;

[15] A jigsaw puzzle acts a metaphor for this perspective. We can visualize (tailorable) games as building jigsaw puzzles – the combination of puzzle pieces contributes to the creation of the final interactive system (the game). Each person has her/his own personal assembling board. Although a part of the board is the same for everyone (the Meta-Game), individual boards may vary slightly from person to person, as every one has her/his own interaction needs. When there are suitable pieces to fill the board, a person can play the game. When there are not, the community creates new pieces. However, these are software pieces – the only limit for software is Turing completeness. Developers create pieces. End-users shall be able to create pieces as well.

- Contributions are not restricted to code; rather, daily life skills can promote inclusion;

- Following the Literature, workflows focus on communities as well as individuals, and independence as well as interdependence (LIU; DING; GU, 2016a; BENNETT; BRADY; BRANHAM, 2018). Likewise, people can help each other at use time provide temporary assistance to enable use (BIGHAM; LADNER; BORODIN, 2011; BRADY; BIGHAM, 2015).

D.4 Scenario: Co-Creation of Tailorable Games in an Inclusive School

One of our study scenarios is at an inclusive school. In a classroom, among the students there are people with hearing or cognitive disabilities. The real case scenario is still a work in progress; although we provided a capacitation course for professors, we have not performed activities with the students yet. Therefore, in this section, we describe an expanded hypothetical situation based on our study scenario. In this situation, fictitious participants add heterogeneous interaction alternatives gradually to fulfill needs – from simpler and easier to understand, to more complex and richer situations. Successive contributions from the participants will induce the model logically.

In a bilingual classroom, there are Anglophone and Lusophone students, with varying proficiency and skills. Table 2[16] describes abilities of hypothetical students; to maximize collaborations, no one is perfectly fluent on both languages at the same time.

[16] Some students in the table cannot talk and/or listen, or read and/or write in both languages. This is intentional for later iterations.

Table 2 – Reading, writing, speaking, and listening proficiency for students of a hypothetical inclusive classroom.

	Portuguese				**English**			
Student	**Read**	**Write**	**Speak**	**Listen**	**Read**	**Write**	**Speak**	**Listen**
Ana			✓	✓				
João	✓	✓		✓				
Isabela	✓				✓	✓	✓	
Pedro								
Linda					✓	✓		
James					✓		✓	
Robert								✓
Elizabeth								

Source – Created by the author.

Note – ✓: Student has the ability.

If we analyze Table 2, to include every student, (1) the content should be able in both languages in written and spoken forms, and (2) input alternatives should match motor abilities. Although Maria, the teacher, may be fluent in both languages[17], she would rather let their students practice. Rather than doing it herself, she will act as a Supervisor, identifying students whose abilities and skills allow them to translate content between languages, and to convert between written and speaking representations.

Maria selects Ana to start creating a story for a game. Ana can speak Portuguese; she records a narration of her story. When Ana finishes, Maria knows that only Ana and João can play a game in Portuguese; the other students cannot. João can listen to and write Portuguese, so Maria asks him to transcribe her narration into written text. João does so, becoming the first Collaborator, an Enabler – someone whose contribution enabled people (Isabela, in this case) to interact with a previously inaccessible content. João does not only provide temporary assistance; he enriches the original game with his provided interaction alternative. He, therefore, helps people who can read Portuguese at any future interactions. In this case, it helped Isabela.

Isabela can read Portuguese and write (non-perfectly) English. Maria requests a translation from Isabela, who becomes the second Collaborator – an Enabler – and adds her contribution to the game project. Her collaboration includes students who can read English – Linda and James.

Maria knows the translation has spelling and syntax mistakes. She requests a review from Linda. Linda becomes the third Collaborator, an Enhancer – someone who improves the quality of existing artifacts. As Linda cannot read Portuguese, it was the collaboration from Isabela who enabled her to understand the story and further collaborate. Abilities and skills may provide interaction feature to full interaction needs from others. As a result, Linda became an active participant. She was assisted, she became able to assist.

In its current state, people are able to play if they can read either language or listen to Portuguese. Maria continues her supervision, requesting James to record his narration of the story and insert into the game. With James' contribution, Robert became able to listen to Ana's story[18]. James, therefore, is the fourth Collaborator (an Enabler).

Elizabeth and Pedro want to play Ana's game as well. Their entries in Table 2 where blank – they cannot read, neither can they hear. Table 3 describes disabilities of the students. It does not make the situation any different, though. According to Table 3, Ana and James had disabilities. They contribute successively because collaborations came from abilities, skills, and knowledge.

[17] She/he does not need to, though – collaborations will fulfill needs as the scenario unfolds.

[18] Text to speech versions help may support low literacy and visual disabilities. Accessibility improvement meant for an interaction need may help others.

Table 3 – Abilities for the students of the classroom of Table 2.

	Physical Disability			
Student	**Vision**	**Hearing**	**Speech**	**Motor**
Ana	✓			
João				
Isabela			✓	
Pedro		✓		
Linda				
James				✓
Robert				
Elizabeth		✓	✓	

Source – Created by the author.

Note – ✓: Student has the disability

For plot convenience, João and Robert know Língua Brasileira de Sinais (LIBRAS) and American Sign Language (ASL), respectively. João becomes a Collaborator (Enabler) once again, in his second contribution and the fifth collaboration overall. Robert provides the sixth collaboration, becoming a Collaborator (Enabler). They insert their creations into the game; they were not experts and made mistakes; Pedro and Elizabeth were glad to fix them as soon as they could play the game (seventh and eighth collaborations), becoming Collaborators (Enhancers).

In this hypothetical scenario, it was easy — perhaps even cheating – to make João and Robert know sign language. However, when working at inclusive institutions, there are domain experts – or other people – who can help to overcome accessibility barriers. Outsider collaborators – parents, friends, colleagues – may also help; if people can request improvements, they generate opportunities for contributions. We focus on what people *can do and how they can contribute* to enable others. Even imperfect contributions can help, as they may include someone to improve them (like Pedro and Elizabeth).

In this particular example, in eight collaborations, seven different people contributed with something they knew how to – and could – do (Table 4 summarizes their collaborations). However, if João, Isabela, James, and Robert had not contributed (and had not received earlier contributions), only João would be able to enjoy Ana's game. Isabela, Pedro, Linda, Robert, James and Elizabeth would miss it, even though, together, they could co-create accessibility. As they co-created, any person to whom there exists suitable interaction alternative can play the game.

Table 4 – Summary of contributions for the classroom scenario.

Student	#	Contribution	Role
Ana	0	Original content creator (story narrated in Portuguese)	
João	1	Transcription of Portuguese audio to Portuguese text	Enabler
	5	LIBRAS video based on the Portuguese written text	Enabler
Isabela	2	Translation of written Portuguese text to written English text	Enabler
Pedro	7	Improvements to the LIBRAS version	Enhancer
Linda	3	Proofreading and review of written English text	Enhancer
James	4	English audio version of text	Enabler
Robert	6	ASL version based on the English audio	Enabler
Elizabeth	8	Improvements to the ASL version	Enhancer

Source – Created by the author.

Note – ✔: Student has the ability #: Contribution number

D.5 Scenario: Inclusive Game Creation to Aid People Undergoing Alcohol and Drugs Rehabilitation

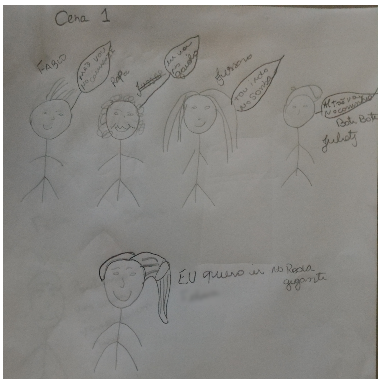
Our second study scenario is at an alcohol and drug rehabilitation public healthcare service. The interaction needs of participants in this scenario is different from those in the school. In this section, we explore cognitive, emotional, and literacy abilities of users of the service undergoing alcohol and drugs rehabilitation – design considerations for the public are described in (RODRIGUES et al., 2014; RODRIGUES et al., 2015).

Besides the two authors of this chapter, two healthcare professionals of a Centro de Atenção Psicossocial – Álcool e Drogas (CAPS-AD), ten users of the service, and five Collaborators (two Computer Science and one Nursing undergraduate students, one MSc in Computer Science, and one PhD in Nursing) participated in the study. In the study, the participants co-created games. Users of the service shared their stories and experiences about alcohol and drugs rehabilitation. The users of the service were young adults to adults, with varying degrees of literacy (from being unable to read and write, to having basic skills) and computer knowledge (from never having using a computer to using them at work). No one had programming nor game design experience. To promote a first contact with computers and digital games, we explored three games that we had created for mouse and keyboard training (one for mouse, one for keyboard, one for both).

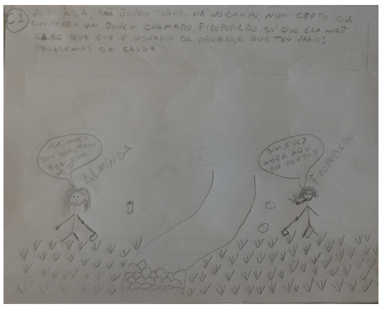
The healthcare professionals participated as Supervisors. The users of the service assumed the roles of Creators and Players, depending on the context. We first describe

Figure 27 – Each participant approached the Conversion phase differently, creating their stories based on their preferences.

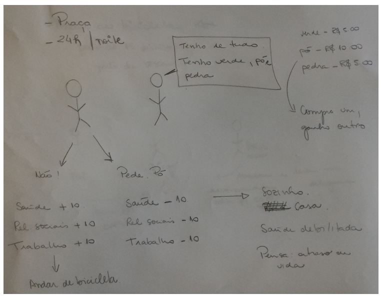
(a) Project 1: Story in the format of a comic book.



(b) Project 2: Story in the format of a children book.



(c) Project 3: Story dictated by the Creator and transcribed by a Collaborator.



Source – Created by the author.

how three participants designed their games. Although the first (Project 1) did not implement her game, the other two (Projects 2 and 3) were implemented by the participants using Lepi. Each of these three participants started their project differently; thus, we can describe three different approaches for the game creation.

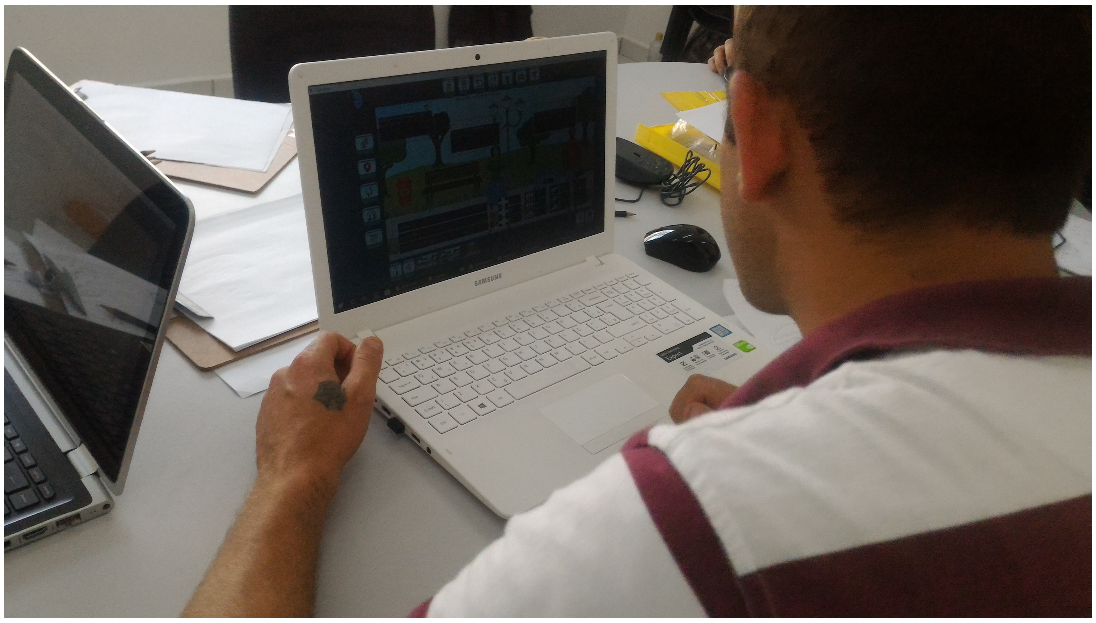
In the C3-IDG, we define a game creation process with nine phases (Subsection D.7.1; Figure 32). Game creation starts in the Conception phase. In this phase, Supervisors define their goals for the project. As the healthcare professionals were not experienced with digital games, we helped them to define the scope and goals of the projects. For these three projects, the goal for Creators was to share experiences related to the use of alcohol and drugs using the branching storytelling activity. At each branch of the story, a Creator would have to define a prompt for the Player with different forms of addressing the situation. For each choice, the Creator would define a score affecting three parameters defined by the Supervisors: health, social relationships, and work.

After the Conception, comes the Conversion. In this phase, Creators define initial prototypes for their game projects (Figure 27). Each Creator from these three projects had different abilities and skills. The Creator of Project 1 (Creator 1) drew her story in the format of a comic, writing short phrases in balloons to define dialogues (Figure 27a). The Creator of Project 2 (Creator 2) drew his story in the form of a children's book (narration on top of the page, drawings on the bottom – Figure 27b) The Creator of Project 3 (Creator 3) participant could not write. He dictated his story to a Collaborator, who made a diagram to represent it (Figure 27c).

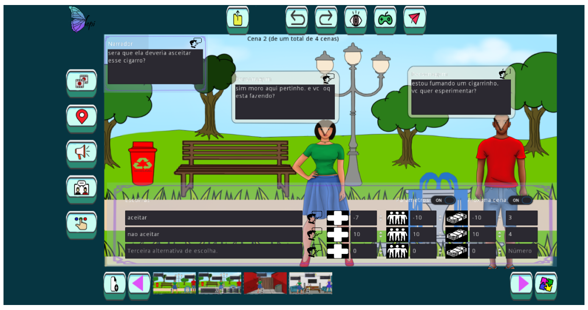
After a Creator finished her/his project, the process started an Evaluation phase.

Figure 28 – Creation phase using Lepi.

(a) Project 2: Creator using Lepi.



(b) Project 2: Designing a prompt to request a decision from the Player.


Cena 2 (de um total de 4 cenas)

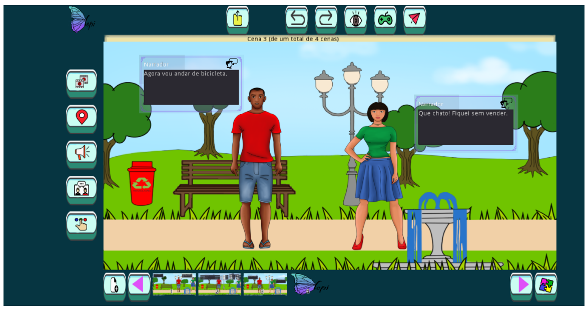
Narrador: sera que ela deveria asceitar esse cigarro?

sim moro aqui pertinho. e vc oq esta fazendo?

estou fumando um cigarrinho. vc quer esperimentar?

Escolhas				Próxima cena
aceitar	-7	-10	-10	3
nao aceitar	10	10	10	4
Terceira alternativa de escolha.	0	0	0	Número

(c) Project 3: Good ending, after a decision that the Creator designed as positive.


Cena 3 (de um total de 4 cenas)

Narrador: Agora vou andar de bicicleta.

Narrador: Que chato! Fiquei sem vender.

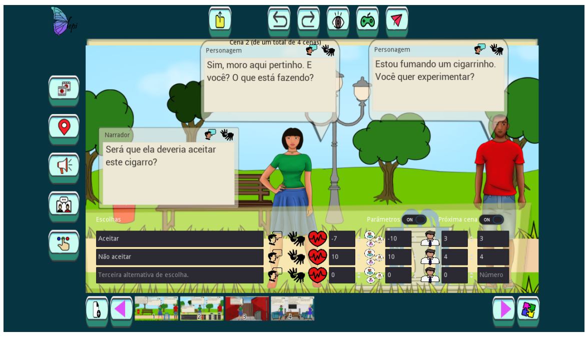
Source – Created by the author.

Supervisors analyzed each project, asking questions and providing advice to its Creator. Upon approval, Creators started the Creation phase. In the Creation phase, Creators used Lepi to create high-fidelity game prototypes (Figure 28). Once again, the approaches to implementation varied by Creator. Creator 2 built his game in Lepi himself, without any external help (Figure 28a and Figure 28b). The Creator 3 created entities (characters, places, and things) for his game, and added dialogue boxes to his scene (Figure 28c). Although he could not write, the Collaborator offered to spell each word for him, so he could type them. He typed a few, then smiled and refused to proceed. Rather, he asked the Collaborator to type it from him.

A new Evaluation phase succeeded the Creation. Supervisors knew that some participants could not read, and that most participants would have difficulties reading. Upon inspection of the projects, they also noticed that the texts would benefit from

Figure 29 – Collaborators created an audio alternative to text on an Enrichment phase, which the Supervisors approved for inclusion, resulting into projects with text and audio output alternatives.

(a) Project 2: Updated version of Figure 28b, with the text to voice collaboration.



Cena 2 (de um total de 4 cenas)

Personagem
Sim, moro aqui pertinho. E você? O que está fazendo?

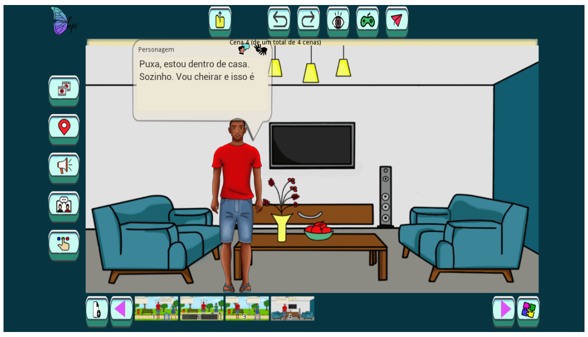
Personagem
Estou fumando um cigarrinho. Você quer experimentar?

Narrador
Será que ela deveria aceitar este cigarro?

Escolhas — Parâmetros ON — Próxima cena ON

Aceitar	-7	-10	3	3
Não aceitar	10	10	4	4
Terceira alternativa de escolha.	0	0	0	Número

(b) Project 3: Bad ending, resulting from a choice the Creator designed as negative.



Cena 4 (de um total de 4 cenas)

Personagem
Puxa, estou dentro de casa. Sozinho. Vou cheirar e isso é

Source – Created by the author.

revision – spell checking, punctuation, capitalization, and accentuation. Therefore, they determined that the next phase would be Enrichment – they would request Collaborators to review the text and provide speech as an output alternative. This way, participants would be able to play regardless of reading skills. A first Collaborator improved the text of the projects (Enhancer). A second Collaborator used the reviewed texts to create the spoken alternative of use (Enabler). Once a Collaboration happens and is accepted, the resulting artifact can become part of the system, enabling people who needed it (or improving interaction and usability for everyone who used it before). Lepi uses a metaphor of slots to represent media alternatives. This is a differential of our approach. It is not limited to one aspect of inclusion; rather, it is general to support multiple, as interaction needs are heterogeneous.

For example, to support Players with hearing disabilities, a Collaborator created LIBRAS videos to complement text and spoken content (Figure 30c). With the LIBRAS video, people could define the written version of the LIBRAS for the game, for instance.

When the alternatives of use (new audio and improved text) were ready, the Supervisor revised the content, and authorized inclusion to the projects (Figure 29[19]). The process was back to Evaluation. With text and audio, Supervisors were able to fulfill the interaction needs for the Players; thus, they decided that the next phases would be Distribution and Use. At Distribution, we generated two versions of the games: one with text, one with text and audio. With those versions, the process advanced to the Use phase.

[19] Screenshots of Lepi may change in figures, for we adjusted its interface to better suit the interaction needs of the participants over the meetings. For example, to improve readability with a larger font size and a light theme for better contrast.

At the Use phase (Figure 30), to our surprise, a strategy which we had not anticipated emerged from the activity[20]. One of the Supervisors conducted the Use phase as an evaluation activity itself – she asked the Creators to explain how they created their games while they were being played by other Players[21]. Thus, she conducted both an Evaluation and a Use phase simultaneously, on which a Creator explained her/his rationales for her/his project, its origin (imagination or past experience), how she/he defined the choices and consequences.

The strategy worked well for Creators and Players – and a fourth user of the service (who did not participate in previous creation activities) who was watching (Person 4). We will highlight two particular reactions that occurred during two Use phases. The first Use phase used a text version of a game. Project 2 started describing his project timidly, often mentioning he did not remember why he had opted for a particular decision. While a Player played his game, a Supervisor asked the Creator to read his story aloud, as Person 4 (currently To Be Included) could not read – she had never gone to school[22]. As the Player progressed at his game, the Creator became bolder and prouder of his creation. The Supervisors were approving his game story, the prompts for decisions that he provided, and the outcomes he had defined (both for the parameters and for the resulting dialogues and scenarios).

His transformation peaked during the second Use phase, on which the Player interacted with the version of his game with spoken text. When the Creator heard his game the first time, his eyes were gleaming of pride and satisfaction, as he perceived the collaboration as a way to increase the quality of his game and enable his friend who cannot read to play his game. Short after, the Creator realized that he could narrate the game with his friends[23]. Then he noticed that he could conceive new stories portraying the CAPS-AD as its space, on which his friends participated. In Homo Ludens, Huizinga (2016) describes the act of playing as entering a "magic circle". Creator 2, then, had just perceived that he, himself, had created his own magic circle, on which he could share his knowledge and experiences with others. As Bennett, Brady & Branham (2018) mentioned, co-creation reinforces a sense of identity in people with disabilities. Based on the reaction from Creator 2, we guess it applies to everyone.

The second reaction worth noting was from Person 4. Besides having a stronger feeling of being part of the group once the was included (first via the reading, next with the audio), the Supervisor's strategy acted as trigger which helped Person 4 to share her

[20] One could say it was a meta-collaboration, as it improved the C3-IDG itself.

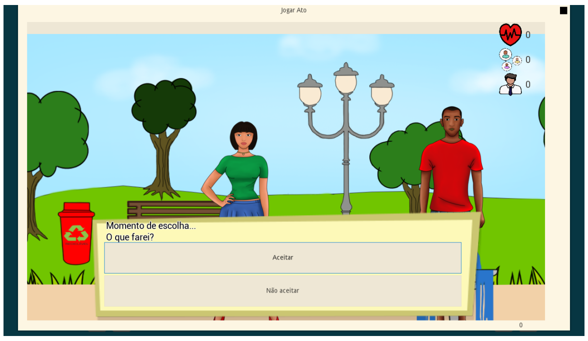
[21] Although we had planned and designed the C3-IDG to enable a Supervisor to use the game as a tool to support her/his practice, we did not foresee her/him using the created game collaboratively with the game Creator and other Players simultaneously.

[22] At this point, thus, the Creator became a Collaborator in the session (Enabler), and Person 4 became Included.

[23] In Portuguese, this is called "jogral".

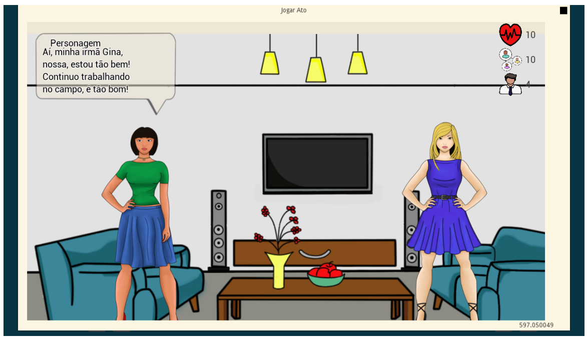
Figure 30 – Use phase session.

(a) Project 2: Player prompted for the decision created in Figure 28b during a Use session.



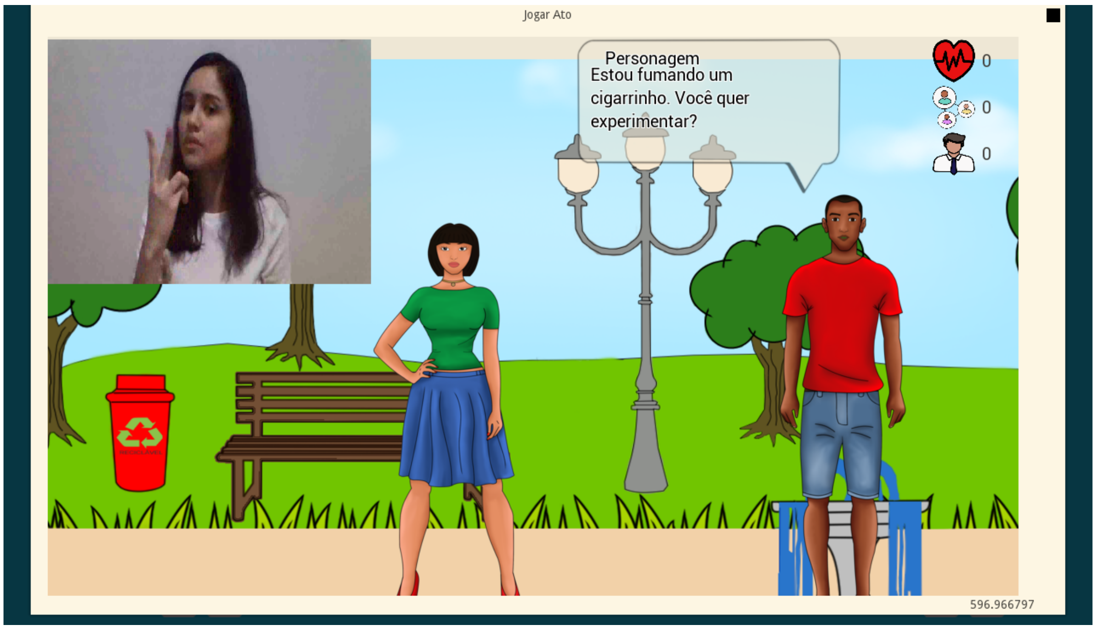
Momento de escolha...
O que farei?

(b) Project 2: Positive modifications in the parameters during use, after the Player made a choice considered good when designed by the Creator.



Personagem
Aí, minha irmã Gina, nossa, estou tão bem! Continuo trabalhando no campo, é tão bom!

(c) Project 2: Image of LIBRAS video added to the project, showed in the sign language version generated at a later Distribution phase a few dialogues before the prompt of Figure 30a.



Personagem
Estou fumando um cigarrinho. Você quer experimentar?

Source – Created by the author.

experiences. In particular, she shared a similar story to that of the Creator. In this way, she both provided insights to the Creator on how to improve the game (acting, therefore, as a co-Creator), and received orientation from the Supervisors for her rehabilitation. She had, therefore, started to co-create the game. Moreover, at a smaller scale, she acted as a Supervisor herself, as she was helping the Creator within the serious context of his game and – especially important – the real Supervisors approved her contributions.

At this point, the Supervisors ended the Project 2 (Conclusion phase). Person 4 showed up showed in the next meetings. If we design *for* change and acknowledge that inclusion should encompass use *and* creation, we step into a uniqueness of our models. From the moment she became able to play, Person 4 became a Player. From the moment we improved Lepi to suit her interaction needs, she became a Creator as well (Creator 4). From yet To Be Included, she became Included; from Player, she became a Creator. Creator 4 was someone who had never used computers before. Although she wanted a smartphone, it was too expensive[24].

Creator 4 was, at first, scared of touching a mouse. Her daughter taught her how to read simple, small words ("yes", "no") and how to write her name. Creator 4 really wanted to learn and "master" the keyboard. With the help from a Collaborator, she narrated her story at a Conversion phase. The Collaborator recorded it. Later, at an Enrichment phase, another Collaborator transcribed her story into text. Likewise, a third Collaborator recorded multiple voices – one for each of her characters. When the Creation phase started, she had digital content for her game. She built the remaining game entities using Lepi to define her characters and scenarios. Then a Collaborator helped her to introduce the media to the game. Based on her abilities and with the help from Collaborators, Creator 4 finished her first game. She played it multiple times, shared with other participants (Use phase). She told her daughter, who could not believe her. Then Creator 4 created a second game – more complex, with more features, receiving less assistance.

From interdependence, Creator 4 started moving towards independence – to the best of her abilities and interaction needs. From someone scared of using a mouse, she became confident in her skills, sure that she could do it. Later, there was a Use phase on which the Supervisors invited other users of the service. In this session, Creator 4 taught people how to use the computer to play her game. Some Players read her story, other players listened to it. She demonstrated Lepi to the Players, showing that they could create games as well.

Overall, there were people who could use Lepi almost unassisted from the start –

[24] ICTs and technology in general are very expensive in Brazil – the "Brazil cost". This fact combined with extreme inequality and awful wealth distribution make access to technology hard for the population. Costs also reflect to infrastructure. For instance, the healthcare service and the inclusive school had a single old each computer for all staff.

both for Conversion and Creation phases. There were people who needed assistance in the beginning, becoming more independent over time. There were people who would need assistance for a very long time – which is fine. They were all co-creating software as non-professional developers – for themselves and for everyone else. No single participant in this study could define interaction alternatives to support everyone else or modify a game to include them; however, the community could co-create them together and use Lepi to combine alternatives. This is not possible with EUD, Meta-Design (MD) (FISCHER, 2009; FISCHER; FOGLI; PICCINNO, 2017), Participatory Design (SCHULER; NAMIOKA, 1993), crowdwork, universal design practices, and tailoring alone. Rather, it requires the best of them all: coordinated collaborations from an active and empowered community, supported by a dynamic inclusion process with iterative contributions towards universal access. This way, universal access become a community issue, which the community itself may be able to improve.

D.6 A Collaborative Work Model for Co-Creation of Universal Access: Individual Collaborations to Reach the Moon

In Section D.4, we described a scenario on which people at a school provided their abilities to enable their peers to play digital games. In particular, they created content alternatives based on skills that they already had. Every alternative could enable others to use, and, potentially, to contribute back. With suitable (meta-)design, digital systems could support similar workflows.

If we analyze the scenario, interaction needs came from people; solutions to address these needs were provided by people. Individually, each one had her/his own needs, skills, preferences, knowledge. Collectively, the abilities from one person could provide solutions to another. In Section D.4, every time that Maria identified a Student A who could not play, she tried to identify a Student B who could provide assistance. Student B used her/his abilities to develop something to help Student A. If the contribution from Student B satisfied the needs of Student A, Student B removed an accessibility barrier for Student A. As defined by Liu, Ding & Gu (2016a), “from each according to their abilities, to each according to their needs” – from Student B, to Student A.

However, we argue that inclusion is a dynamic process. First, once the contribution of Student B was introduced to the system, it can help multiple people with similar abilities to those of Student A. Second, Student A (as well as others) was not only included to the system – she/he became a potential provider of assistance in further iterations. Therefore, Student A was (previously) assisted by the community to (prospectively) assist the community. Thus, she/he does not need to concerned or in debt (as people with disabilities may feel (LIU; DING; GU, 2016a)), for she/he can reciprocate assistance in the future. Once able to use and create, every person is equal. The Collaborative Co-Creation of Inclusion (C3-I), thus, aims to enable people to co-create for inclusion to enable others, who potentially, may further co-create – forming cycles of inclusion. People are assisted to assist – generalizing to software the idea behind the term "cross-ability cooperation" mentioned by two participants in the study of Bennett, Brady & Branham (2018). When the contribution becomes part of the software, it changes from temporary support for interdependence to a feature for independence.

Figure 31 illustrates a cycle of inclusion. Contributions could potentially happen in other orders, in parallel, others with similar abilities, or multiple from a same person. They all benefit the community, potentially increasing its size. Small features add to each other, resulting into greater effects over cycles. Every contribution complements all others; they define different means to achieve a same goal (enable / improve human interaction). We are, after all, designing *for* adaption.

Aristotle's quote "the whole is greater than the sum of its parts" became an important concept in system thinking (SELLERS, 2017; MEADOWS, 2008). The quote means that, when parts are combined, there may emerge properties and behaviors that did not exist separately. This happens here. Individually, the system was usable for some, unusable for others. Together, inclusion was built iteratively. People were yet To Be Included. Once Included, they may further improve the system[25]. Thus, roles for inclusion at a given version (state, time, set of features) of a digital system can be:

Included Person for whom the version of the system is accessible.

To Be Included Person for whom the version of the system is inaccessible. If there is an Enabler who can provide an interaction alternative which makes the system usable to a To Be Included person, she/he becomes an Included person at the next version.

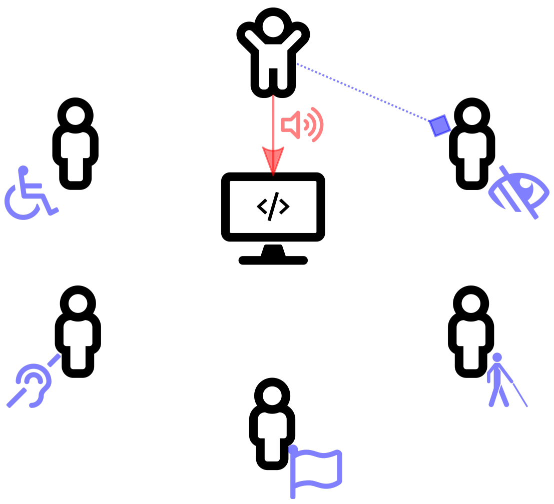
Collaborator A special case of Included person – a person who helps to improve the system. Collaborators can be Enablers or Enhancers.

- **Enabler** At a given version of a digital system, she/he may provide an interaction feature that enables To Be Included persons to use it upon her/his contribution.
- **Enhancer** At a given version of a digital system, she/he may improve the quality of an existing interaction feature. Although it will not include new people,

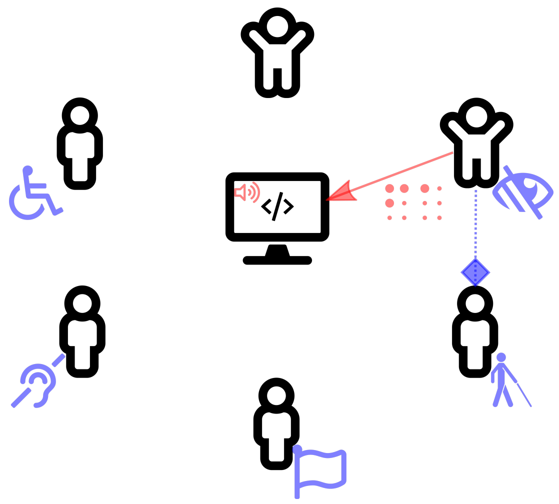
[25] Creation, thus, may come from non-professionals as active makers.

Figure 31 – A diagram for the C3-I.

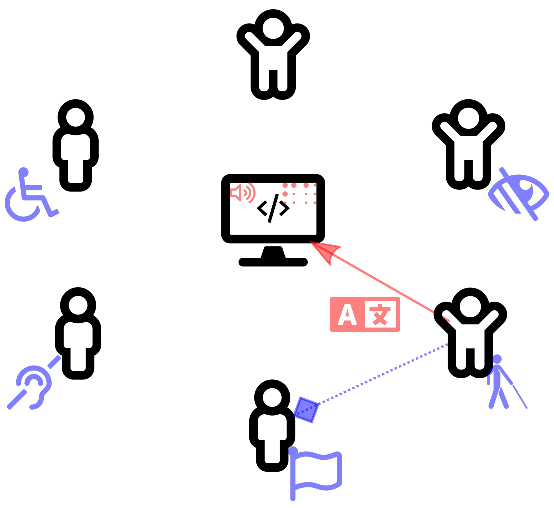
(a) The original system was accessible to a user, who introduced spoken content to enhance it.



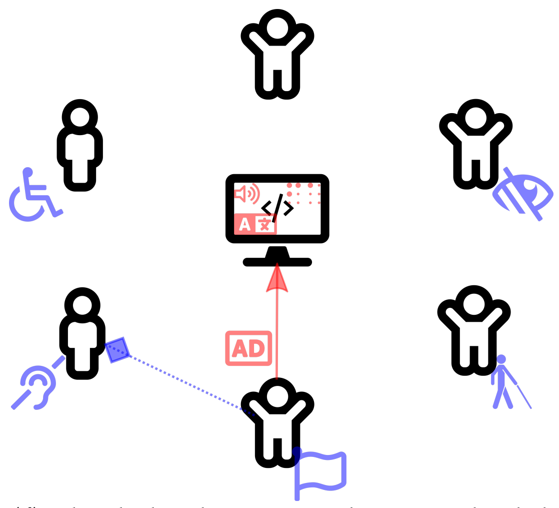
(b) She/he included (new) a user with low-vision, who converted the content to a braille-printer friendly format.



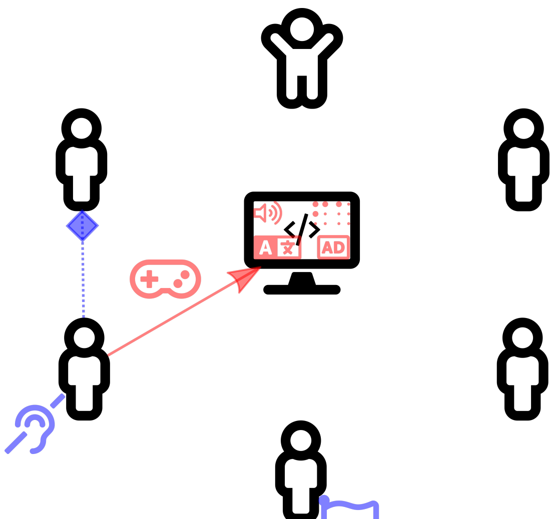
(c) This included a user with blindness, who could translate the content to a new language.



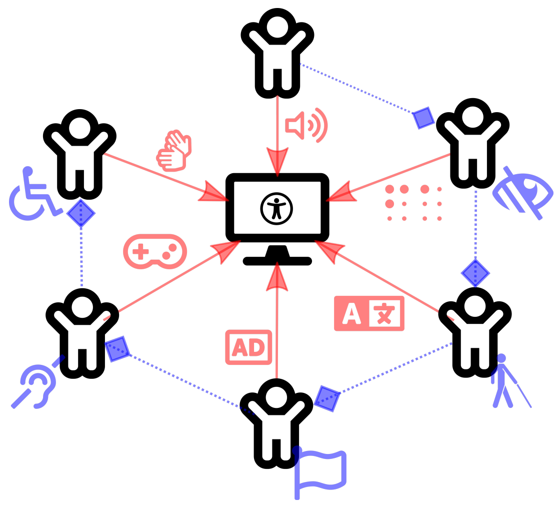
(d) A foreigner user could, then, provide audio descriptions to the content.



(e) A user with hearing disabilities thanked the collaboration with new input mappings to simplify user input.



(f) This helped a user with motor disabilities to interact with the system. Potentially, iterations could happen to reach universal access.



her/his contribution may improve the system for those already Included[26].

These roles are *transient*, potentially varying at a new version of a system. Towards universal access, there must exist Enablers among Included users at each version of a not yet universal system to transform To Be Included people into Included ones. If there are not any, it is not possible to progress into universal access without outside participants, because available abilities and skills of current Included members cannot provide required alternatives for the remaining To Be Included people.

With the roles, we can generalize a workflow from the scenario described in Section D.4 with a few steps. Towards universal access, each iteration follows the same pattern[27]:

1. Identify existing interaction issues;
2. Choose an existing interaction issue;
3. Identify a feature to overcome the chosen issue;
4. Identify people (Enablers) who can create the feature;
5. Request the features to an Enabler;
6. Upon conclusion, add the feature to the system.

Theoretically, a version V_i of the system starts with a set of interaction features F_i, and a set of users U, which is the superset of the disjoint subsets I_i of included users and O_i of To Be Included people. A_i is the set of possible interaction features at version i. E_i is a subset of I_i describing potential Enablers at version i. Each Enabler can contribute a subset of features belonging to A_i; in other words, the Enabler n can provide the contributions $C_{n,i} \subseteq I_i$ at version i.

The issue is selected at Step 2; its solution defined S at Step 3, affecting $T_I \subseteq O_i$ people. If there exists an Enabler E_u who can provide S (that is, $S \cap C_{u,i} \neq \emptyset, \forall u \in \{1, 2, ..., |E|\}$), and E_u contributes S, then, after conclusion of Step 6, V_{i+1} will have $I_{i+1} = I \cup T$ Included users and $O_{i+1} = O_i \setminus T$ To Be Included people[28]. At $i+1$, people

[26] The difference between Enablers and Enhancers is more theoretical than practical. An Enhancer is an Enabler whose inclusions contributions already exists in the system. The Enhancer role is not inferior to the Enabler. We defined this role for, although a system with only Enhancers can still improve, it will not include new people. Thus, theoretically, although there is a limit to the contributions of Enablers, Enhancers can keep improving the system indefinitely.

[27] For usability improvements, switch "Enablers" with "Enhancers", "create" with "improve", "add" with "update" or "modify".

[28] For simplicity, this assumes that every person needs a single interaction feature. Otherwise, each person would have a set of required interaction needs. After they were all satisfied, the person would

who could provide S became enhancers. Likewise, newly-Included people may become Enablers and Enhancers, depending on their abilities and skills[29].

Table 5 traces the collaboration dynamics of the C3-I instanced to the scenario described in Section D.4. It presents successive iterations of end-user collaborations[30]. It starts from the original system (V_0), proceeding to next versions as people contributed to it. Although not in the table, upon contribution, the role of a Enabler may change to Enhancer (if she/he cannot included new people once again).

D.6.1 Strategies to Foster End-Users as Enablers and Enhancers Collaborators

For inclusion, adaptation should be oriented to modify human-computer interaction. The first step is acknowledging that human senses define boundaries for any interaction. The five basic senses (vision, audition, olfaction, gustation, and somatosensation) are the "primitives" to define input and output strategies. Vision, audition, and somatosensation are, currently, the three main options for human interaction with digital systems[31]. Graphical, auditory, and haptic interfaces are examples of mediating human-computer interaction with each sense. For digital accessibility, sensory-wise improvements may happen in seven main ways (OBRENOVIC; ABASCAL; STARCEVIC, 2007; GARCIA, 2014; YUAN; FOLMER; HARRIS, 2011):

1. Replacing representation of media from one sense to another;
2. Simplifying representation of media to its essential information;
3. Enhancing representation of media to magnify its contents;
4. Reducing the universe of possible commands;
5. Automating part of the interaction;
6. Enabling the use of a new input device;
7. Adding/improving the compatibility with an assistive technology.

become Included. To support this mathematically, we could a new set representing required interaction features per person. A person would be Included when the system provided all interactive needs that she/he needed.

[29] Mathematically, Enablers who became Enhancers would remain Enhancers. As real people, however, people may learn or acquire new knowledge, which could they transform into Enablers once again. This is a positive deviation from the theory, and emphasizes the *transiency*: the role of a user may be altered whenever new (dis)abilities, knowledge, skills, and needs change. For completeness, a real person may also lose abilities (temporarily or permanently), which may revert her/his role back to yet To Be Included.

[30] Different orders or collaborations would result into other outcomes.

[31] Further alternatives may exist in the future. For instance, advances in brain-computer interfaces (BCI).

Table 5 – Collaborative model workflow for each contribution of the example presented in Section D.4.

System							**Student**													
#	**Features**						**Name**	**Needs**						**Can Provide**						**Role**
	Portuguese			**English**				**Portuguese**			**English**			**Portuguese**			**English**			
	A	🎙	📹	**A**	🎙	📹		**A**	🎙	📹	**A**	🎙	📹	**A**	🎙	📹	**A**	🎙	📹	
0		✓					Ana		✓						✓					Included
							João		✓					🧩		🧩				Included
							Isabela	🧩									🧩			To Be Included
							Pedro			🧩						🧩				To Be Included
							Linda				🧩						🧩			To Be Included
							James				🧩							🧩		To Be Included
							Robert					🧩							🧩	To Be Included
							Elizabeth						🧩						🧩	To Be Included
1	✓	✓					João		✓					✓		🧩				Enabler
							Isabela	✓									🧩			Included
2	✓	✓		✓			Isabela	✓	✓								✓			Enabler
							Linda				✓						🧩			Included
							James				✓							🧩		Included
3	✓	✓		✓			Linda				✓						✓			Enhancer
4	✓	✓		✓	✓		James				✓							✓		Enabler
							Robert					✓							🧩	Included
5	✓	✓	✓	✓	✓		João		✓					✓		✓				Enabler
							Pedro			✓						🧩				Included
6	✓	✓	✓	✓	✓	✓	Robert					✓							✓	Enabler
							Elizabeth						✓						🧩	Included
7	✓	✓	✓	✓	✓	✓	Pedro			✓						✓				Enhancer
8	✓	✓	✓	✓	✓	✓	Elizabeth						✓						✓	Enhancer

Source – Created by the author.

Note – ✓: Feature is part of the system; requirement fulfilled. 🧩: Feature is needed; someone can contribute it. **A**: Text transcription. 🎙: Audio speech. 📹: Sign language video. #: Version of the system (in text, $V_{\#}$); contribution number

The second step is noticing that messages can be decoupled from their representation in digital communication, and, thus, from their implementation. Concepts from Semiotics such as sign and signifier are helpful for this regard (BOUISSAC, 1998). Signs exist by themselves. They abstract conceptions and functions for things. Signifiers provide denotation for the content. They define a representation for a sign – for instance, icons and symbols represented by sounds, images, and words. Signifiers result from perception and interpretation of a sign, generating mental thoughts – connotation for content.

From Semiotics, a sign may have multiple signifiers representing it. For accessibility – and, especially, for our purposes –, decoupling messages from representations can be useful. If a very same message can have multiple media to convey it, we are able to define interaction alternatives to provide the same content. In this perspective, universal access for digital systems does not necessarily imply "one-size fits all" solutions. Rather, it can mean "adjustable solutions" composed of subsets of interaction features. It is, therefore, a tailoring approach to universal access. People define the way they interact with digital systems by selecting the most suitable alternatives to their needs.

For input, interaction alternatives assist people to command the system. Strategies include input reduction, automation, and scanning (YUAN; FOLMER; HARRIS, 2011). In this case, systems should define semantics of use instead of particular physical-level interactions and implementations – for instance a command to confirm instead of button to confirm. This enables input mapping/remapping to define multiple schemes for use (GREGORY, 2014).

For input, interaction alternatives are input devices and mappings. The system should not assume any particular devices for input; rather, it should provide application programming interfaces (APIs) that allows developers to introduce any device they want to communicate with the system. For output, interaction alternatives are media transformation. For the main interfaces, at a high-level, they are transformations of:

- Animation/Video to/from Audio;
- Animation/Video to/from Graphics;
- Animation/Video to/from Text;
- Animation/Video to/from Haptic Stimuli;
- Audio to/from Graphics;
- Audio to/from Text;
- Audio to/from Haptic Stimuli;

- Graphics to/from Text;
- Graphics to/from Haptic Stimuli;
- Text to/from Haptic Stimuli.

At a low-level, approaches define how to perform transformations. From Subsection D.2.2, we can note that sensory-wise improvements to accessibility in games (real-time interactive systems) related to content redundancy. If we define ways for end-users to create content redundancy based on their abilities, we enable them to co-create accessibility features for inclusion. Table 6 provides examples of possible approaches. The table is not exhaustive in any way; our goal is to provide examples of modifications that do not, necessarily, requires expert knowledge from end-users. Instead, we focus on daily life skills that can help others to overcome accessibility barriers.

Table 7 provides examples of possible companion meta-data for common digital media. Enablers and Enhancers are not, necessarily, digital systems experts; thus, anticipating alternatives provide guidance to suggest what they may include into the digital system. On the other hand, by anticipating possibilities, developers may implement features to adapt their system to suit the content it will need to reproduce. Hive combined with interdependence as a frame (SectionD.2.4) could contribute to identify more possibilities.

D.7 A Collaborative Work Model for Co-Creation of Universal Digital Games: The Collaborative Work Model Applied to Games

The C3-I from Section D.6 originated from our Collaborative Co-Creation of Inclusive Digital Games (C3-IDG), a pillar of our framework. The C3-IDG defines a game creation process to support creation and use[32]. It has four transient roles:

Supervisor Person (usually a domain expert in serious contexts) responsible for defining the goal of the creation. She/he guides (correct, suggest, recommend) Creators during the development of the project; requests, whenever needed, improvements from Collaborators; and shares the project to Players.

Creator Person who creates game content.

[32] Even though, most of the model is sufficiently generic to support other creation activities, with adjusts in terminology and goals.

Collaborator Person who can improve aesthetics and functional qualities of the game by providing a new resource or interaction alternative, or modifying an existing one. This is a combination of the Enabler and the Enhancer roles from the C3-I.

Player Person who plays the game, without modifying it (at use time).

The C3-IDG defines a process of creation, which is overseen by Supervisors. For inclusion, Supervisors track interaction problems to enable their intended audiences (for Players) to play projects (by Creators). This way, they play "Maria" in the scenario from Section D.4 – whenever a Creator cannot provide interaction alternatives and/or improvements on her/his own, a Supervisor can request aid from Collaborators. This way, Creators and Collaborators co-create inclusion – the community may provide what a single person cannot.

Game creation is the end practice of our framework. This means that development shortcuts are welcome instead of condemned; programming and technology in general are part of the process – not the process itself. The Elemental Tetrad (ET) from Schell (2008) provided a useful game design framework for our purposes, because it divided game design into four interconnected and equally important elements: mechanics, story, aesthetics, and technology. With four elements, we could provide different opportunities for collaboration. In turn, more opportunities resulted into greater possibilities to support diversity, because we could explore more approaches to suit heterogeneous abilities. In serious contexts, this had the added benefit to aiding domain experts who by any chance wanted to apply game creation as part of their activities.

D.7.1 Game Creation Process

To help end-users to create tailorable games as non-professional developers, we divided game creation into eight phases, which orchestrates when and how roles (Supervisor, Player, Collaborator, and Player) collaborated (Figure 32):

Table 6 – Possible conversions of interaction features.

Existing Feature	Ability Perception	Skills / Knowledge Perception	Contribution	Provided Feature	Potential Inclusion For
Audio in language (speech)	Audition	Language, speech	Language, writing	Subtitles	Hearing disabilities, Cognition (foreigners)
Audio in language (speech)	Audition	Language, speech	Sign language, speech	Sign language video	Hearing disabilities
Text in language	Cognition	Language, reading	Sign language, writing	Subtitles	Hearing disabilities
Text in language	Cognition	Language, reading	Language (another)	Subtitles (translation)	Cognition (foreigners)
Video	Vision, audition	Language, speech	Language, audio description	Audio Description	Vision disabilities
Video (with audio description)	Audition	Language, reading	Language, writing	Subtitles	Hearing disabilities, Cognition (foreigners)
Controls	Somatosens	Fine motor skills	Programmin;	Alternate control	Motor disabilities
Controls	Somatosens	Fine motor skills	Programmin;	Control (automation)	Motor disabilities, Cognitive disabilities
Sound effects	Audition	Language, speech	Language, writing	Text effects (onomatopeas)	Hearing disabilities, Vision disabilities

	Ability	Skills / Knowledge			
Existing Feature	Perception	Perception	Contribution	Provided Feature	Potential Inclusion For
Sound effects	Audition	Language, speech	Cognition, drawing	Visual effects	Hearing disabilities
Visual effects	Vision		Language, writing	Text effects (onomatopeas)	Hearing disabilities
Visual effects	Vision		Language, speech	Sound effects	Vision disabilities
Text effects (onomatopoeia)	Cognition	Language, reading	Language, speech	Sound effects	Vision disabilities
Text effects (onomatopoeia)	Cognition	Language, reading	Cognition, drawing	Visual effects	Hearing disabilities
Image (colors)	Vision	Fine motor skills	Cognition, drawing	Image (colors for color blindness)	Vision disabilities
Map (graphical)	Vision		Cognition, language	Sonar, GPS, Audio compass	Vision disabilities
Standard difficult	Cognition		Programmin	Difficult levels	Cognitive disabilities
Navigation	Somatosens	Fine motor skills	Programmin	Text (list of navigable areas)	Motor disabilities, Cognitive disabilities, Visual disabilities
Exploration	Somatosens	Fine motor skills	Programmin	Text (list of nearby entities)	Motor disabilities, Cognitive disabilities, Visual disabilities

	Ability	**Skills / Knowledge**			
Existing Feature	**Perception**	**Perception**	**Contribution**	**Provided Feature**	**Potential Inclusion For**
Simultaneous input	Somatosens	Fine motor skills	Programmin	Sequential input	Motor disabilities
Instructions	Cognition	Language	Language	Instructions (with finer details)	Cognition
Informative content	Cognition	Language, reading, hearing	Language, writing, speech	Simpler language	Cognition (literacy)
Informative content	Cognition	Language, reading, hearing,	Language, writing, speech	Summary of information	Cognition (literacy)
Sensory stimuli	Sensory skill	Ability	Programmin	Assistive technology support	Disabilities addressed by the technology
Text	Cognition	Language, reading	Programmin	Font for dyslexia	Cognition (dyslexia)

Source – Created by the author.

Table 7 – Examples of purpose and intent of use for common digital media, with companion meta-data to ease user contribution.

Media	Purpose	Context of Use	Meta-Data	Alternate Form
Text	Information	Label	Content	Image, Sound
		Description	Content	Image, Sound
		Subtitle	Subject, content	Image, Sound
		Instructions	Content	Image, Sound
Image	Information	Graph	Summary of data	Text, Sound (speech)
		Illustration	Description of illustration	Text, Sound (speech)
	Aesthetic	Static	Description	Text, Sound (speech)
		Animation	Description	Text, Sound (speech)
Sound	Information	Instructions	Description	Text, Image, Video (sign)
		Dialogue	Subject, content	Text (script)
	Aesthetic	Sound effect	Description, graphical effect	Text, Image
		Music	Lyric	Text, Video (sign)
Video	Information	Documentary	Description, subtitles	Text, Audio, Video (sign)
	Aesthetic	Music clip	Lyric	Text, Audio, Video (sign)

Source – Created by the author.

1. **Conception**. The starting phase is the conception of the game project. In this phase, Supervisors plan the project, defining its scope, goals, parameters, and the Creators who will participate. After they finish planning, Creators can start working on the project – transitioning the process to the Conversion phase.

2. **Conversion**. In the Conversion phase, Creators define the first prototype for the game project. As the first prototype is not the final game, its technology may differ from the final game's. Thus, the Conversion phase acts as a phase of "adaptation" for Creators, who may had never created games, programmed, or used TICs before.

   Creators who are tech savvy may opt to create their first prototype with the chosen technology for the final product – this results into a high-fidelity prototype, the first version of the project. Other Creators, however, may explore different technologies to express their ideas. For instance, text descriptions, drawings, and collages – or whatever material is suitable for the needs of the Creator – can materialize low-fidelity prototypes to explore ideas. With them, a Creator can build her/his game iteratively at future Creation phases. Alternatively, a Supervisor may request that a Collaborator implements an initial version of the game based on the low-fidelity prototype[33]. Instead of creating the technology from scratch, this enables Creators to work with an existing project first – a possible way to remove an initial accessibility barrier[34]. Regardless of the approach, the next phase is Evaluation.

[33] Therefore, although Creators are the main actors for the Conversion phase, Collaborators may aid them.

[34] In the future, the goal is to expand the Conversion phase to include noveau ways to create prototypes – for instance, with approaches similar to Pixel Press (<http://projectpixelpress.com>), Google Quick

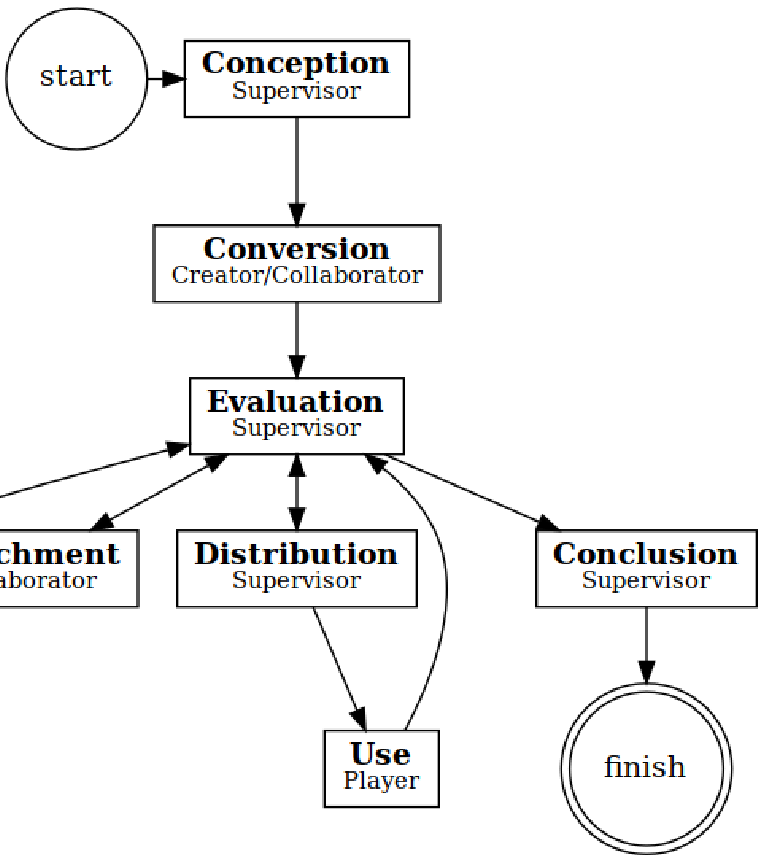


Figure 32 – Phases of the game creation process. Below the name of a phase, we list its main actor.

3. **Evaluation**. In the Evaluation phase, Supervisors analyze the current state of the project. Their goal is verifying whether the project fulfill their plans. If it does not, Supervisors plan the next step, providing feedback to Creators on what should improve, on how to succeed. This way, the Evaluation phase promotes share and transfer of knowledge – a master and apprentice metaphor, on which a Supervisor supports her/his apprentices with the creation. The game becomes a medium of communication for the apprentice. Consequently, it becomes means of counseling, reviewing, suggesting, teaching, and sharing for domain experts.

   Depending on the evaluation of Supervisors, the next phase is:

   - Creation, if they want Creators to act next. In this case, Supervisor suggests the next steps for Creators – for instance, improvements, problems and corrections, new scenarios for consideration.
   - Enrichment, if Collaborators should be the next actors. In this case, Supervisors want external improvements for the project; they can request new assets, features, or interaction alternatives for Collaborators to improve the project.

Draw! (<https://quickdraw.withgoogle.com>), GURI VR (<https://gurivr.com/>), and WordsEye (<http://www.wordseye.com/>)

- Distribution, if the next audience are Players. In this case, a play session comes next.
- Conclusion, should they consider that the project fulfilled its purpose – or they want to terminate it for any other reason.

4. **Creation**. The Creation phase is the main development phase for Creators. This is the moment that Creators iterate on the development of their game, considering comments from Supervisors to work towards the next version. Creation may encompass any combination of elements of the ET (mechanics, aesthetics, story, and/or technology) suitable to address the requests from the Supervisors, and to implement ideas into game content.

   For inclusion, Creators should provide as many interaction alternatives for created content as they can, according to their abilities. However, as the C3-IDG is for co-creation, it is not expected that Creators provide all; rather, Supervisors can request them to Collaborators at the next Evaluation phase. Likewise, if there were new assets and content alternatives provided by Collaborators at an earlier Enrichment phase, Creators add them to their games at this step.

   When a cycle of creation ends, it starts a new Evaluation phase.

5. **Enrichment**. When Supervisors notice that some features are missing or can be improved, they may request these features in a future Creation phase. However, Creators might not be able to fulfill their requests at times – they might be outside their abilities. In these cases, external Collaborators may help.

   Collaborators act as Enablers and/or Enhancers. If a Supervisor requested improvements for aesthetics or for existing features – to improve the gaming experience –, Collaborators are Enhancers. If a Supervisor requested features to enable access, they are Enablers. In this case, the goal of a Collaborator is to provide a feature to promote inclusion. Contributions can include, for instance, media transformations from Subsection D.6.1, alternatives for input (mappings, devices), support for assistive technologies, and/or even programming support (for end-users are not professional developers). Collaborators can be end-users, professional from various domains, participants from other projects – whoever the Supervisors can or want to reach and request.

   When a Collaborator finishes, she/he provides her/his work to the Supervisor. This starts a new Evaluation phase.

6. **Distribution**. At the Distribution phase, Supervisors assemble a game to suit the needs of their intended audience of Players. Interaction needs from the Players may differ from those of the Creators and Supervisors themselves – or even among

Players. Assuming that there are suitable interaction alternatives suitable for the needs of Players, the Supervisor selects them at the Distribution phase to generate games accessible to them. Distribution is the phase on which the "adjustable solutions" materialize via tailoring. Supervisors select desired alternatives to compose playable games matching the abilities of Players. If they do not exist yet, Supervisors should request them before proceeding; the process returns to the Evaluation phase. If they exist, the creation platform should combine the chosen interactions to generate the game[35] [36]. The process advances to the Use phase when the game is ready.

7. **Use**. If the Distribution phase is successful, Players can play games suitable to their interaction needs and abilities at the Use phase. Supervisors may participate in game sessions, if it suits their goals, to request feedback to Players, share projects among Creators, evaluate if interaction alternatives are working properly, promote new experiences, or any other purposes they consider relevant. When the session ends, a new Evaluation phase starts.

8. **Conclusion**. "Art is never finished, only abandoned". Da Vinci is right, for, although it is always possible to improve it, it must end (normally abandoned) sometime.

D.7.2 From Story to Mechanics to Programming: A Creation First Approach to End-User Game Development

We wanted to foster game creation from the start in our activities. Considering the ET, we chose story as the starting point, with goal of progressing towards technology (promoting computational thinking) over time. Our reasoning was that story-related activities were closer to daily experiences of end-users – especially of those who had never used computers before – than programming activities. With this choice, we selected the following activities as an approach for game-centered EUD:

1. Linear storytelling;
2. Branching storytelling;
3. Collaborative storytelling;
4. Storytelling with pre-defined mechanics;

[35] This is the role of our software architecture and creation platform. Another alternative would be generating a game with all available alternatives, and letting Players combine the ones they want at run-time.

[36] The metaphor of the jigsaw puzzle applies here again – we are placing the pieces to create a game.

5. Computational thinking patterns;

6. Creation of new mechanics.

Our inspiration were the phases of the Consume-Create Spectrum (Subsection D.2.3): animation, interactive simulation, collective simulation, construction set simulation, pattern-based authoring, end-user programming, and traditional programming. With a storytelling approach, we assumed that most people would not start clueless. People are used to consume non-interactive, story-centered media – including books, theater plays, movies, and soap operas – in daily life activities. This way, they would have a head start in the process. The consumption part of the model could happen at Use phases of the process – Creators (as Players) could play the games from each other, as well as examples that we provided exploring more techniques and possibilities.

To follow the "gentle slope" of the original Consume-Create Spectrum, we defined linear storytelling as the initial creation activity, as we argue it is similar to animation. Like an Animation, a linear story has a single plot flow; it always starts, progresses, and finishes the same way. Moreover, people can create and share stories in multiple ways – they can tell, write, draw, diagram, stage. For accessibility, thus, storytelling was a rich medium to define low, medium, or high-fidelity prototypes quickly. Game-wise, linear storytelling relates to simple visual novels.

Branching storytelling came next, as an approximation to interactive simulations. Non-linear, branching stories can define ramifications for the plot. Events, decisions, and choices can define how a story progresses; scores based on choices allows Creators to communicate how they evaluated a match from a Player. Programming-wise, we approach variables, conditional structures, and – for creative Creators – repetition commands (a branch may return to a previous one). Game-wise, we approached visual novels and interactive fiction games.

Collaborative storytelling follows. It approaches collective simulations when we consider that multiple Creators can co-create a story if each of them creates her/his own branches. Programming-wise, defining and merging branches acted as a primitive strategy of modularization.

Hitherto, games are interactive storytelling. Characters and dialogues were fundamental for the plot; places and objects were secondary, backgrounds to the story. From storytelling with pre-defined mechanics on, end-users start to dabble with game programming.

In storytelling with pre-defined mechanics, characters, places, and objects start becoming dynamic entities in game worlds. Instead of story players, games become simulations with agency due to game mechanics. Players Act to play, Creators add pre-defined game mechanics to convert entities into interactive elements with behaviors – it is our take on construction set simulation. Mechanics including dialogues, movement, character and object interaction, object picking and using add behaviors to entity. Events enable Creators to define how an entity reacts upon interaction. Programming-wise, this strategy approaches component based programming and event-driven programming. Game-wise, it approaches adventure and role-playing games.

Next comes computational thinking patterns, based on pattern-based authoring. From a combination of mechanics, there might emerge more complex interactions. This strategy includes more complex pre-defined mechanics, with pre-defined systems combining them. For instance, places, characters, objects and physics allow the creation of platform games[37]. Programming-wise, this strategy promotes computational thinking patterns and systemic design. Game-wise, it approaches action and platform games.

Finally, comes the creation of new mechanics, reaching the end-user programming phase of the Consume-Create Spectrum. At this point, scripting, macros, and visual programming languages come, moving towards more traditional programming practices. As these activities can be Turing-complete, there are not theoretic limits to what can be implemented; rather, the underlying creation platform imposes the limitations.

Although these mechanics may appear impossible for some audiences, people should not have to create and play exactly the same way. "One size fits all" solutions are limited; tailoring is not.

In this chapter, games are not video games; they are Meta-Games and Games. We should, thus, think differently – we design for no one to include everyone. Thus, we should think on ways to convey what is happening in the game world by ability; the Meta-Game has data, not human-IO content. The goal of the Game is to provide different ways to present the data and request input. All Games share the same rules (as they are the same Meta-Game); however, each of them may be played differently.

Do all games need to be played the same way by all players? We argue that they do not. If we design for adaptation, we can explore the best way to convey a mechanic for each ability.

D.7.3 Supporting The Other Pillars for Inclusive End-User Creation: Collaboration and the Lepi Game Platform

Interaction composability is not the goal of traditional software, for most systems are concerned with limited options for interaction. Consequently, current development approaches do not promote designing for interaction composability. To address both issues, we needed to support use and creation.

[37] Side scroller.

Lepi is an end-user game creation platform based on our architecture. It implements our run-time Tailoring algorithm, which allows run-time (re-)definition of user interaction. As an EUGD approach, Lepi enables Creators to define our approach to tailorable games without having to understand the underlying architecture and the theory behind it. Although games created with Lepi share the same logical rules, their interaction features may vary to support particular interaction needs of players. At this time, Lepi supports the three first strategies outlined in Subsection D.7.2 – it focus on storytelling based games[38].

From Subsection D.6.1, interaction alternatives are an important aspect for inclusion. When people co-create and provide alternatives, they can enable others to use and create. Lepi encourages this workflow. Output in Lepi uses Semiotics' signs as its primitive abstraction for content. As a sign, content may have multiple representations to convey the same message. Lepi express this concept to end-users as slots[39] – as proof of concept, we have text, audio, and sign language video slots. For instance, "Hello" is a textual representation of a message; it goes into the text slot. Similarly, a voice narration of the phrase goes into the audio slot; and a sign language interpretation can be attached to the video slots. Creators, thus, have three ways to insert game content into their project: writing to a text input field, attaching a recording of spoken content, or attaching a video of sigh language content. Together, these alternatives can generate seven possible output combinations to convey the story to Players (text; audio; sign; text and audio; text and sign; audio and sign; text, audio, and sign)[40] [41]. If a Player can perceive at least one of these combinations, she/he may perceive the output of game to play.

Input in Lepi explore semantics of use to define game commands. Instead of physical-level interactions, commands express actions abstracting Players' intents. For interactive storytelling, actions include "advancing the story", "selecting an option of a decision", "confirm/provide the selected option". With input mapping, people can assign any input mechanism (button, stick, axis, sensor...) of an input device to play. Accessibility-wise, this means that developers can provide implementations to support for any input devices and assistive technologies that they desire. Once an implementation is added to Lepi, Players can use its underlying input device to play. Moreover, as commands express semantics, developers can define macros and bots to help people with motor disabilities (instead of a mapping from an input device, the

[38] This will expand once we approach next phases with new mechanics.

[39] This reinforces the metaphor of game composition and inclusion as a jigsaw puzzle, on which people create and place pieces to improve accessibility. Each slot is a placeholder for pre-defined media alternatives.

[40] This is for a single language. Internationalization would multiply this number, as well as dialetic variations, and written forms of sign language.

[41] The total will not always be a combination, for some alternatives may clash with others. For instance, we could not support audio in two different languages at the same time. Likewise, complex game mechanics will result into more clashes in the future.

command comes from an algorithm). Lepi currently supports mouse, keyboard, and gamepads as input devices. As we explore new audiences, we could add more.

Lepi supports, primarily, the Creation, Distribution, and Use phases of the game creation process described Subsection D.7.1. For Creators who are tech savvy, had created games with the framework before, or wish to start from a high-fidelity prototype, Lepi can also be an approach for the Conversion phase. Likewise, due to its slots, Supervisors can monitor what interaction alternatives are missing, and, thus, request them at Enrichment phases. People whose abilities can fill a slot can create the desired alternative; afterwards, Creators can attach them to their games into a later Creation phase. Finally, at the Distribution phases, Evaluators can choose which slots will be part of a game generated by Lepi. The underlying implementation combines the chosen alternatives to present the game to the Players at the Use phase of the process.

D.8 Discussion

One small step from a user, a giant leap for universal access. This adaptation of Neil Armstrong's quote serves as a motto for this chapter, as end-users collaborate to improve accessibility in digital systems. People contribute based on their own abilities, knowledge, and skills. People fulfill their interaction needs from the kindness of others. From each, small and punctual improvements. To each, the possibility of combining improvements from community to choose how to interact with a system.

Even more important than providing digital inclusion, the mutualistic and communistic processes empowers end-users. Abilities become the means by which a end-user can assist her/his peers, whose interaction needs are potentially different that those of herself/himself. From the individual to the community. From the community to the individual. Every contribution helps to build universal access.

Inclusion should be part of software. If support for accessibility comes from within the software, it is easier to include people – with better usability and interaction quality. If we consider inclusion as a dynamic process, people can work together towards universal access. Together, they may overcome barriers that a single person could not alone. Co-creation supersedes stagnation, as additions enable people to use, and modifications improves the ease of use. If developers enable end-users to work together towards universal access, the community may provide accessibility solutions towards inclusion. Rather than external, assistive technology become part of the system as means of alternatives of use – for software is mutable, and, thus, inclusion can be iterative, incremental, and dynamic.

One approach to foster collaboration is defining work models for software creation – like we presented in this chapter – and build architectures and system to support and embrace them. As end-users are not developers, they need computational support to use, co-create, and share. Moreover, they need to know that their contributions may help others, and what they can provide. In the C3-IDG from Section D.7, we centralized decisions in the role of Supervisor and with abstractions in creation tools (slots in Lepi).

In the generic C3-I from Section D.6, other developers could define their protocols and mechanisms for their own software – potentially digital systems for other domains. To avoid "re-inventing the wheel", a better approach could be defining open formats to describe and store content enriched with interaction alternatives. With a standard, there could exist multiple tools for content creation. With multiple tools, end-users have choice to use the most suitable for their abilities and/or other preferences. In this way, we start supporting universal access for use and creation alike.

D.8.1 Limitations and Currently Unresolved Issues

The models presented in this chapter assume that end-users can modify interaction features of a system. This is not a traditional approach. As such, we can list the following limitations and unresolved issues:

- We are considering very small scale games in this study, with few mechanics. Our goal is to expand over time – by audiences, mechanics, and complexity.
- It is not realistic to expect every system to become universal – for games, in particular, it is often impossible (BARLET; SPOHN, 2012). However, every system can become more accessible and include a broader public.
- Implementation requires particular architectural choices, which are not present in state of practice software development kits. There is not a single user interface to suit everyone; rather, people compose their own interfaces by selecting existing interaction alternatives;
- Every human-interactive feature requires alternatives. This requires greater efforts than defining a single way for interaction, as it is now. People must collaborate to provide interaction alternatives for every new feature;
- There are, currently, few approaches to enable people with disabilities to create digital systems and game content, potentially limiting contributions;
- Our evaluation scenarios did not consider every possible disability and/or interaction barriers. Rather, they aimed to evaluate the C3-IDG. They were small tests of viability so far;
- The C3-IDG centralizes the orchestration of the process in the roles of Supervisors. For large case scenarios involving many participants, this might become

unfeasabile. In such cases, distributed approaches could be desirable[42]. An even better solution would be having a large scale Web system on which people could register their abilities and skills, so others could request improvements;

- Content enriched with media will be much larger, incurring costs for storage and distribution;
- Reuse of enriched media will probably be low for many domains – games, for instance;
- Alternatives involving human voice, image, and video may incur potential breaches of confidentiality, privacy, and anonymity. This brings the question of authoring and rights for redistribution;
- People may choose not to collaborate, even if they could;
- Whether it is ethical to explore community improvements to commercial products;
- Interaction alternatives and the architecture incur performance overheads;
- It is not realistic to expect every system to be universal, as it would depend on the participants;

Especially for games, our approach is very distant from the needs of the game industry[43] – at least at this time. We may list costs, performance, security, and even ethical and legal issues for commercial products as potential issues [44].

Especially for games, our C3-IDG is very distant from the needs of the game industry[45]. We focus on very small scale games. We may list costs, performance, security, and even ethical and legal issues for commercial products as potential issues. Although the architecture can help with some of these issues, the discussion fall out of scope of this chapter. The general idea is that the implementation strategies would allow developers to keep their traditional processes, while still enabling the community to modify the game for accessibility. Nevertheless, it incurs overheads, which is unsuitable for high-performance games – and it would need a large scale version of the C3-I.

[42] Hive exploring the C3-I, for instance, could be an alternative.
[43] The reader may refer to (PORTER; KIENTZ, 2013; PORTER, 2014)).
[44] Although the architecture can help with some of these issues, the discussion fall out of scope of this chapter. The general idea is that the implementation strategies would allow developers to keep their traditional processes, while still enabling the community to modify the game for accessibility. Nevertheless, it incurs overheads (which are unsuitable for high-performance games), modifications occur at run-time (restricting the potential for compile time optimizations and complicating efficient memory layouts), and it would require significant design, implementation, and evaluation efforts.
[45] The reader may refer to (PORTER; KIENTZ, 2013; PORTER, 2014)).

D.9 Concluding Remarks and Current Work

In this chapter, we described a generic collaborative work model to improve accessibility of digital systems by means of end-user collaboration (C3-I) and one for games (C3-IDG). The models explore what people are able to do – their abilities, capabilities, knowledge, and skills – to enable others to interact with a game. People are active; they create and improve. Their contributions become part of the system, helping people with similar needs. The process is communistic, for people contribute according to their abilities, and receive according to their needs, and mutualistic, as every improvement contributes with an overall net gain of interaction quality.

In special, people who were previously unable to interact with a system may become potential collaborators upon inclusion. They understand the difficulties from using inaccessible and unusable solutions; this knowledge makes them (non-professionals) experts who truly understand the struggles of using inaccessible systems. When they become able to use and improve a digital system, their knowledge and expertise may improve interaction quality for themselves and their peers.

For this, developer should empower people to transform inclusion into a dynamic, iterative, and collaborative process. When people have the means for collaborating, every small step from individuals results into potentially giant leaps of inclusion for the community. Each contribution enables more people to use a digital game. Interaction alternatives allow users to customize game interaction to suit their needs and abilities. More alternatives allow for more combinations, suiting a broader range of interaction needs. In this way, every contribution makes systems more inclusive, progressing towards universal access.

The C3-IDG originated from our ongoing work on developing a framework to support tailorable end-user creation of tailorable games. Our motto is the ideal scenario of "games *by* everyone, *for* everyone". The model, a software architecture and a game creation platform form the three pillars of our framework. With the architecture, games and creation platforms allow users to tailor all input and output at run-time to define human-computer interaction suitable for their needs. End-users create for fun, learning, and to contribute with digital and social inclusion. We are currently performing participatory activities on which end-users (with different (dis)abilities) create games for themselves and people with, potentially, different needs than theirs. We have started with people with low literacy and hearing disabilities; we plan to expand our audiences to include other interaction needs. Furthermore, our plan is to follow the activities in our game-centered EUD activities as a road map to introduce complex game mechanics over time.