DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

Um Motor para Jogos Digitais Universais

ALUNO: Franco Eusébio Garcia
ORIENTADORA: Prof.ª Dr.ª Vânia Paula de Almeida Neris

São Carlos
Maio/2014

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MOTOR PARA JOGOS DIGITAIS UNIVERSAIS

FRANCO EUSÉBIO GARCIA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software, Banco de Dados e Interação Humano-Computador.
Orientadora: Prof.ª Dr.ª Vânia Paula de Almeida Neris

São Carlos - SP
Maio/2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

G216mj

Garcia, Franco Eusébio.
Um motor para jogos digitais universais / Franco Eusébio Garcia. -- São Carlos : UFSCar, 2014.
166 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Engenharia de software. 2. Jogos por computador. 3. Motores de Jogos. 4. Design universal. 5. Acessibilidade. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Um Motor para Jogos Digitais Universais"

Franco Eusébio Garcia

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Profa. Dra. Vânia Paula de Almeida Neris
(Orientadora - DC/UFSCar)

Profa. Dra. Renata Pontin de Mattos Fortes
(ICMC/USP)

Prof. Dr. Flávio Soares Correa da Silva
(IME/USP)

São Carlos
Maio/2014

DEDICATÓRIA

Trabalho dedicado aos meus pais, à minha irmã e a toda minha família pelo amor, a união e o apoio incondicional durante toda minha vida.

AGRADECIMENTOS

Este trabalho foi realizado graças ao esforço, à atenção, à dedicação e aos ensinamentos oferecidos pela família, pelos amigos e pelas pessoas ao longo de minha vida. Assim, agradeço, sem ordem específica e evitando-se nomeações, por questão de espaço:

À minha família por todo o apoio e incentivo em todos os momentos de carreira acadêmica. Às amigas e aos amigos por todos os momentos e contribuições.

Às professoras, aos professores e às inúmeras pessoas que dispendem de seu tempo seja para o ensino, seja para a elaboração de material didático ou paradidático e que, infelizmente, nem sempre são reconhecidos pela importância de seus esforços e trabalhos.

Às instituições de ensino e de divulgação de conhecimento, por oferecerem o espaço para o ensino e por promover conhecimento. Agradeço também aos funcionários destes locais pelos serviços e auxílios prestados aos alunos e à comunidade.

À professora Vânia, pela orientação, pela atenção, pelas inúmeras sugestões e contribuição a este trabalho. Às amigas e aos amigos do LIFES, pelo apoio, sugestões e críticas para a melhoria deste trabalho. Às alunas (Alessandra e Flávia) e aos alunos (Eduardo, Marcos, Lucas, Pedro e Vinícius) de iniciação científica pela confiança depositada em mim e pela exemplar dedicação aos projetos e ao aprendizado.

A todas as pessoas que direta ou indiretamente contribuíram para a realização deste trabalho. Ao *International Game Developers Association* (IGDA) *Game Accessibility Special Group* pelo apoio, recomendações e comentários providos – em especial, para Thomas Westin, Steve Spohn e Ian Hamilton.

À FAPESP (2012/22539-6) pelo apoio financeiro e institucional provido a este projeto.

Ao leitor, pelo interesse acerca da temática e, possivelmente, pelo interesse em desenvolver soluções mais acessíveis.

"Imagination is more important than knowledge”

(Imaginação é mais importante que conhecimento)

-- Albert Einstein

“Sócrates – Agora imagina a maneira como segue o estado da nossa natureza relativamente à instrução e à ignorância. Imagina homens numa morada subterrânea, em forma de caverna, com uma entrada aberta à luz; esses homens estão aí desde a infância, de pernas e pescoços acorrentados, de modo que não podem mexer-se nem ver senão o que está diante deles, pois as correntes os impedem de voltar a cabeça; a luz chega-lhes de uma fogueira acesa numa colina que se ergue por detrás deles; entre o fogo e os prisioneiros passa uma estrada ascendente. Imagina que ao longo dessa estrada está construído um pequeno muro, semelhante às divisórias que os apresentadores de títeres armam diante de si e por cima das quais exibem as suas maravilhas.

(...)

Sócrates - Agora, meu caro Glauco, é preciso aplicar, ponto por ponto, esta imagem ao que dissemos atrás e comparar o mundo que nos cerca com a vida da prisão na caverna, e a luz do fogo que a ilumina com a força do Sol. Quanto à subida à região superior e à contemplação dos seus objetos, se a considerares como a ascensão da alma para a mansão inteligível, não te enganarás quanto à minha idéia, visto que também tu desejas conhecê-la. Só Zeus sabe se ela é verdadeira. Quanto a mim, a minha opinião é esta: no mundo inteligível, a idéia do bem é a última a ser apreendida, e com dificuldade, mas não se pode apreendê-la sem concluir que ela é a causa de tudo o que de reto e belo existe em todas as coisas; no mundo visível, ela engendrou a luz; no mundo inteligível, é ela que é soberana e dispensa a verdade e a inteligência; e é preciso vê-la para se comportar com sabedoria na vida particular e na vida pública.”

Platão (A República - Mito da Caverna)

RESUMO

Jogos digitais são cada vez mais utilizados para o entretenimento, lazer e como ferramenta para a educação. No entanto, não se observa um aumento de acessibilidade correspondente ao aumento do uso e da importância destes sistemas de software. O Design Universal apresenta-se como alternativa para o desenvolvimento de soluções usáveis e acessíveis ao maior número possível de pessoas, independentemente de suas capacidades físicas, cognitivas e emocionais. Em jogos, o *Unified Design*, é um processo que norteia a realização do design de um jogo digital universal. Entretanto, a implementação do design obtido é deixada a cargo do desenvolvedor. *Game engines* (motores de jogos) são sistemas de software concebidos para facilitar o desenvolvimento e a implementação de jogos digitais. Neste sentido, este trabalho apresenta um motor para jogos universais, denominada *UA-Game Engine* (UGE) com o objetivo de facilitar a desenvolvedores criarem jogos universais. Para isto, o motor explora arquiteturas *data-driven*, *event-driven* e sistemas entidade-componente para promover o desenvolvimento de jogos flexíveis e adaptáveis em tempo de execução. São introduzidos os conceitos de Mundo Abstrato de Jogo, Mundo Concreto de Jogo, Meta-Jogo e Jogo. Para isto, este trabalho demonstra, informalmente, que é possível construir um jogo universal decompondo-se um mundo de jogo qualquer de forma a torna-lo independente de entradas e saídas por meio de três elementos: entidade (ator), componente e evento. Estes elementos são usados para se obter um Mundo Abstrato de Jogo – resultando-se em um Meta-Jogo livre de entradas e saídas. Em seguida, sugere-se que é possível combinar estes três elementos para se reconstruir um ou mais Mundos Concretos de Jogo a partir de um Mundo Abstrato de Jogo de acordo com necessidades de interação específicas de usuários. Com base neste resultado, desenvolveu-se o motor UGE que utiliza um perfil de usuário extensível e flexível para, em tempo de execução, adaptar o Meta-Jogo em um Jogo acessível, definindo-se todas as entradas e saídas necessárias para adequar às interações às necessidades do usuário. Ao final, descreve-se o processo de avaliação e validação do motor, atualmente em progresso – os resultados obtidos até o momento classificam o projeto como promissor.

Palavras-chave: jogo; *engine*; motor; design universal; acessibilidade; uge.

ABSTRACT

Digital games are being played gradually more for entertainment, leisure and as a tool for education. However, one does not observe an increase in gaming accessibility corresponding to the ascending importance and use of these software systems. Universal Design provides an alternative approach to design more usable and accessible solutions to as many people as possible, regardless of their physical, cognitive and emotional capabilities. The Universal Design for games is in its early stages: so far, only a few universal titles have been created. Those few games were designed following the Unified Design, a process which leads the game design to a more abstract and modality independent way. However, albeit aiding designing the game, the implementation is left to the developer. Game engines are software created to ease the development and the implementation of digital games. Hence, this work describes a game engine for universal games *UA-Game Engine* (UGE), which purpose is to ease the development of universal games. UGE explores data-driven, event-driven and entity component systems to ease the development of run-time flexible and adaptable games. This dissertation introduces the concepts of Abstract Game World, Concrete Game World, Meta-Game and Game. To achieve this result, this dissertation informally demonstrates that is it possible to construct a universal game by decomposing any game world into three elements: entity (actor), component and event. These elements are used to create an Abstract Game World without a user – resulting into an input-output free Meta-Game. Afterwards, it is suggested that it possible to combine the three elements to reconstruct one or more Concrete Game Worlds from the Abstract Game World to suit the interaction needs of the users. With this result, the UGE game engine was developed. UGE uses an extensible and flexible user profile to, during run-time, tailor the Meta-Game into an accessible Game, by defining all the input and output interactions according to the user's interaction needs. Finally, it is described the evaluation and validation process, currently in progress – the obtained results so far classify the project as promising.

Keywords: game; *engine*; universal design; accessibility; uge.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGO-FONTE

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

Abreviatura / Sigla	Significado
AAA	*Triple-A Game*
AI	*Artificial Intelligence*
API	*Application Programming Interface*
COTS	*Commercial Off-the-Shelf*
ES	Entrada e Saída
GCC4	*Game Coding Complete* 4
HUD	*Head-Up Display*
IA	Inteligência Artificial
IGDA	*International Game Developers Association*
IO	*Input-Output*
NPC	*Non-Playable Character*
OGRE	*Object-Oriented Graphics Rendering Engine*
OIS	Object Oriented Input System
SDK	Software *Development Toolkit*
UA-Games	*Universally Accessible Games*
UGE	*UA-Game Engine*
XML	*Extensible Markup Language*

LISTA DE DEFINIÇÕES

Mundo de Jogo

O mundo de um jogo digital qualquer. Inclui as regras, mecânicas, entidades (atores) e o jogador. A simulação é regida por entradas controladas pelo usuário. A saída é feita por meio de estímulos sensoriais.

Mundo Abstrato de Jogo

O Mundo Abstrato de Jogo é o mundo de jogo resultante da remoção de entradas e saídas relativas ao usuário – é a redução de um jogo qualquer a uma simulação. Toda a simulação de um jogo envolve apenas um conjunto finito de elementos primitivos: entidades (atores), componentes e eventos. Toda as demais partes de um jogo emergem da cominação destes elementos primitivos. A comunicação entre atores ocorre pode meio de eventos. O resultado de regras gera um ou mais eventos.

Mundo Concreto de Jogo

O Mundo Concreto de Jogo é o mundo de jogo resultante da adição de entradas e saídas relativas ao usuário ao Mundo Abstrato de Jogo. Desta forma, reintroduz-se o usuário à simulação, restaurando-a ao estado de Jogo.

Meta-Jogo

O Meta-Jogo é o molde para a criação de jogos com os mesmos elementos e regras obtidos pela simulação de um Mundo Abstrato de Jogo. O Meta-Jogo é como um *template* para a criação de clones de um mesmo jogo. Por ser isento de entradas e saídas, ele pode ser especializado para permitir quaisquer interações que se deseje. Esta especialização ocorre sem afetar o conjunto de regras do jogo.

Jogo

É o resultado da especialização do Meta-Jogo – o artefato de software provido de entradas e saídas com o qual o usuário interage. Ele é, portanto, a especialização do Meta-Jogo considerando-se às necessidades de interação de um usuário.

SUMÁRIO

Capítulo 1

INTRODUÇÃO

"Play is the work of the child"

(Jogar é o trabalho da criança)

Maria Montessori

1.1 Contexto e Motivação

Jogos digitais (doravante jogos) são sistemas de software interativos usados para lazer, entretenimento e, cada vez mais, como ferramenta para a educação (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; YUAN; FOLMER; HARRIS, 2011). Embora a importância e a utilização de jogos continuem a aumentar, a acessibilidade ainda não é uma prioridade no desenvolvimento desses sistemas (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2004; MCCRINDLE; SYMONS, 2000; YUAN; FOLMER; HARRIS, 2011).

O acesso participativo e universal é considerado um dos cinco grandes desafios da área de Computação no Brasil[1] (CARVALHO et al., 2006). Neste contexto, como possibilitar que pessoas com diferentes capacidades, habilidades e necessidades de interação possam utilizar um mesmo jogo?

[1] Mais especificamente, é o 4º Desafio da Sociedade Brasileira de Computação: acesso participativo e universal do cidadão brasileiro ao conhecimento (CARVALHO et al., 2006).

A interação com jogos está sujeita a diversas barreiras de acessibilidade, incluindo sensoriais, mentais, motoras, culturais e de idioma (DENIS; JOUVELOT, 2004; GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009; YUAN; FOLMER; HARRIS, 2011). O uso adequado de um jogo, portanto, pressupõe diversas habilidades e capacidades do jogador para a interação. Considerando-se esta informação, seria uma mesma forma de interação com um jogo adequada a todos os possíveis usuários, indistintamente de habilidades ou capacidades individuais?

A Figura 1 ilustra que a comunicação entre jogador[2] e jogo é mediada por diversos dispositivos e interfaces – tanto para entrada, quanto para saída. A interação costuma estar diretamente relacionada ao design, à implementação e ao *gameplay*[3] do jogo: busca-se transmitir ao jogador o mundo criado para o jogo da melhor forma possível.

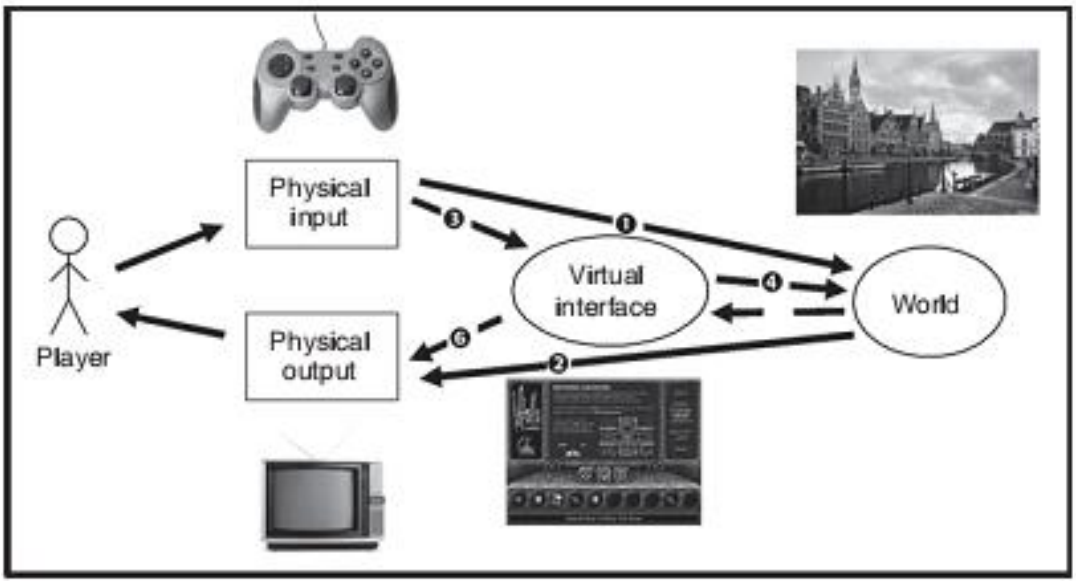


Figura 1. Interfaces e dispositivos mediam a comunicação entre usuário e jogo. (Extraído de **(SCHELL, 2008)**)

Pode-se observar que, atualmente, a maioria dos jogos utilizam os mesmos dispositivos para a interação com os jogos: por exemplo, os estímulos utilizados são predominantemente visuais – de fato, muitos jogos não podem ser utilizados sem gráficos. Da mesma forma, as interações de entrada costumam ser mediadas pelos mesmos dispositivos, como controles e teclados.

[2] Os termos jogador e usuário serão considerados sinônimos ao longo do texto. Salvo menção contrária, ambos os termos referem-se à pessoa que joga um jogo digital.

[3] *Gameplay* pode ser entendido como a forma com que um usuário interage com um jogo digital. O *gameplay* também está relacionado à experiência de jogo, sendo definido pelo conjunto de regras e mecânicas definidas pelos designers para se atingir os objetivos do jogo.

A interação decorrente do uso destes dispositivos tende a ser adequada a usuários médios[4] – àqueles que se enquadram no desvio-padrão de uma distribuição normal de usuários (FISCHER, 2007; NERIS, 2010). Entretanto, ela não beneficia a todos os possíveis usuários.

Para ilustrar esta situação, pode-se considerar um jogo de puzzle no qual o jogador deva agrupar imagens de mesma cor. Para este jogo, como o *gameplay* é dependente da representação gráfica, a percepção do jogo é prejudicada (ou mesmo impossibilitada) caso o usuário afaste-se do modelo de usuário médio.

A Figura 2 ilustra esta situação, apresentando um esquema de como usuários com diferentes capacidades visuais percebem sensorialmente um mesmo jogo. Nesta figura, apenas o usuário médio (1) consegue perceber sensorialmente toda a interface do jogo. Os demais usuários, representados nas imagens restantes (2 – 5), têm a percepção prejudicada devido a alguma deficiência visual.

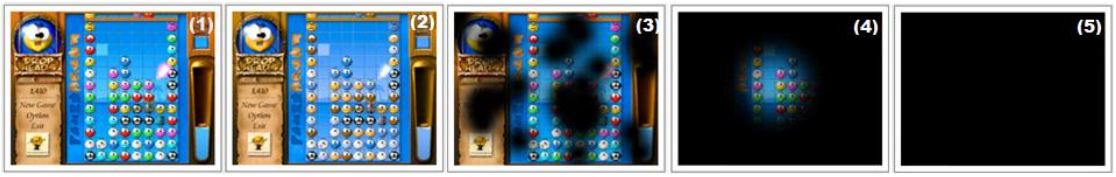


Figura 2. Representação de como jogadores com diferentes deficiências visuais visualizam a mesma cena de um jogo. Em (1), visualização de um usuário médio – todos os detalhes e cores podem ser depreendidos. Em (2), visualização de um jogador com daltonismo – cores são perdidas e a resolução, impossibilitada. Em (3) e em (4), visualização de jogadores com diferentes graus de visão reduzida – áreas menores são visualizadas. A imagem (5) revela a ausência da visualização por um usuário cego. (Adaptada de *Game Acessibility*[5])

No caso da população brasileira[6], por exemplo, mais de 45 milhões de pessoas têm algum tipo de deficiência (IBGE, 2012). O número de pessoas com algumas das deficiências mais comuns é apresentado na Tabela 1.

As capacidades e necessidades de interação de pessoas com alguma deficiência tendem a ser distintas daquelas de um usuário médio. Por exemplo, para compreender e utilizar um jogo, o jogador deve perceber os estímulos sensoriais providos por sua interface.

---

[4] Para jogos, pode-se assumir um modelo de usuário médio típico como uma pessoa alfabetizada, sem deficiências visuais, auditivas, mentais ou motoras. Embora o número de jogos com suporte a diferentes idiomas aumentara nos últimos anos, o inglês segue como idioma padrão para a maioria dos jogos. Assim, pode-se considerar um usuário médio para jogos típico como fluente (ou, no mínimo, proficiente) em inglês.

[5] Disponível em: <http://www.game-accessibility.com/index.php?pagefile=visual>

[6] As vastas diferenças sociais, culturais e de acesso à tecnologia e ao conhecimento da população brasileira podem ser observadas em estudos como os de (FILGUEIRAS et al., 2005) e (NERIS, 2010), que retratam diferentes necessidades e capacidades de interação da população brasileira.

Tabela 1. Número de pessoas com algum tipo de deficiência visual, auditiva, motora, mental e/ou intelectual. Extraído de **(IBGE, 2012)**.

Tipo de Deficiência	Não consegue de modo algum	Possui grande dificuldade	Alguma dificuldade	Total
Visual	506.377	6.056.533	29.211.482	35.774.392
Auditiva	344.206	1.798.967	7.574.145	9.717.318
Motora	734.421	3.668.929	8.832.249	13.235.599
Mental e/ou intelectual	-			2.611.536

Yuan, Folmer e Harris (2011) descrevem um modelo cíclico divido em passos que descrevem a interação de usuários com jogos (Figura 3).

Figura 3. Modelo de interação simplificado para jogos. Os passos são relativos ao jogador. (Adaptado de (YUAN; FOLMER; HARRIS, 2011))

No modelo ilustrado na Figura 3, o jogador recebe um estímulo da interface de usuário (por exemplo, uma representação sensorial do mundo do jogo ou um feedback decorrente de uma ação do jogador). A compreensão do estímulo recebido é necessária para a determinação da resposta mais adequada à situação de jogo. O jogador deve prover esta resposta por meio de entrada adequada para continuar a jogar.

Para se prover a um usuário uma interação adequada com o jogo, faz-se necessário satisfazer todos os passos do modelo de interação – isto é, permitir ao usuário receber o estímulo, determinar a resposta e realizar a entrada. Uma possível opção envolveria o uso de tecnologias assistivas. Entretanto, o uso de tecnologias assistivas costuma ser insuficiente ou inadequado para a interação com jogos (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009). Muitos jogos não são compatíveis com tecnologias assistivas e, em geral, aqueles que são compatíveis costumam

possuir baixa qualidade de interação e de usabilidade (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009).

Outra opção seria permitir ao jogo adaptar-se às habilidades e capacidades de interação do usuário – por exemplo, possibilitando-se aos designers moldarem a interação com o jogo de forma a atenderem às necessidades do usuário, poder-se-ia obter um jogo mais acessível. Neste contexto, o Design Universal (ou Design para Todos) (STORY; MUELLER; MACE, 1998) apresenta-se como uma alternativa para um design mais inclusivo.

O Design Universal baseia-se no desenvolvimento de soluções que possam ser utilizadas pelo maior número possível de pessoas, considerando-se suas capacidades sensoriais, físicas, cognitivas e emocionais para o desenvolvimento de uma solução mais inclusiva[7] (NERIS, 2010; STEPHANIDIS, 2001; STORY; MUELLER; MACE, 1998). Uma solução universal deve ser o mais inclusiva possível e, quando necessário, se adaptar às necessidades dos usuários. Assim, ao invés de adotar um modelo particular e limitado de usuário, o Design Universal incorpora o design para a diversidade em seus princípios.

O Design Universal de jogos, por sua vez, apresenta-se em fase incipiente. Poucos jogos universais foram criados até o momento: a *survey* sobre acessibilidade em jogos realizada por Yuan *et al*. (2011), por exemplo, destaca apenas três títulos como jogos universais: os *Universally Accessible Games* (UA-Games) (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007, 2009, 2011), jogos baseados nos princípios do Design Universal.

O design destes jogos baseou-se no *Unified Design* (Design Unificado, doravante), um framework descrito por Grammenos, Savidis e Stephanidis (2007, 2009). O objetivo do framework é permitir que a realização do design do jogo independa de seu contexto de uso; para isto, o design feito em um nível abstrato, com o mínimo possível de referências a interações, dispositivos e técnicas que empreguem uma modalidade específica (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007). Com isto, o design resultante pode ser especializado para um usuário ou para um grupo de usuários (constituindo um perfil de usuário).

[7] É importante notar que a definição utiliza a expressão "maior número possível de pessoas" – ou seja, não se exige que a solução seja acessível a todas as pessoas. Uma solução realmente universal tende a ser impossível de ser obtida no caso de jogos digitais (BARLET; SPOHN, 2012).

Contudo, embora o Design Unificado descreva como realizar o design do jogo, a implementação é deixada a cargo do desenvolvedor – com todos os seus pormenores e complexidade. A implementação de um jogo pode ser apoiada como ferramentas de alto-nível para a criação de conteúdo, como editores de mundo, e por plataformas de desenvolvimento, como *game engines* (motores para jogos) (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012).

Motores extensíveis podem fornecer uma base para o desenvolvimento de novos jogos (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012). Motores fornecem uma plataforma de software base para que os desenvolvedores possam focar em aspectos específicos da implementação de seus jogos ao invés do desenvolvimento de toda a tecnologia envolvida.

Caso projetado para permitir diferentes interações para um mesmo "modelo" (considerado por este trabalho como lógica do jogo, definida por regras e mecânicas), um motor poderia permitir aos designers especializarem a interação de forma a adequá-la ao usuário – contribuindo-se, assim, para contribuir com o desafio do acesso participativo e universal.

1.2 Objetivo

Observando-se a falta de ferramentas de alto-nível especializadas para a construção de jogos acessíveis, este trabalho buscou apoiar desenvolvedores na construção de jogos digitais universais por meio de uma base para o desenvolvimento: um motor para jogos. Para isto, fez-se necessário combinar diferentes técnicas e abordagens de implementação em uma arquitetura que permitisse a uma aplicação alterar a interação em tempo de execução para a adequar às necessidades de interação de diferentes usuários[8].

Como resultado, apresenta-se, nesta dissertação, o motor *UA-Game Engine*[9] (doravante UGE) (GARCIA, 2014c). O motor UGE separa toda a implementação da

[8] Este tipo de adequação é denominado *tailoring* (KAHLER et al., 2000 apud NERIS, 2010). *Tailoring* permite alterações da aplicação de acordo com seu contexto e em tempo de uso, permitindo adequar um sistema de software para diferentes contextos e necessidades de uso de usuário. O termo *tailoring* provem do termo *tailor* (alfaiate), ou seja, é uma metáfora sobre àquilo que é feito sobre medida – tem "bom caimento" – para o usuário.
[9] UGE. https://github.com/francogarcia/uge

lógica de um jogo de quaisquer referências a interações de nível físico específicas, permitindo a construção de Meta-Jogos – *templates* para a construção de jogos com as mesmas regras.

Assim, pode-se dizer que UGE é para a prototipação e para a implementação o que o Design Unificado é para o design: uma forma de criar jogos totalmente independentes de interações de nível físico específicas. Isto permite que, em tempo de execução, sejam definidas as entradas e saídas (ES) desejadas para o jogo – ou seja, transformar-se um Meta-Jogo em um Jogo com quaisquer ES que se desejar. Em potencial, isto permite aos designers definirem ES adequadas às capacidades de interação do usuário para o desenvolvimento de jogos acessíveis.

Devido à sua arquitetura, o motor UGE atua como uma plataforma de software que contribui com a estruturação, o controle e a adaptação em tempo de execução de Meta-Jogos para Jogos, permitindo aos desenvolvedores transformarem suas ideias e designs em jogos.

Para isto, o motor UGE:

I. Fornece uma base para o desenvolvimento de novos jogos acessíveis;

II. Possibilita, em tempo de execução, a especialização dos estímulos utilizados para entrada e saída (ES) para a interação com o jogo;

III. Auxilia, quando possível, na apresentação de conteúdo convertido de uma modalidade específica para outra determinada;

IV. Permite que o modelo e a lógica do jogo sejam reusados o máximo possível, focando alterações, na medida do possível, apenas na interação;

V. Permite que a interação seja reusada o máximo possível, caso assim os designers desejarem.

Essas características promovem funcionalidades que permitem aos desenvolvedores focarem, iterativamente, em melhorias para o design e o *gameplay* do jogo. Ademais, a separação entre modelo e lógica do jogo da apresentação possibilita a criação futura de ferramentas de criação de jogos, como editores de mundo (GREGORY, 2009) para a criação de jogos acessíveis e/ou universais.

1.3 Abordagem de Pesquisa

O projeto consistiu de um contínuo levantamento bibliográfico e estudo da literatura acerca de temas relacionados, como design e implementação de jogos digitais, acessibilidade e design universal; da proposta e implementação do motor; da realização de estudos de caso e de avaliações; e da divulgação dos resultados obtidos (GARCIA; NERIS, 2013a, 2013b, 2014; GARCIA, 2014c).

A implementação do motor foi feita em linguagem C++. Para facilitar a prototipação de jogos e aumentar a flexibilidade do motor, também é possível utilizar a linguagem Lua[10] para a implementação da lógica do jogo.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte forma: no Capítulo 2, apresenta-se uma síntese do levantamento bibliográfico. No Capítulo 3 e no Capítulo 4, descreve-se, respectivamente, a construção do Mundo Abstrato de Jogo e do Mundo Concreto de Jogo, que servem como base lógica para UGE. Estes mundos são compostos por combinações de apenas três elementos primitivos de jogo (entidade ou ator, componente e evento), de cujas relações derivam toda a lógica de um jogo.

No Capítulo 5, apresenta-se o motor UGE, discutindo-se como transformar Meta-Jogos em Jogos em tempo de execução. No Capítulo 6, apresenta-se o uso do perfil de usuário, o recurso mais poderoso do motor UGE para a realização de *tailoring* (KAHLER et al., 2000 apud NERIS, 2010) em tempo de execução.

No Capítulo 7, apresenta-se algumas avaliações realizadas para o motor. Por fim, no Capítulo 8, apresenta-se as conclusões e trabalhos futuros decorrentes da realização da pesquisa.

[10] Lua é uma linguagem de *scripting* interpretada, rápida, leve e embutível que pode ser utilizada para estender aplicações (PUC RIO, 2013). Em jogos digitais, Lua é frequentemente utilizada para prototipação, para permitir a criação de modificações no jogo (*mods*) e para criação de conteúdo de *gameplay*.

Capítulo 2

SÍNTESE DO LEVANTAMENTO BIBLIOGRÁFICO

"If I have seen further it is by standing on the shoulders of Giants"

(Se eu pude ver mais longe, foi por estar sobre ombros de gigantes[11])

Isaac Newton

2.1 Considerações Iniciais

Neste capítulo, apresenta-se brevemente os principais tópicos de interesse para a realização deste trabalho. Iniciar-se-á pela descrição de um modelo de interação para jogos, que será explorado ao longo dos próximos capítulos.

Em seguida, apresenta-se conceitos relacionados ao design de jogos acessíveis e universais, com uma breve descrição do Design Universal e de conceitos relevantes, como *tailoring*. Será apresentado o Design Unificado, um processo para o design de jogos universais.

Por fim, apresentam-se motores para jogos e técnicas de arquitetura exploradas por este trabalho. As técnicas exploradas serão utilizadas para a construção do motor proposto neste trabalho, conforme descrito ao longo dos próximos capítulos.

[11] Em um contexto de desenvolvimento de software, talvez o ideal seria adaptar a frase para: "se pude ver mais longe, foi por construir sobre rodas de gigantes".

2.2 Modelo de Interação para Jogos Digitais

Durante o *gameplay*, o designer deseja fornecer uma experiência interativa e imersiva ao jogador. A apresentação do jogo, o significado de seus elementos e a forma do jogador interagir costumam estar relacionados a essa experiência – muitas vezes, eles podem ser explorados de forma a contribuir para um aumento da imersão.

Devido a diferenças entre a percepção e a interpretação de um mesmo estímulo por diferentes sentidos, nem todo elemento de interface de um jogo pode ser apresentado por modalidades distintas daquela para a qual foi concebido. Por exemplo, o uso da técnica *drag and drop* dificilmente seria convertido para uma modalidade não visual sem perdas para a interação.

Estas diferenças vêm sendo estudadas por trabalhos com ênfase em acessibilidade em jogos. Trabalhos como os de Ellis *et al.* (2013), *International Game Developers Association* (IGDA) (2004), Ossman e Miesenberger (2006) e UPS Project (2004, 2011) fornecem recomendações para guiar desenvolvedores no design de jogos mais acessíveis. Um design acessível às pessoas com diferentes capacidades e necessidades de interação procura minimizar referências a interações ou modalidades específicas, buscando-se permitir a realização de um mesmo objetivo de várias formas distintas.

O uso de modelos constitui uma possível forma para se analisar a interação entre usuário e sistema. Como mencionado na Seção 1.1 e ilustrado na Figura 3, Yuan *et al.* (2011) propuseram um modelo genérico de interação para jogos, dividido em três passos relativos ao usuário:

1. Recebimento de estímulo;
2. Determinação da resposta;
3. Realização de entrada.

Como ilustrado na Figura 4[12], os três passos são realizados continuamente pelo jogador, desde o início até o final de um jogo digital qualquer. Para o jogador, um estímulo é responsável pelo início do jogo. Após o estímulo inicial, o jogo

[12] Na Figura 4 (a), é necessário observar que o tempo é algo que deve ser considerado quando são estudados jogos em tempo real (b). Um jogo em tempo real continua independentemente de entrada do usuário.

prossegue seguindo os três passos do ciclo anterior até seu encerramento. O encerramento pode decorrer de uma solicitação do usuário ou caso uma condição de vitória/derrota seja satisfeita.

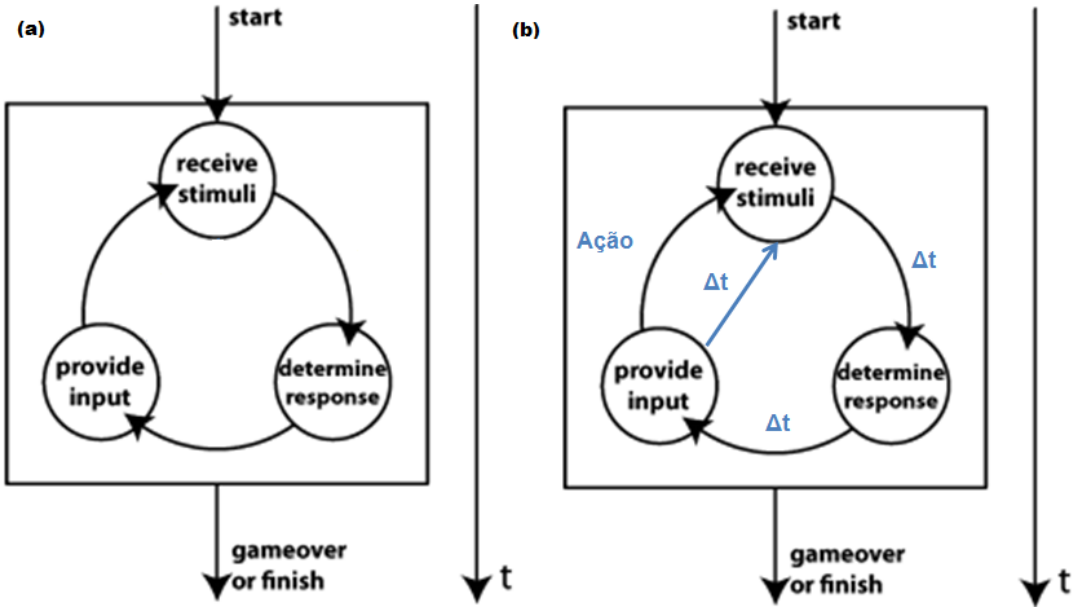


Figura 4. Passos do modelo de interação. Todos os passos são relativos ao usuário. (Adaptado de **(YUAN; FOLMER; HARRIS, 2011)** – (a) apresenta a imagem original).

O estímulo recebido pelo jogador (Passo 1) é sensorial – normalmente, visual, sonoro e/ou táctil[13]. Um estímulo pode ser primário, caso sua percepção seja essencial para que usuário compreenda o jogo, ou secundário, caso sua ausência não comprometa a continuidade do jogo.

A determinação da resposta (Passo 2) é realizada pelo usuário a partir da compreensão e interpretação do estímulo recebido. Esta transformação de estímulo em informação permite ao jogador a determinação de quais ações ele deve exercer. Este passo, portanto, envolve uma resposta cognitiva por parte do usuário – ele deve entender àquilo que lhe foi transmitido e tomar a melhor decisão sobre como proceder para dar continuidade ao jogo.

Determinada a resposta, o jogador fornece ao jogo a entrada que julga mais adequada (Passo 3). A entrada envolve uma ação física em um dispositivo de hardware (por exemplo, pressionar um botão ou movimentar-se diante de um sensor). Entradas podem ser analógicas (como uma alavanca) ou discretas (como um botão ou uma tecla).

[13] A rigor, *haptic*.

Após a realização (ou omissão da realização, representada na Figura 4 (b) como Δt) da entrada, o jogo é atualizado e o ciclo se repete. Por se tratar de um modelo cíclico, problemas de um passo podem prejudicar ou impossibilitar o(s) passo(s) seguinte(s). Um usuário que não consiga realizar a transição para o próximo passo terá sua experiência com o jogo prejudicada. Dentre outros, isto pode ocorrer se o usuário tiver alguma deficiência.

Yuan *et al.* (2011) observam que, para uma determinada deficiência, a maioria dos problemas de acessibilidade ocorre em passos e momentos específicos do modelo de interação e sugerem que, caso exista uma alternativa para a realização do passo, usuários com a deficiência considerada poderão utilizar o jogo. Explorar possibilidades para prover esta alternativa é o objetivo de muitos jogos acessíveis.

2.3 Design Universal e Jogos Universais

Um jogo acessível para um usuário não é, necessariamente, acessível para outro usuário. Jogos acessíveis são concebidos para uma determinada necessidade de interação e, portanto, possuem um público-alvo específico. Da mesma forma que um jogo convencional pode ser inacessível para um usuário com deficiência visual, um jogo com representação predominantemente aural pode ser inacessível para um usuário com deficiência auditiva.

Uma forma de permitir que pessoas com diferentes necessidades de interação possam utilizar um mesmo design é por meio do Design Universal. Uma das melhores formas de se definir Design Universal é por meio da descrição de seus princípios. São princípios do Design Universal (STORY; MUELLER; MACE, 1998):

1. Uso equitativo: o design é útil para pessoas com diversas habilidades e capacidades;
2. Flexibilidade de uso: o design acomoda uma vasta gama de preferências individuais e capacidades;
3. Uso simples e intuitivo: o uso do design é simples de se entender, independentemente da experiência, conhecimento, nível de concentração e habilidades de linguagem do usuário;

4. Informação perceptível: o design comunica informações efetivamente ao usuário, independentemente de condições do ambiente ou de capacidades sensoriais do usuário;

5. Tolerância a erros: o design minimiza acidentes ou consequências adversas de ações acidentais ou não pretendidas pelo usuário;

6. Baixo esforço físico: o design pode ser usado eficientemente e confortavelmente com um mínimo de fatiga do usuário;

7. Tamanho e espaço para aproximação e uso: tamanho e espaço apropriados são fornecidos para aproximação, alcance, manipulação e uso independentemente do tamanho do corpo, postura ou mobilidade do usuário.

O Design Universal busca que pessoas com diferentes capacidades e necessidade de interação possam utilizar uma mesma solução de design (STORY; MUELLER; MACE, 1998). Desta forma, o Design Universal incorpora e estende o conceito de acessibilidade: uma solução é concebida para ser adequada as necessidades de diferentes públicos. Uma possibilidade para a prática do Design Universal, mais adequada aos propósitos desta dissertação, é permitir que uma solução seja estendida de forma a se adequar, da melhor forma possível, à necessidade do usuário[14].

Em sistemas de software, o uso de soluções ajustáveis apresenta-se como uma das formas de se praticar o Design Universal (NERIS, 2010). Soluções ajustáveis permitem modificações em seu comportamento em tempo de uso e, assim, oferecem ao usuário a possibilidade de modificar a interface de acordo com suas preferências, necessidades e situações de uso (NERIS, 2010).

Isto pode ser feito, dentre outros, por *tailoring* – atividade de modificar uma aplicação computacional de acordo com seu contexto de uso (KAHLER et al., 2000 apud NERIS, 2010). As alterações realizadas por *tailoring* não se restringem a mudanças estéticas na interface: novas funcionalidades, mais adequadas a novos contextos de uso, também podem ser introduzidas (NERIS, 2010).

[14] Esta segunda possibilidade difere do uso de uma tecnologia assistiva para adaptar o resultado. Ao passo que a tecnologia assistiva adéqua a entrada ou a saída fornecida pelo sistema original, uma aplicação concebida empregando o Design Universal permitiria aos designers moldarem o sistema para atender às necessidades de interação do usuário. Ou seja, é possível utilizar uma tecnologia assistiva se for necessário – entretanto, seu uso foi considerado pelos designers para a melhor interação possível, ao invés de representar uma alternativa adicionada posteriormente para permitir a interação. Isto ficará mais claro considerando-se o conceito de *tailoring*.

Com o uso de *tailoring*, pode-se “projetar para a mudança”, pois se cria a possibilidade de atender a diferentes contextos ou cenários de uso em tempo de interação (NERIS, 2010). No contexto do design de jogos, a aplicação de *tailoring é* interessante, por permitir refinar a interação conforme for necessário. Isto, em potencial, pode ser feito buscando-se atender a diferentes necessidades de interação – mantém-se o conjunto de regras do jogo e alteram-se as entradas e saídas possíveis para se prover uma interação adequada ao usuário.

2.3.1 Design de Jogos Universais

Em virtude de diferentes necessidades de interação e das diversas barreiras de acessibilidade apresentadas em jogos, permitir que toda pessoa use um mesmo jogo é impossível (BARLET; SPOHN, 2012). Observa-se que, até o momento, grande parte da ênfase do Design Universal de jogos é destinada a melhorias de acessibilidade para que pessoas com diferentes necessidades de interação possam utilizar um mesmo jogo (BARLET; SPOHN, 2012; GRAMMENOS et al., 2006).

Neste contexto, destaca-se o conceito de *Universally Accessible Games* (UA-Games) (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007, 2009, 2011). O conceito de UA-Game baseia-se nos princípios do Design Universal e enfatiza a necessidade de tornar jogos mais acessíveis e jogáveis por pessoas com diferentes capacidades (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007; GRAMMENOS et al., 2006).

Dentre outros, o design de um UA-Game deve se adaptar dinamicamente a diferentes características individuais de usuários (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007). Em especial, pode-se também permitir que um mesmo jogo seja utilizado simultaneamente por pessoas com diferentes necessidades de interação.

O *Unified Design* (Design Unificado) é um processo que apoia o design de um UA-Game (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007, 2011) e é descrito como um processo estruturado de design de jogos, participativo, centrado no usuário e iterativo (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007, 2009). Desta forma, designers, especialistas de domínios (como especialistas em avaliação de usabilidade e de acessibilidade) e usuários finais do jogo trabalham em conjunto para melhorar continuamente o design do jogo.

Os passos do Design Unificado, como proposto por Grammenos *et al.* (2009), estão ilustrados na Figura 5[15]. Como mencionado na Seção 1.1, o objetivo do Design Unificado é permitir que o design do jogo seja realizado de forma a independer do contexto de uso – isto é, de interações de nível físico específicas. O design resultante pode ser especializado para um usuário ou para um grupo de usuários (perfil de usuário) por meio de *tailoring*.

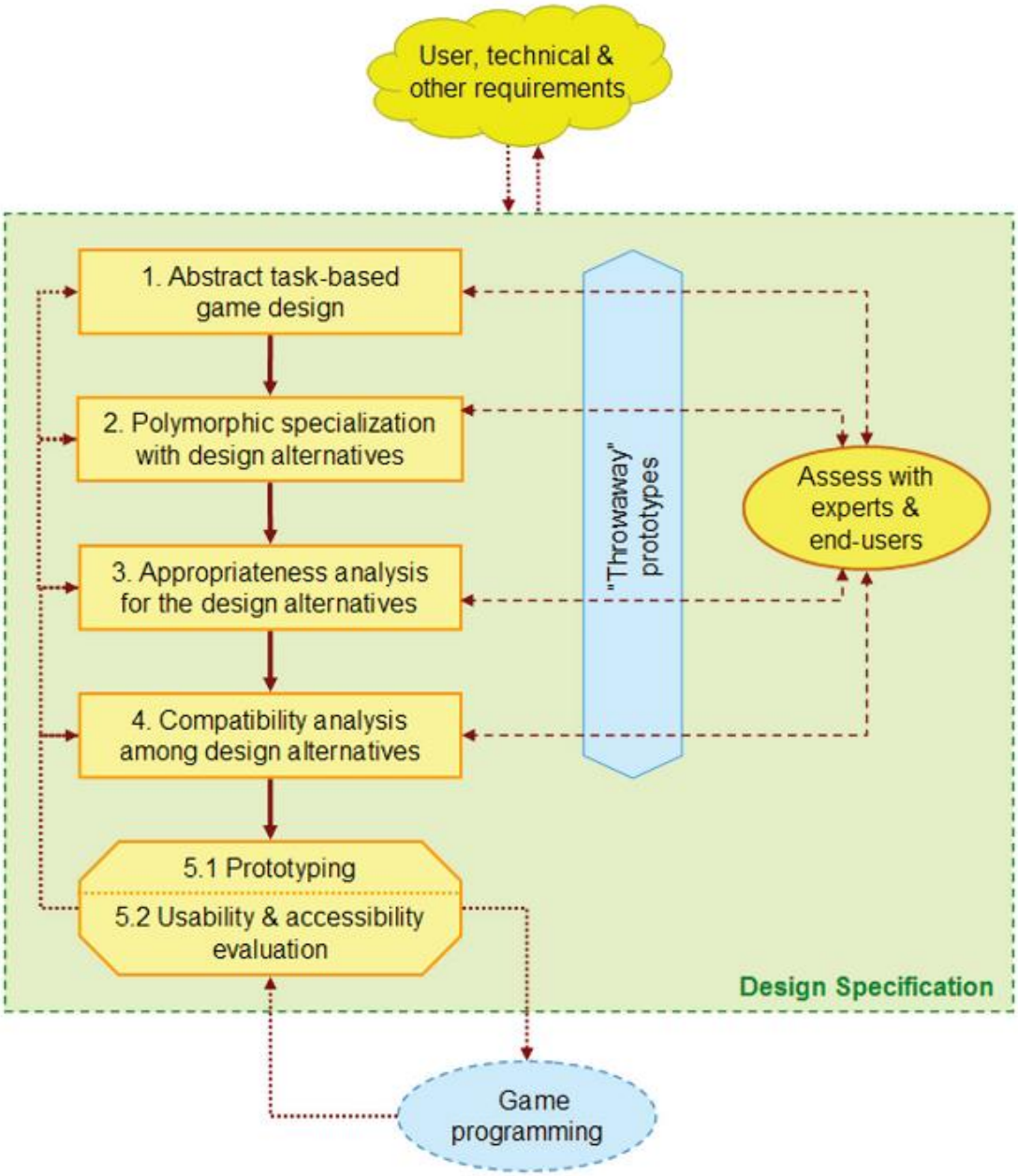


Figura 5. Passos do Design Unificado para o design de um UA-Game. (Extraído de **(GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009)**)

[15] A definição do processo pode ser encontrada em (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007).

Em linhas gerais, são criados perfis de jogadores contendo habilidades e capacidades de usuários. Em seguida (Passo 1), definem-se tarefas de alto-nível abstratas (isto é, isentas de interações de nível físico específicas) que deverão ser realizadas pelos jogadores (por exemplo, selecionar peça em um jogo de xadrez (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007)). Estas tarefas não pressupõem representações nem interações específicas.

No Passo 2, são propostas múltiplas especializações de nível físico para cada tarefa abstrata. Tipos e mecanismos de interação possíveis para a realização da tarefa são definidos neste passo. As especializações definidas são agrupadas de acordo com as suas exigências de capacidades e habilidades. Assim, por exemplo, dispositivos de entrada e saída e representações sensoriais são mapeadas para as tarefas abstratas definidas no Passo 1 – para um controle manual da peça de xadrez, a interação poderia envolver teclado ou comandos de voz.

No Passo 3, são correlacionadas as diversas tarefas especializadas propostas com as capacidades e habilidades de interação de usuários. Busca-se classificar a adequação da especialização de nível físico proposta para cada capacidade de interação – os possíveis valores são: ideal, apropriado, possível de ser usado, inadequado e neutro (a especialização não afeta nem é afetada pela capacidade em questão). Para o exemplo anterior e para um determinado perfil, a movimentação da peça por teclado poderia ser adequada e, empregando comandos de voz, poderia ser ideal. Para outro, os comandos de voz poderiam ser inadequados e o uso do teclado poderia ser adequado.

No Passo 4, verifica-se a compatibilidade de cada especialização proposta como entrada no Passo 3 com relação às demais especializações para entrada. Caso duas especializações não possam ser realizadas simultaneamente, elas são classificadas como incompatíveis. Caso contrário, elas são classificadas como compatíveis. Assim, por exemplo, como comandos de voz podem ser usados simultaneamente a um teclado, essas especializações são compatíveis.

Nos Passos 1 a 4, pode-se usar um protótipo do tipo *throw-away*[16] para a obtenção de feedback das partes interessadas. Este protótipo tende a ser simples,

[16] Um protótipo *throw-away* pode ser considerado como um protótipo criado para a validação de uma idéia ou de uma funcionalidade específica. Estes protótipos podem ser descartados após a validação, razão a qual origina a nomenclatura.

envolvendo desenhos feitos à mão ou implementações de baixa fidelidade para provas de conceito.

Em seguida, as tarefas são prototipadas (Passo 5.1) e avaliadas (avaliações de usabilidade e acessibilidade – Passo 5.2). O processo é repetido iterativamente, conforme necessário. Ao final do processo, o design é implementado: são geradas versões acessíveis do jogo para cada um dos perfis de usuário utilizados.

Ao jogar, o usuário escolhe o perfil mais adequado às suas necessidades e o jogo, então, é apresentado de acordo com as definições do perfil. Assim, para um jogador cego, a apresentação do conteúdo gráfico poderia ser realizada por meio de áudio. Já para um jogador surdo, o conteúdo sonoro poderia ser apresentado por meio de gráficos.

Interaction model	High-level strategies		Low-level strategies
Receive stimuli (visual and hearing)	Enhance stimuli	*visual*	High contrast color schemes
			Increase font size
			Color blind color schemes
			Zoom options
	Replace stimuli	*Audio → visual*	Text (subtitles, closed captioning)
			Non-text (visual cues, sound radar, signing)
		Visual → audio	Speech (screenreader, self voicing)
			Audio cues
			Sonification (earcons, sonar, auditory icons)
		Visual → haptic	Haptic cues
Determine response (cognitive)	Reduce stimuli	*Visual*	Limit number of game objects
			Simplify storyline
	Reduce time constraints		Increase response time
			Slow down game
	Reduce input		Remove input
			Automate input
Provide input (motor)	Reduce input		Scanning
			Remove input
			Automate input
	Replace input		Voice/brain control

Figura 6. Estratégias para promover acessibilidade. (Extraído de (YUAN; FOLMER; HARRIS, 2011))

2.4 Estratégias Empregadas em Jogos Acessíveis e Universais

Um resumo das principais estratégias exploradas para aumentar a acessibilidade em jogos é apresentado na Figura 6 (tradução no Quadro 1). Ela se refere ao modelo de interação descrito na Seção 2.2.

Quadro 1. Estratégias para promover acessibilidade (traduzido da **Figura 6**).

Modelo de Interação	**Estratégias de Alto-Nível**		**Estratégias de Baixo-Nível**
Recebimento de estímulo (visual e auditivo)	Aumentar o estímulo	Visual	Esquemas de cores com alto-contraste
			Aumento do tamanho da fonte
			Esquemas de cores para daltonismo
			Opções de zoom
	Substituir o estímulo	Aural → Visual	Textuais (legendas, *closed captions*)
			Não-textuais (indicações visuais, radares sonoros, sinais)
		Visual → Aural	Fala (leitor de tela, *self voicing*)
			Indicações sonoras
		Visual → Táctil	Sonificação (*earcons*, sonar, *auditory icons*)
			Indicações tácteis
Determinação de resposta (cognitivo)	Reduzir o estímulo	Visual	Limitar o número de objetos de jogo
			Simplificar a história
	Reduzir restrições de tempo		Aumentar o tempo de resposta
			Reduzir a velocidade do jogo
	Reduzir entrada		Remover entrada
			Automatizar entrada
Realização de entrada (motor)	Reduzir entrada		Escaneamento
			Remover de entrada
			Automatizar de entrada
	Trocar entrada		Controles por voz/cérebro

Por exemplo, na Figura 6, a segunda linha refere-se ao Passo 1 do modelo de interação, descrevendo possíveis estratégias para promover a acessibilidade em jogos. As estratégias indicadas para estes casos são *enhance stimuli* (aumentar o

estímulo) e *replace stimuli* (substituir o estímulo). Em *replace stimuli*, caso se deseje trocar o estímulo de uma modalidade em áudio para uma modalidade visual, uma possível técnica seria trocar o som por representação textual (como legendas) ou usar uma representação não-textual (como indicativos visuais).

O Quadro 2 sumariza alguns jogos estudados pelo autor ao longo do projeto de mestrado. A maioria deles são jogos acessíveis; três são universais e alguns outros são jogos convencionais. Para cada jogo, é apresentado seu nome, gênero, o perfil de usuário para o qual o jogo foi pensado e o tipo de estratégia utilizada para aumentar a acessibilidade. Todos os jogos apresentados podem ser usados por usuários médios. No entanto, como mencionado ao longo desta seção, o jogo normalmente será inacessível para jogadores com deficiências distintas daquelas para às quais o jogo foi concebido.

Quadro 2. Alguns jogos estudados ao longo da realização deste trabalho.

Nome	**Deficiências Consideradas**				**Técnicas e Estratégias**
	Visual	*Auditiva*	*Motora*	*Cognitiva*	
(ARAÚJO; GOTO; VIOLARO, 2000)			✓		Entrada por Voz
(LIN et al., 2004)			✓		*Eye-Tracking*
(PALKE, 2004)			✓		Ondas cerebrais
(YUAN; FOLMER; HARRIS, 2011)			✓		Controles de uma mão
Access Invaders (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009)	✓	✓	✓	✓	Perfis de usuários personalizáveis; Possibilidade de usar diferentes dispositivos para ES; Alterações do *gameplay* para adaptar ao jogo ao usuário; Também é possível alterar o tamanho de elementos da interface, layout, dificuldade, esquemas de cores, contraste e sons 3D.
Bayonetta (SEGA, 2012)				✓	Níveis de dificuldade *easy* / *super easy* com controles automatizados.

Nome	**Deficiências Consideradas**				**Técnicas e Estratégias**
	Visual	*Auditiva*	*Motora*	*Cognitiva*	
Coucou Cachè (SEHABA; ESTRAILLIER; LAMBERT, 2005; YUAN; FOLMER; HARRIS, 2011)				✓	Autismo. Maiores limites de tempo para interação. Controles simples e intuitivos. Poucos estímulos.
Dark Destroyer (Dark Destroyer, 2004)	✓				*Auditory interfaces*. Som 3D.
Doom3 [CC] (GAMES[CC], 2007)		✓			Legendas em diálogos. Transcrição textual de efeitos sonoros.
Dragon Age: Origins (BIOWARE, 2009))	Parcial	✓			Esquemas de cores para usuários daltônicos. Legendas em diálogos.
Drive (Drive, 2003)	✓				*Auditory interfaces*. Som 3D.
Dwarf Fortress (ADAMS; ADAMS, 2013)		✓			Ausência de sons. Jogo com gráficos em texto.
FIFA 13 (ELECTRONIC ARTS, 2014)			✓		Possibilidade de jogar usando apenas mouse.

Nome	**Deficiências Consideradas**				**Técnicas e Estratégias**
	Visual	*Auditiva*	*Motora*	*Cognitiva*	
Ilbo (KWEKKEBOOM; VAN WELL, 2002; YUAN; FOLMER; HARRIS, 2011)				✓	Dificuldade de aprendizado. Maiores limites de tempo para interação. Resolução de problemas mais permissiva.
Lone Wolf (Lone Wolf, 2000)	✓	Parcial			*Auditory interfaces*. Versão em texto.
Mudsplat (Mudsplat, 2005)	✓				*Auditory interfaces*. Som 3D.
Smile (ADAMO-VILLANI; WRIGHT, 2007)		✓			Uso de linguagens de símbolos.
SóSoprando (FAVA, 2008)			✓		Uso de sopro para entrada.
Super Mario Bros Wii (NINTENDO, 2009)				✓	Modo “autopilote”: permite que usuários avancem fases ou partes problemáticas.

Nome	**Deficiências Consideradas**				**Técnicas e Estratégias**
	Visual	*Auditiva*	*Motora*	*Cognitiva*	
The Sims (ELECTRONIC ARTS, 2013)		✓			Sons estéticos. Imagens exibidas quando um som é reproduzido.
Top Speed 3 (PLAYING IN THE DARK, 2011)	✓				*Auditory interfaces*. Som 3D.
UA-Chess (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2009)	✓	✓	✓	✓	Perfis de usuários personalizáveis; Possibilidade de usar diferentes dispositivos para entrada e saída; Scanning, navegação não-visual, entrada por voz; Leitor de tela embutido; Redimensionamento da tela; Zoom.
XIII (UBISOFT, 2003)		✓			Sons estéticos. Imagens exibidas quando um som é reproduzido.
Zork (INFOCOM, 1980)		✓			Ausência de sons. Jogo textual.

2.5 *Motores de Jogos (Game Engines)*

Para o desenvolvimento de jogos, iterações são fundamentais. Para Schell (2008), a única regra existente no design de jogos é "*The Rule of the Loop*". A regra diz que: "*the more times you test and improve your design, the better your game will be*". Ou seja, quanto mais iterações forem realizadas para o design, melhor será o jogo. Seja para testar a viabilidade de novas mecânicas, seja para testes de viabilidade, é importante que protótipos possam ser criados rapidamente e, preferencialmente, com o menor esforço possível.

Neste sentido, esforços de implementação (como tempo e custo) são oponentes da quantidade de iterações. Métodos, técnicas, ferramentas e processos são empregados para transformar o design concebido para o projeto no sistema correspondente.

O reuso de software constitui uma das formas de se facilitar e agilizar o desenvolvimento de novos sistemas. Bibliotecas, *Application Programming Interfaces* (APIs), padrões de projeto e de arquitetura e *frameworks* são algumas das formas de se reutilizar conhecimento, experiência e/ou código-fonte criados anteriormente em novos projetos. Desta forma, espera-se que os novos projetos possam ser criados com maior qualidade e em menor tempo.

Em jogos digitais, *game engines* (motores para jogos) são estruturas ou *frameworks* utilizados para o desenvolvimento de novos jogos. Motores promovem o reuso de software em jogos, facilitando a implementação de tarefas recorrentes em jogos (como a renderização) e permitindo que os desenvolvedores possam se focar mais no desenvolvimento do jogo. Motores populares incluem *Unity*[17], *Unreal Engine*[18] e Cry*Engine*[19] – as duas últimas, em especial, são frequentemente lembradas em virtude da qualidade da renderização gráfica.

[17] Unity® 3D. http://unity3d.com/
[18] UDK®: Unreal Developer's Kit. http://www.unreal*engine*.com/udk/
[19] Cry*Engine*® 3. http://www.crytek.com/cry*engine*/cry*engine*3/overview

Como a maioria dos jogos existentes são voltados para um modelo de usuário médio, grande parte do foco no desenvolvimento de um motor é direcionado para o uso de gráficos (o estímulo primário adotado na maioria dos jogos convencionais). De fato, a evolução gráfica em jogos e em motores costuma ser muito mais significativa que a de outras modalidades.

Outras modalidades, como, por exemplo, o áudio, são normalmente preteridas em relação ao conteúdo gráfico (GREGORY, 2009). Isto pode dificultar a um desenvolvedor implementar um jogo que independa de representação gráfica, como o design resultante do Design Unificado descrito anteriormente.

2.5.1 Propósito e Funcionalidades

Não existe uma única definição sobre o que é um motor de jogos (GREGORY, 2009). Motores podem possuir diferentes propósitos e complexidades. Um motor pode variar de uma simples API especializada para o desenvolvimento de jogos até *frameworks* ou *middlewares* complexos, contendo ferramentas sofisticadas para auxiliar na criação de conteúdo para o jogo. O último caso é comum no desenvolvimento de jogos *Triple-A*[20] (AAA).

De forma geral, motores são estruturas ou *frameworks* utilizados para auxiliar no desenvolvimento de jogos ou facilitar a adaptação de um jogo existente para uma nova plataforma (como diferentes sistemas operacionais ou dispositivos). Diversas abordagens são possíveis para implementar um motor – cada qual com suas vantagens e desvantagens (GREGORY, 2009).

O número e os gêneros de jogos que podem ser criados usando um motor também podem variar. Um motor pode ser construído para atender aos requisitos específicos de um único jogo. Um segundo motor pode ser mais adequado e especializado para um gênero de jogos. Um terceiro pode ter um propósito mais geral e, assim, ser usado para o desenvolvimento de jogos de gêneros diversos – por exemplo, ele não assume um tipo de projeção (como câmera em primeira pessoa) ou *gameplay* específico.

[20] Jogos AAA são jogos criados por grandes estúdios ou empresas de desenvolvimento de jogos, com orçamentos estimados em dezenas de milhões de dólares.

Em geral, motores como os últimos costumam ser muito mais extensíveis e genéricos que o primeiro. Motores extensíveis podem fornecer uma base para o desenvolvimento de novos jogos (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012). Quanto mais extensível, genérico e flexível o motor, maior será o potencial de um novo jogo poder usá-lo.

Por outro lado, quanto mais específico for o motor, maior será sua adequação ao(s) jogo(s) (ou gênero(s)) para o(s) qual(is) foi concebido, mas menor será seu potencial de reuso.

2.5.2 Funcionalidades e Subsistemas Característicos

Independentemente da complexidade e da abordagem para a implementação, algumas funcionalidades são comuns a diversos motores. Algumas destas funcionalidades são: renderização, animação, reprodução de sons, simulação de física e inteligência artificial.

Estas funcionalidades são comumente implementadas por meio de módulos ou de sistemas (normalmente denominados subsistemas). Um módulo ou um subsistema trata de uma funcionalidade específica, contribuindo para a separação do processamento por objetivo (ou responsabilidade). Por exemplo, o sistema de renderização é o responsável por gerar a representação uma imagem bi ou tridimensional a partir de um modelo e exibi-la em um dispositivo de saída (como um monitor). O sistema de física, por sua vez, realiza a simulação de processos físicos pertinentes para o jogo (por exemplo, verificação de colisões entre corpos rígidos).

A implementação de subsistemas pode ser própria do motor ou utilizar uma biblioteca, um Software *Development Kit* (SDK) ou um *middleware* existente.

Muitos motores utilizam arquiteturas baseadas em componentes de software (*Commercial Off-the-Shelf* - COTS (PRESSMAN, 2009; SOMMERVILLE, 2010)) para a implementação de subsistemas – especialmente para áudio, física e reprodução de vídeos. FMOD[21], Miles[22] e WWise[23] são SDKs comumente utilizadas para a reprodução de áudio em jogos. Por outro lado, OpenAL[24] é uma API usada por

[21] FMOD. http://www.fmod.org/
[22] Miles Sound System. http://www.radgametools.com/miles.htm
[23] WWise. http://www.audiokinetic.com/en/products/208-wwise/
[24] OpenAL. http://connect.creativelabs.com/openal/default.aspx

diversos motores para a implementação do subsistema de áudio. Havok[25] e Bullet Physics[26] são SDKs usadas para simulação de física.

2.5.3 Modelos Arquiteturais Relacionados

Como mencionado na Seção 2.5, diversas abordagens podem ser utilizadas para a construção de um motor de jogos. Esta seção apresenta algumas das abordagens que contribuem para tornar o motor mais extensível e flexível em tempo de execução – tanto em relação para o *gameplay* quanto para a apresentação. Estas abordagens serão detalhadas ao longo dos próximos capítulos para a construção de um motor para jogos universais.

2.5.3.1 Arquitetura *Data-Driven*

Um motor com uma arquitetura *data-driven* permite definir o fluxo de execução do programa por meio de dados fornecidos como entrada (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012). Uma arquitetura *data-driven* carrega os dados utilizados ao longo da execução do software por meio de recursos externos, como arquivos de texto ou banco de dados. Isto torna o motor mais extensível e permite utilizá-lo, sem grandes modificações, como base para o desenvolvimento de novos jogos (GREGORY, 2009).

Por exemplo, ao invés de se definir constantes e valores no código-fonte, em uma arquitetura *data-driven* os valores de inicialização são obtidos de arquivos externos, como arquivos na linguagem de marcação *Extensible Markup Language* (XML). Isto permite que alterações possam ser feitas sem alterações no código-fonte, aumentando-se a velocidade de iterações e melhorias no fluxo de execução. No caso de jogos, por exemplo, ao invés de se definir um arquivo para uma imagem no código-fonte, em uma arquitetura *data-driven* a aplicação extrairia de um arquivo externo o caminho para a imagem a ser usada. Desta forma, caso se deseje alterar a imagem, basta-se alterar o caminho no arquivo e re-executar a aplicação.

Além de se promover reuso de software, um motor *data-driven* pode facilitar a criação de jogos tanto para novos quanto para desenvolvedores experientes – pois

[25] Havok. http://www.havok.com/products/physics
[26] Bullet Physics. http://www.bulletphysics.com/

permite, por exemplo, a implementação e o uso de editores destinados à criação e edição de conteúdo para o jogo.

2.5.3.2 Modelo Entidade-Componente

O modelo Entidade-Componente (*Actors Model* em (MCSHAFFRY; GRAHAM, 2012) permite a caracterização de entidades por meio de agregação (ou composição) de componentes[27] (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012; NYSTROM, 2012). Este tipo de abordagem facilita a criação de objetos que serão representados no jogo, aumentando-se a flexibilidade para o design e promovendo o reuso.

Cada componente agrega atributos e comportamentos característicos à entidade. Componentes podem ser adicionados ou removidos da entidade conforme necessário – inclusive em tempo de execução. Os componentes das entidades são processados por sistemas específicos. Assim, para adicionar um comportamento a uma entidade, basta adicionar-lhe o(s) componente(s) necessário(s).

Em algumas implementações[28] deste modelo, uma entidade pode ser vista, como um identificador. Seus atributos e suas características são conferidos exclusivamente por meio dos componentes a ela agregados. A Figura 7 ilustra uma representação simplificada para os elementos do clone de *Pong*.

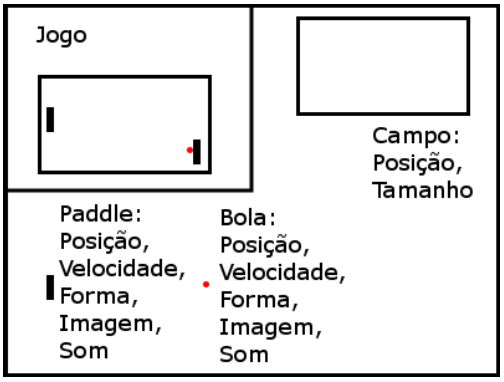


Figura 7. Decomposição das entidades do jogo em alguns de seus respectivos componentes **(GARCIA; NERIS, 2013a)**.

Na imagem, um `Paddle` não é uma instância da classe `Paddle`; ele é uma instância da classe `Entidade` que tem um componente de posição, um componente

[27] No modelo Entidade-Componente, o significado de um componente não deve ser confundido com componentes de software (como os citados na Seção 2.5.2).

[28] Por ser um modelo relativamente novo (uma das primeiras descrições pode ser encontrada em (SCOTT BILAS, 2002)), não existe uma única forma de implementação. Assim como no caso de motores, várias abordagens são possíveis – cada qual com suas vantagens e desvantagens.

de velocidade, um componente de forma, um componente de imagem e um componente de som. Por outro lado, o `Campo` é uma instância da classe `Entidade` que tem os componentes posição e tamanho. Assim, um `Campo` não apresenta os mesmos comportamentos de um `Paddle`, dado que seus componentes são diferentes.

Algumas implementações do modelo (*c.f.* (MCSHAFFRY; GRAHAM, 2012)) permitem definir componentes como interfaces. O componente abstrato permite que a implementação seja definida por um componente concreto, como ilustrado na Figura 8. O componente concreto pode ser instanciado em tempo de execução e, portanto, pode ser alterado de acordo com as necessidades do jogo. Componentes abstratos são normalmente utilizados para componentes de física (como ilustrado na Figura 8) e de inteligência artificial.

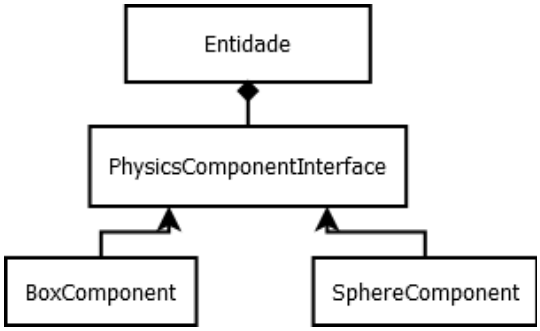


Figura 8. Componentes sendo utilizados como uma interface. (Adaptado de **(MCSHAFFRY; GRAHAM, 2012)**)

Para o motor GCC4, McShaffry e Graham (2012) definem entidades de jogo como atores. Esta definição é precisa: da mesma forma que atores encenam seus papeis para compor, dar forma e criar um espetáculo, atores de jogo moldam o mundo e definem a experiência de jogo. Um espetáculo sem atores seria inanimado; da mesma forma, um jogo sem atores seria vazio, não-interativo.

2.5.3.3 Arquiteturas *Event-Driven*

Para Gregory (2009), “jogos são inerentemente *event-driven*”. Gregory define um evento como “qualquer coisa de interesse que acontece durante o *gameplay*”. Arquiteturas *event-driven* exploram o uso de eventos para desacoplar a lógica da implementação.

Estas arquiteturas exploram estratégias como os padrão *Observer* (GAMMA et al., 1994) para permitir que partes interessadas respondam ao objeto de interesse ao qual se inscreveram. Quando um evento é despachado, todos os *event handlers* registrados a ele são comunicados de sua ocorrência e podem trata-lo de forma conveniente. Isto permite alterar-se o fluxo de execução de programa de forma conveniente. Outra possibilidade é usar um evento para a comunicação entre diferentes partes de um sistema.

Um evento pode ter tantos *handlers* quanto forem necessários; para que um *handler* responda a um evento, basta inscrever-se (ou registrar-se) ao evento desejado. Da mesma forma, é possível a um *handler* desregistrar-se de um evento. Ambas as operações podem ser feitas em tempo de execução.

2.6 Limitações de Motores Convencionais de Código-Aberto para o Design de Universal

O desenvolvimento de um jogo mais universal exige esforços conjuntos de design e de implementação. É necessário que diferentes modalidades possam ser exploradas para a apresentação do jogo e do *gameplay* – sem a obrigatoriedade de uma modalidade específica. Ou seja, tanto o design quanto a implementação do jogo devem ser feitas da forma mais abstrata possível, sem referências às modalidades ou interações específicas. Neste sentido, o Design Unificado pode auxiliar no processo de design.

Mesmo em motores destinados a criação de grandes jogos comerciais, como os jogos AAA, pode-se observar que, devido à ênfase visual de jogos digitais convencionais, subsistemas relacionados à geração e apresentação de conteúdo gráfico são, normalmente, muito mais complexos e importantes que os relacionados a outras modalidades. Ao passo que subsistemas gráficos costumam apresentar e implementar técnicas avançadas como *shaders*, iluminação e *radiosity*, um subsistema de áudio, muitas vezes, é restrito a reprodução de arquivos de som.

Isto é observado tanto em motores comerciais quanto de código-aberto – conteúdos de outras modalidades, como sonoros, são, muitas vezes, preteridos no desenvolvimento de jogos e de motores (GREGORY, 2009).

Dada a variedade de possíveis necessidades de interação em um contexto de jogos universais, um motor deve ser extremamente flexível quanto as possibilidades para ES. Entretanto, como normalmente motores apresentam estruturas complexas e otimizadas para a apresentação de gráficos, pode ser difícil de se alterar a arquitetura para a inclusão de diferentes sistemas de ES ou de tecnologias assistivas.

Com base na nos modelos arquiteturas descritos na Seção 2.5.3, o motor GCC4 descrito por McShaffry e Graham (MCSHAFFRY; GRAHAM, 2012) foi considerado pelo autor como o de arquitetura mais próxima para a necessário para este trabalho. Utilizando-se os resultados de Rocha *et al.* (2010) e páginas especializadas como DevMaster[29], Gamasutra[30] e Pixel Prospector[31], foram analisados motores código-aberto mais populares como alternativas ao motor GCC4. Com base nos estudos anteriores, escolheu-se os motores Crystal Space[32], Delta3D[33], GamePlay[34], Irrlicht[35], Ogre3D[36], Open Game *Engine*[37], OpenSceneGraph[38] e Panda3D[39] para uma análise de apropriação para o desenvolvimento de um jogo universal.

Considerando-se estes motores em relação aos modelos da Seção 2.5.3 e adotando-se o motor GCC4 como mais próximo do ideal, observa-se a existência de alguns entraves ao desenvolvimento de jogos universais mediante uso dos demais motores, como:

- Alguns dos motores de código-aberto mais usados (Ogre3D, Irrlicht e OpenSceneGraph) dedicam-se exclusivamente a renderização de gráficos;
- No caso dos motores de código-aberto estudados (apresentados no Quadro 3), percebe-se que grande ênfase é dada na interface de usuário com representação visual[40]. Ao passo que diversos recursos são fornecidos para

[29] DevMaster.net. http://devmaster.net
[30] Gamasutra. http://www.gamasutra.com/
[31] PixelProspector.com. http://www.pixelprospector.com/
[32] CrystalSpace. http://crystal.sourceforge.net/
[33] Delta3D. http://www.delta3d.org/
[34] GamePlay. http://www.gameplay3d.org/
[35] Irrlicht. http://irrlicht.sourceforge.net/
[36] Ogre3D. http://www.ogre3d.org/
[37] OpenGameEngine. http://oge.sourceforge.net/
[38] OpenSceneGraph. http://www.openscenegraph.org/
[39] Panda3D. http://www.panda3d.org/
[40] DevMaster (http://devmaster.net/devdb/engines) apresenta uma lista de *game engines*. Os critérios de seleção de funcionalidades revelam a ênfase gráfica dada pelas *engines* convencionais.

renderização e efeitos gráficos (como *shaders*, sombras, animações e luzes), menos recursos são fornecidos para a apresentação ou renderização de outras modalidades, como sons.

Quadro 3. Características de alguns dos principais motores de código-aberto.

Nome	EC	Gráficos	Áudio	Documentação	Última atualização	*Scripting*	Suporte a Eventos
GCC4	Sim	3D	2D	Adequada	2012	Lua	Sim
Crystal Space		2D/3D	3D	Adequada	02/2014	Python	Parcial
Delta3D	Sim	2D/3D	3D	Defasada	02/2014	Python	Parcial
GamePlay		2D/3D	3D	Inadequada	04/2014	Lua	Não
Open Game *Engine*	Sim	2D/3D	3D	Defasada	07/2009	Lua	Parcial
Panda3D		2D/3D	3D	Adequada	02/2014	Python	Parcial

De fato, todos os motores estudados pressupõem que o conteúdo principal será gráfico e que os demais conteúdos serão complementares aos gráficos. Para que outros modos de apresentação sejam utilizados com mesma ênfase, a extensão destes motores exigiria mudanças estruturais;

- Uma das principais vantagens de se empregar um motor consiste no uso de editores de mundo (*world editors*) para o desenvolvimento do jogo (GREGORY, 2009). Como modalidades não-visuais não são exploradas primariamente em motores convencionais, seus usos em editores são limitados. O uso do motor seria restrito quase que apenas aos módulos ou sistemas de renderização – o que pode ser feito por meio de APIs (como OpenAL) ou *frameworks* especializados para jogos (por exemplo, Ogre3D e Irrlich, para gráficos; FMOD, Miles ou Wwise, para áudio);
- Dentre os motores de código-aberto estudados, apenas Delta3D e Open Game *Engine* adotam (em parte) uma arquitetura voltada a componentes. No entanto, ambos os motores não estão mais em desenvolvimento. O modelo de componentes usado nestes motores também não é flexível o suficiente para os propósitos desta dissertação – dado que deve ser extensível e adaptável em tempo de execução. Esta é uma limitação mesmo em motores comerciais, como *Unity*3D (FREITAS et al., 2012). Motores que possibilitem que toda comunicação entre entidades e subsistemas possam ser feitas por meio de eventos podem contribuir neste sentido (FREITAS et al., 2012;

GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012). Entretanto, nenhum dos motores estudados permite que a comunicação seja feita exclusivamente por meio de eventos.

Neste sentido, esta dissertação adotou o motor GCC4 como ponto de partida para a realização do trabalho. Para isto, reusou-se ao máximo à arquitetura do motor GCC4, tornando-a ainda mais flexível para o contexto do desenvolvimento de jogos universais.

Algumas das funcionalidades foram reusadas, como será apresentado no Capítulo 5. Várias outras foram introduzidas com o propósito de tornar o motor livre de referências de entradas e saídas – a arquitetura resultante do desenvolvimento deste projeto permitir adicionar ou remover sistemas de ES de forma simples e sem mudanças arquiteturais. Conforme será apresentado no Capítulo 6, o resultado obtido permite que grande parte das modificações de ES nos jogos criados com o motor UGE possam ser feitas sem alterações no código-fonte – elas podem ser feitas a partir de um flexível e extensível perfil de interação definido por esta dissertação.

2.7 Considerações Finais

Este capítulo apresentou uma breve síntese da literatura necessária para este trabalho. Para isto, foram discutidos temas relacionados ao design e à implementação de jogos. Iniciou-se com a definição de um modelo para a interação em jogos, que será explorado ao longo desta dissertação. O modelo apresentado permitiu a identificação de partes de um jogo que podem ser problemáticas para diferentes usuários.

Em seguida, foram apresentados os principais conceitos, objetivos e princípios do Design Universal. Por meio do Design Universal, busca-se possibilitar o uso de um mesmo design (ou, no caso, de um mesmo sistema de software) pelo maior número possível de usuários, da melhor forma possível. Por ser inclusivo desde a definição, o Design Universal mostra-se como uma possível alternativa rumo ao Acesso Participativo e Universal do Cidadão Brasileiro ao Conhecimento, contribuindo para os propósitos deste trabalho.

Para sistemas de software, foi descrito que *tailoring* permite alterações da aplicação de acordo com seu contexto de uso e em tempo de uso. Desta forma, pode-se "projetar para a mudança", adequando-se o sistema para diferentes contextos e necessidades de uso de diferentes usuários. Não se restringe, portanto, às necessidades e ao contexto de uso de um modelo de usuário médio.

Em sequência, descreveu-se brevemente o design de jogos universais. O Design Universal foi descrito, mostrando-se como proceder para a realização do design de um jogo mais universal. Realizando um design abstrato, especializado por meio de *tailoring* para as necessidades e capacidades do usuário, mostrou-se possível permitir que um mesmo jogo fosse utilizado por pessoas com necessidades de interação diferentes.

No entanto, um jogo digital não se restringe ao seu design. Antes de ser utilizado, o jogo deve ser implementado por meio de alguma linguagem de programação ou de um framework especializado. Além disso, devido à flexibilidade necessária para a aplicação de *tailoring* em jogos universais, as técnicas empregadas para a implementação devem ser igualmente flexíveis e extensíveis.

Para isto, apresentou-se que a implementação de um design (ou de uma ideia) para um jogo é facilitada com o uso de um motor. Por abstrair tarefas e funcionalidades comuns a jogos, um motor permite que os desenvolvedores possam se dedicar mais à implementação *per se* do jogo. Motores extensíveis contribuem para o desenvolvimento de novos jogos. Quanto mais extensível o motor, maior tende ser a quantidade de novos jogos que possam utilizá-lo em seu desenvolvimento.

Entretanto, pode-se observar que muitos dos motores existentes tendem a valorizar a apresentação gráfica, preterindo apresentações em outras modalidades. Com isto, para o desenvolvimento de um jogo universal, alterações em motores existentes são, portanto, inevitáveis. Muitas destas alterações devem ser estruturais para apoiarem a implementação de um design mais abstrato.

Gregory (2009) afirma que "tudo em uma *game engine* deve ser feito da forma mais simples possível, mas não mais simples"[41]. Desta forma, este projeto partiu do pressuposto de que construir um novo motor, utilizando estratégias, APIs e *frameworks* mais adequados para o desenvolvimento da solução, pode gerar

[41] Parafraseando a famosa frase de Einstein.

resultados melhores do que se adaptar um motor existente e subutilizar seus recursos e funcionalidades.

Desta forma, os próximos capítulos descrevem o resultado desta dissertação: a construção de um motor destinado a facilitar o desenvolvimento de jogos universais.

Capítulo 3

UM JOGO DIGITAL SEM JOGADOR

"If a tree falls in a forest and no one is around to hear it, does it make a sound?"

(Se uma árvore cair na floresta, mas não houver ninguém para ouvi-la, ela fez barulho?)

3.1 Considerações Iniciais

O Capítulo 2 apresentou o Design Universal e um processo para a prática do Design Universal para jogos, o Design Unificado. Diante da diversidade de possíveis necessidades e habilidades de interação dos usuários, o Design Unificado sugere a identificação e a criação de tarefas de alto-nível independentes de interações de nível físico. No caso de jogos digitais, estas interações refletem as possíveis ES realizadas pelo usuário ao jogar.

Pelo modelo de interação apresentado no Capítulo 2, pode-se observar que, nos Passos 1 e 3, os problemas de acessibilidade decorrem da interação (relacionados diretamente a ES). Desta forma, para se implementar um jogo universal, deve-se fornecer mecanismos que tornem a simulação de lógica do jogo independente de ES.

Neste capítulo, busca-se reduzir jogos a simulações, eliminando a interação presente em jogos digitais: um jogo sem jogador, por assim dizer. Este paradoxo tem como objetivo identificar protocolos e mecanismos que possam simular a interação

sem a necessidade de um usuário. Parte-se do pressuposto de que um jogo sem usuário é um jogo universal. Ao se mostrar possível criar um jogo sem jogador, pode-se adicionar usuários com perfis específicos de necessidades de interação, um a um. Se o jogo resultante da adição de cada necessidade de interação manter-se acessível, conclui-se que é possível implementar um jogo universal para os públicos abrangidos com os mecanismos e protocolos definidos. Em outras palavras, remove-se o usuário do jogo para, mais tarde, permitir ao jogo incluir o jogador.

Neste capítulo, realiza-se a primeira parte deste processo – a remoção do jogador do jogo. No Capítulo 4, reintroduz-se o jogador ao jogo de forma inclusiva – ele indica como alterar o jogo para que este possa incluir o jogador.

3.2 Um Mundo de Jogo Digital Sem Jogador

O ponto de partida considera as estratégias de implementação elencadas no Capítulo 2 como primitivas para a construção de um jogo. Alguns dos conceitos que definem as estratégias escolhidas serão adotados como elementos para a construção de uma simulação de jogo. Estes elementos serão combinados para se realizar uma análise das relações existentes entre os elementos de um jogo digital para identificar seus relacionamentos.

Usando um número mínimo de elementos e explorando seus relacionamentos, buscar-se-á moldar um mundo genérico de jogo que possa abstrair toda a simulação da lógica do jogo. A partir deste mundo de jogo (chamado, doravante, Mundo Concreto de Jogo), procurar-se-á remover as interações de nível físico de forma a obter-se uma forma de simular o jogo em alto nível, sem referências a ES específicas. O mundo resultante será chamado de Mundo Abstrato de Jogo e servirá como base teórica para a simulação de mundo do motor proposto por esta dissertação.

Nas próximas subseções, procura-se montar uma simulação de jogo com um número mínimo de elementos de jogo adotados como primitivas. Os elementos serão combinados de forma a se ter relações suficientes para a construção de um

jogo digital. Este modelo está ilustrado em "*UGE in a Nutshell*" (GARCIA, 2014a) como parte da documentação do motor criado[42].

3.2.1 Elementos de Jogo, Camadas de Jogo e Usuário

Considerando um jogo digital qualquer, esta seção partirá das definições de ator (*Actor*), componente (*Component*) e evento (*Event*) como primitivas para a construção da simulação de um Mundo Concreto de Jogo. Estas três abstrações serão usadas como definidas no Capítulo 2. Como atores e componentes permitem a adição e a remoção de dados arbitrários e eventos permitem relatar e tratar acontecimentos, buscar-se-á definir relações entre três elementos para se definir o jogo.

Além dos elementos anteriores, serão considerados outros três elementos para a construção do Mundo Concreto de Jogo: duas camadas e o usuário. Estes elementos não considerados como primitivas – cada uma das primitivas elencadas afetará um ou mais destes elementos. Elas têm fins teóricos e didáticos para separar os interesses resultantes da combinação das primitivas.

A primeira camada a ser considerada é a Camada de Lógica (*Game Logic*). Esta camada é a responsável por controlar o comportamento do jogo, definindo regras que regem o comportamento dos atores. A segunda camada é a Camada de Visão[43] (*Game View*), responsável por apresentar a camada de Lógica ao usuário e permitir que este interaja com a Lógica. Por fim, o usuário (*User*) é o jogador humano que interage com o jogo.

3.2.2 Construindo um Mundo Concreto de Jogo

O Mundo Concreto de Jogo representa a simulação de jogo. Um novo Mundo Concreto de Jogo é criado completamente vazio (Figura 9); portanto, em seu início, não existem regras ou relações entre os elementos considerados. As regras e

[42] As figuras na versão impressa podem ser de difícil visualização. Recomenda-se, portanto, o acesso ao URL disponível na referência.
[43] Um nome mais apropriado para esta camada seria Camada de Apresentação em um primeiro momento; posteriormente, camada de Interação.

relações são definidas pelos designers ao longo do desenvolvimento – elas podem ser vistas como regras de negócio em sistemas convencionais.

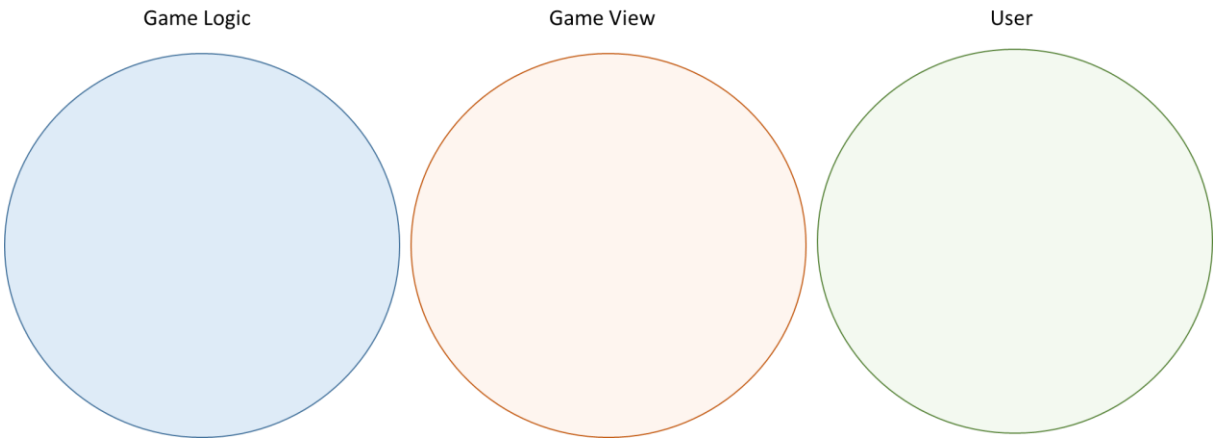


Figura 9. A criação do Mundo Concreto de Jogo.

O primeiro elemento a ser considerado em um Mundo Concreto de Jogo é o ator (Figura 10). O ator é qualquer personagem, objeto ou elemento de cenário que interage com o Mundo Concreto de Jogo. O ator é, portanto, um agente: ele age sobre o Mundo Concreto de Jogo. Considerando-se o modelo Entidade-Componente, o ator por si próprio não possui nenhum dado ou comportamento no jogo – ele apenas possui uma forma de ser identificado e diferenciado de outros.

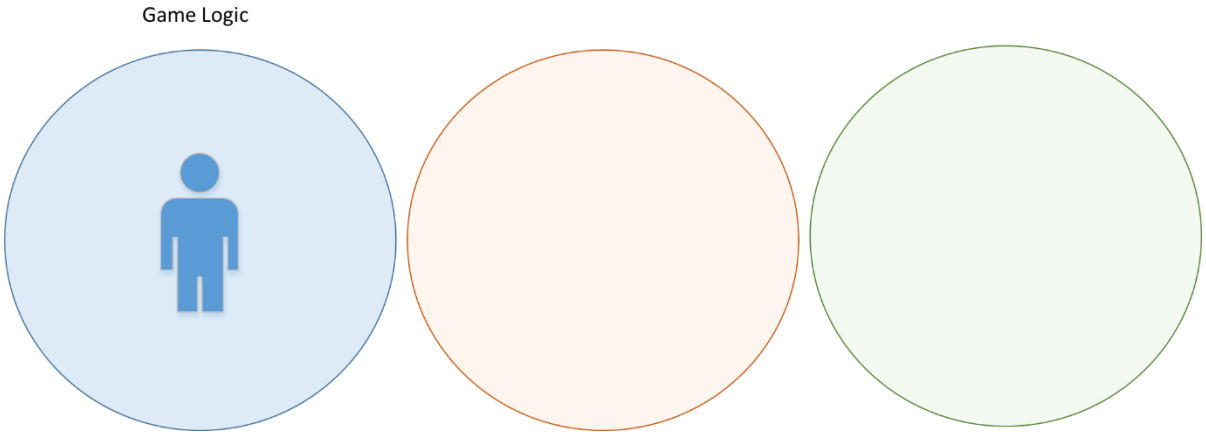


Figura 10. O ator.

Os comportamentos de um ator são definidos por uma coleção de componentes a ele agregados (Figura 11). Por exemplo, um ator com um `TransformableComponent` agregado passa a ter informações sobre sua posição, orientação e escala no Mundo Concreto de Jogo. Com isto, torna-se possível alterar sua posição, modificar seu tamanho ou orientação. Um ator com um `CollidableComponent` possui um corpo rígido para colisão. Desta forma, ele pode colidir com outros atores que também possuam um componente de colisão.

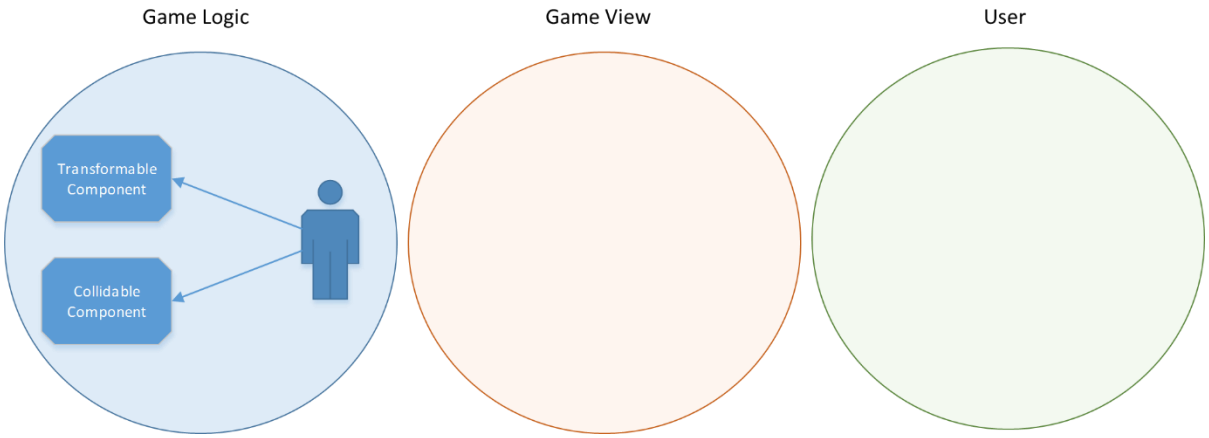


Figura 11. O ator com alguns componentes.

Por outro lado, um ator com um `DrawableComponent` pode ser desenhado na tela na posição definida por seu `TranformableComponent` (Figura 12). Já um ator com um `AudibleComponent` apresenta um som que o representa na posição do `TranformableComponent`.

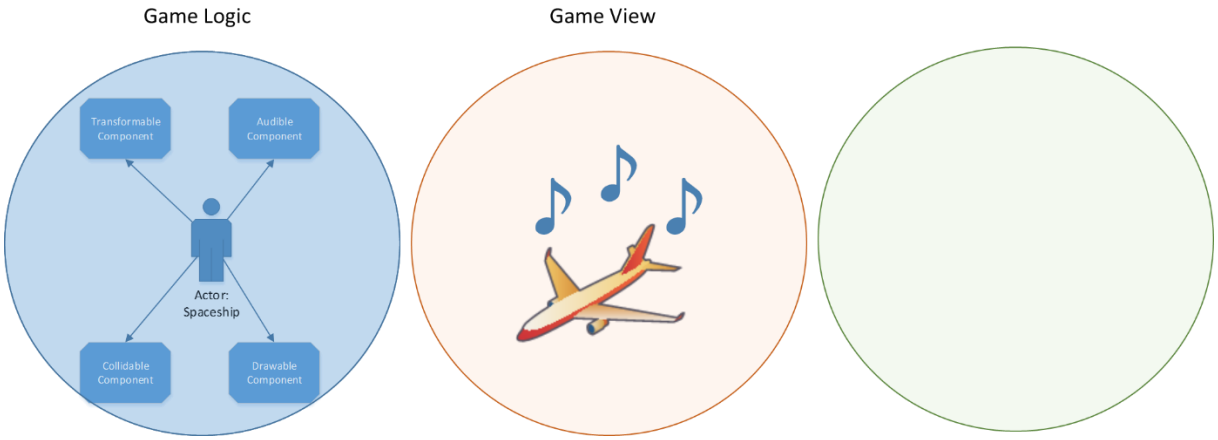


Figura 12. Um ator com componentes de saída.

É importante notar que, com a abordagem utilizada, os componentes de saída (`DrawableComponent` e `AudibleComponent`) são opcionais e não interferem na Camada de Lógica. Eles definem dados para respectiva apresentação modal do ator ao qual foram adicionados e, assim, apenas são usados pela Camada de Visão para apresentar o jogo ao jogador. Portanto, segue daqui que a Camada de Lógica independe e está segregada da Camada de Visão (Resultado I).

Um Mundo Concreto de Jogo possui múltiplos atores – em geral, de dezenas a milhares deles. O que diferencia os comportamentos de um ator de outro são seus componentes. No entanto, é importante observar que, mesmo que dois atores possuam os mesmos componentes, eles não são o mesmo ator. Cada ator é único

no Mundo Concreto de Jogo. Atores com os mesmos componentes apenas estão susceptíveis as mesmas ações – ou regras.

No Mundo Concreto de Jogo, existem duas possibilidades de ações entre atores: atores podem agir sobre atores ou o usuário pode agir sobre um ator. O comportamento de uma ação é regido por uma regra ou por uma mecânica de jogo – por exemplo, a gravidade ($\vec{g}$) na Figura 13. Regras são, portanto, relações que atuam sobre componentes e, indiretamente, sobre atores. O resultado de uma regra ou mecânica relevante é um evento (Figura 14).

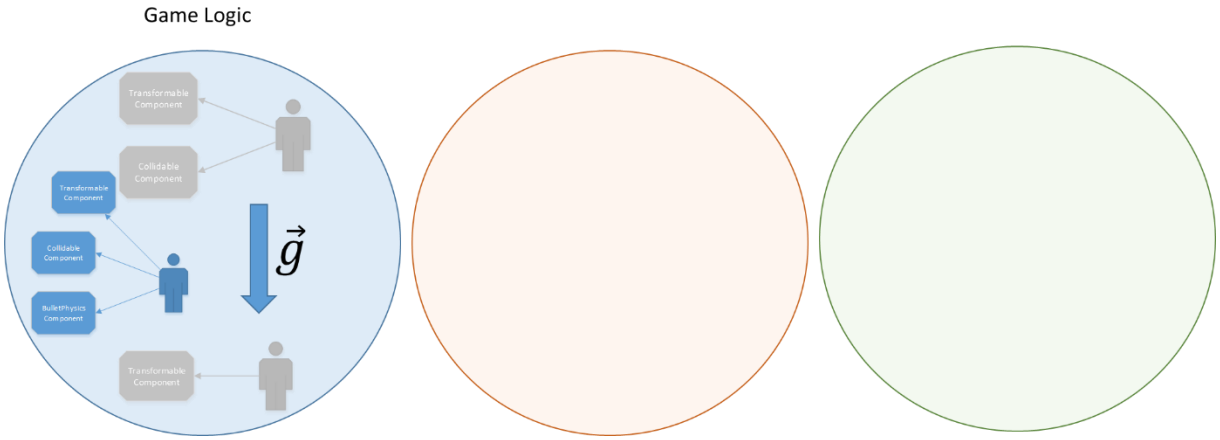


Figura 13. O Mundo Concreto de Jogos tem regras.

Por exemplo, a colisão é um aregra definida no Mundo Concreto de Jogo sobre pares de atores que possuam os componentes `TransformableComponent`, `CollidableComponent` e `PhysicsComponent`. Ao ocorrer uma colisão, os valores destes componentes são alterados – após uma colisão, os valores das posições e das velocidades dos atores envolvidos tendem a se modificar. O início da colisão é marcado por um evento (`PhysicalCollisionStarted`), assim como seu final (`PhysicalCollisionEnded`) (Figura 14).

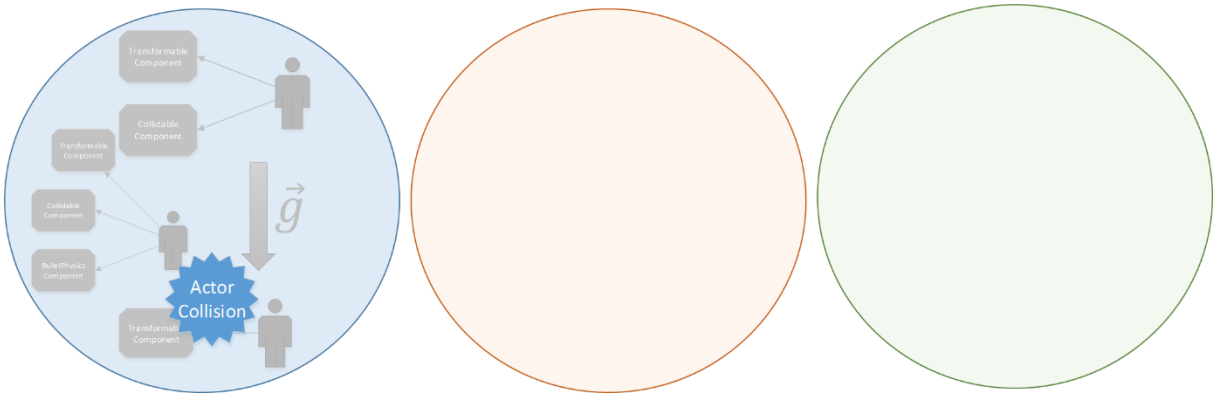


Figura 14. Interações entre atores são regidas por regras. Regras geram eventos.

Como o Mundo Concreto de Jogo contém as regras e regras podem tratar eventos, eventos podem desencadear novos eventos, fornecendo uma sequência de atualizações para o jogo.

Até este momento, tem-se uma simulação. O usuário não foi envolvido no jogo – ele apenas foi citado como possuidor da capacidade de agir sobre um ator. Assim como atores agem sobre atores, o usuário pode agir sobre atores por meio de dispositivos de entrada. Uma entrada pode ser tratada pela aplicação e alterar valores de componentes – por exemplo, ao pressionar o botão `'A'` em seu teclado, o usuário acelera seu ator (por exemplo, a nave da Figura 15).

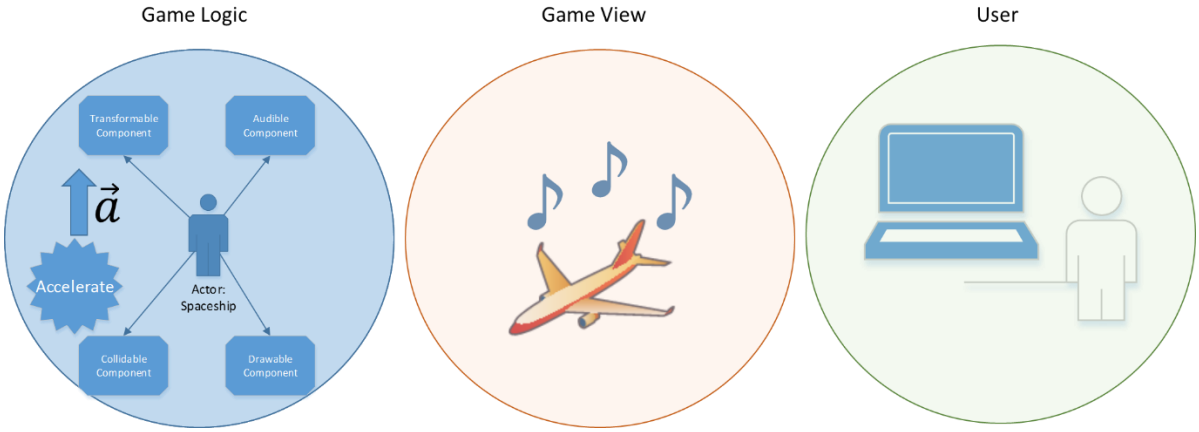


Figura 15. Uma ação externa ao jogo - a entrada de um comando pelo usuário.

Isto desencadeia uma alteração de aceleração do `PhysicsComponent` que, posteriormente, afetará a posição do `TransformableComponent`.

Pode-se estender este raciocínio para se definir quantas regras e mecânicas se desejar para o Mundo Concreto de Jogo – afinal, a Literatura descreve que é possível criar um jogo usando apenas eventos (GARCIA; NERIS, 2013a; GREGORY, 2009). De fato, é possível criar software usando exclusivamente (ou majoritariamente) eventos com arquiteturas *event-driven*.

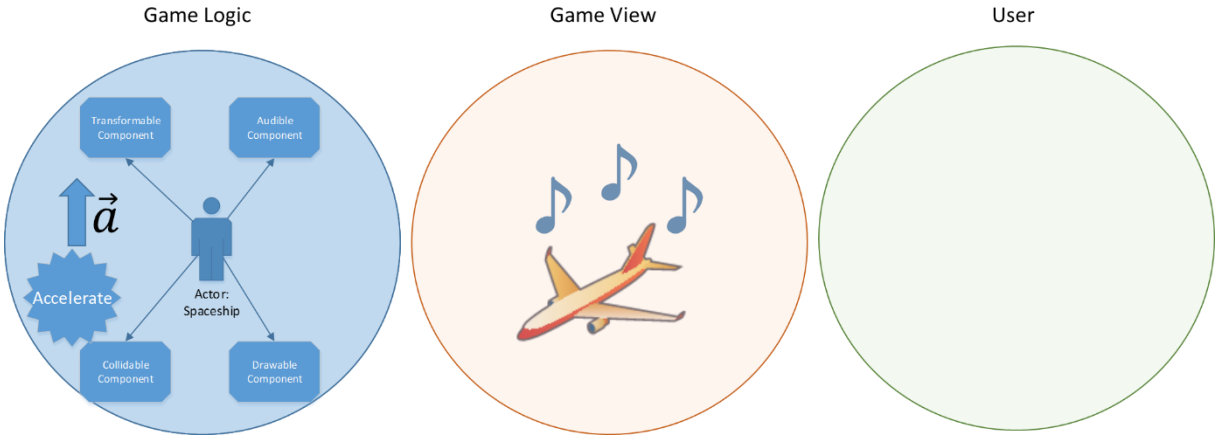
Esta subseção, portanto, definiu um Mundo Concreto de Jogo usando atores, componentes e eventos. Mostrou-se também que a Camada de Lógica, construída segundo este raciocínio, independe da Camada de Visão (Resultado I).

3.2.3 Removendo o Jogador do Mundo Concreto de Jogo

Até este momento, pelo Resultado I, o Mundo Concreto de Jogo é independente de apresentar de saída, dado que toda Lógica pode ser simulada sem

a necessidade de componentes de saída. Ou seja, atingiu-se metade dos objetivos propostos.

O usuário ainda interage com o jogo. Se houver um mecanismo capaz de remover a entrada do usuário do jogo, então o jogo será independente de entrada. A Literatura de desenvolvimento de jogos aponta o uso de eventos para a construção de agentes (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012) – isto também pode ser visto com o padrão *Command* (GAMMA et al., 1994). Agentes são atores providos de inteligência artificial (IA) que podem atuar sobre o mundo. Uma forma elegante de realizar a comunicação entre um agente e o jogo é por meio de eventos (MCSHAFFRY; GRAHAM, 2012).



Isto significa que um ator provido de IA poderia enviar comandos ao jogo por meio de eventos. Pode-se concluir, portanto, que, caso uma IA atue pelo jogador, um evento poderia substituir a entrada de um jogador para o jogo (Resultado II). Para facilitar a compreensão do restante do texto, eventos criados com a intenção de controlar um ator serão denominados comando de jogo.

Do Resultado I e do Resultado II, decorre que um Mundo Abstrato de Jogo pode utilizar apenas atores, componentes e eventos. Como apresentado ao longo desta seção, é possível combinar estes elementos ou alterar seus valores para definir o restante do jogo.

Atendendo aos interesses deste capítulo, os Resultados I e II removem a ES de um jogo, ou seja, remove-se o jogador do jogo. Efetivamente, isto transforma o jogo em uma simulação. Esta dissertação denominará o Mundo Concreto de Jogo sem usuário obtido como Mundo Abstrato de Jogo.

3.3 O Jogo Digital como um Sistema Formal

É interessante notar que, embora o processo adotado neste capítulo seja informal, a simulação do Mundo de Jogo Abstrato pode ser visto como um sistema formal[44] [45]. O processo apenas apresentou um mecanismo (evento) que, aplicado a jogos digitais, permitiu a obtenção de mundo digital genérico independente de ES – o Mundo Abstrato de Jogo.

Supostas estas condições, dada uma condição inicial e uma sequência de entradas, realizadas com os mesmos intervalos de tempo entre cada uma delas, o jogo deverá atingir um mesmo estado final[46].

Desta forma, pode-se observar um jogo como um sistema formal determinístico, que reage a um conjunto determinado de entradas para modificar seu estado. As regras de transição deste sistema formal são determinadas pela lógica do jogo, que define as regras. As entradas são eventos ou comandos de jogo. A saída é o próximo estado – e, portanto, não a apresentação sensorial do jogo.

A apresentação sensorial do jogo é um reflexo do estado atual do jogo – ela é como a captura de um instantâneo: uma foto, um quadro de som ou uma descrição feita por um observador externo.

3.4 Considerações Finais

Se uma árvore cair na floresta, mas não houver ninguém para ouvi-la, ela fez barulho? A famosa questão é um convite à Filosofia há séculos. Para os interesses desta dissertação, entretanto, a resposta é simples: se a Camada de Lógica simulou a queda da árvore, ela caiu. Se ela fez barulho ou não, isto vai depender de como a

[44] Isto é um axioma, dado que sistemas de software decidíveis devem ser executados em tempo não-infinito e podem ser simulados, na pior das hipóteses, em máquinas de Turing.
[45] Dada a existência de um mecanismo, tem-se um contra-exemplo de que a situação não é impossível. Pode-se argumentar que, por redução ao absurdo, é impossível não existir um mecanismo e, portanto, validar o mundo.
[46] Pode-se argumentar que jogos apresentam situações aleatórias. Esta é uma argumentação válida. Entretanto, convém ressaltar que algoritmos computacionais definem números pseudoaleatórios que, dada uma mesma semente, produzirão números na mesma sequência. Desta forma, considerar-se-ia também a(s) semente(s) para tornar o sistema determinístico.

Camada de Visão apresentará a situação ao usuário – e, portanto, de componentes e/ou do tratamento de eventos relacionados.

Neste capítulo, construiu-se um modelo a partir de uma análise sintética da Literatura de Desenvolvimento de Jogos para a construção de um Mundo Abstrato de Jogo e apontou três elementos primitivos que, combinados, podem simular todas as relações da Camada de Lógica de um jogo de forma a torná-la independente de ES.

Para isto, foi necessário remover o usuário do jogo. Por um lado, não existe mais jogo, apenas uma simulação. Por outro, obtém-se um novo paradoxo: um jogo que não possui usuários é acessível a todos os seus usuários e, portanto, universal.

Para a segunda parte desta análise, faz-se necessário reintroduzir o usuário ao jogo. Todavia, para os propósitos desta dissertação, isto deve ser feito de forma cuidadosa e especial: é necessário que as ES do jogo possam ser adaptadas ao usuário. Dado que o caso base, o Mundo Abstrato de Jogo é um jogo universal, se for possível reintroduzir o usuário de forma acessível, então o Mundo Concreto de Jogo obtido continuará a representar um jogo universal[47].

[47] A ideia é similar a um conjunto.

Capítulo 4

O JOGADOR ESTÁ NO JOGO E O JOGO ESTÁ NO JOGADOR

(... (n)o jogador está (n)o jogo está (n)o jogador está (n)o jogo...)

Jesse Schell (2008)

4.1 Considerações Iniciais

No Capítulo 3, introduziu-se o conceito de Mundo Abstrato de Jogo. Definiu-se o ator como objeto de jogo, o evento como protocolo/mecanismo de comunicação e o componente como o agregador, habilitador de comportamentos ao ator. Estes três elementos foram suficientes para construir a simulação de jogo.

Entretanto, pagou-se um preço inaceitável pelo resultado: para se obter uma simulação de jogo fechada, bem definida e universal, foi necessário remover o jogador do jogo. Naturalmente, um jogo sem jogador é inconcebível: o jogador é o elemento central de um jogo e a experiência de jogo provém do uso humano do jogo. De fato, a experiência de jogar está na mente de jogador (SCHELL, 2008).

Neste capítulo, procura-se reintroduzir o jogador ao jogo considerando-se suas necessidades de interação. Serão utilizadas as três primitivas de jogo e discutidas como cada uma delas pode ser explorada para auxiliar jogadores com algum problema de interação a jogar. Parte-se do pressuposto de que, se existir uma

combinação de atores, componentes e eventos que permita a um usuário com uma determinada necessidade de interação a jogar, então existe ao menos uma especialização para tornar o Mundo Abstrato de Jogo acessível ao jogador. Se isto acontecer, existe, portanto e teoricamente, uma forma de especializar as ES do jogo para torná-lo acessível ao jogador[48].

Desta forma, o pressuposto adotado neste capítulo é que, dados suficientes componentes de saída e tratamentos de evento, é possível especializar as ES de forma a atender às necessidades de interação de usuários do público considerado[49]. Se este for o caso então será, portanto, possível criar uma versão acessível do jogo a este público.

4.2 Um Mundo de Jogo Digital que Inclui o Jogador

Se, por um lado, o Mundo Abstrato de Jogo removeu o usuário da simulação, por outro ele apresenta possibilidades infinitas para a definição e especialização das entradas e saídas. A Camada de Lógica não é afetada por dispositivos de entrada nem de saída. Não importa se a árvore fez barulho ou não; ela caiu. Se isto é fato para a Camada de Lógica, isto é uma verdade incontestável para quaisquer possíveis Mundos Concretos de Jogo. Ou seja, existem infinitas formas de apresentar a simulação ao usuário e permitir a ele interagir com ela – tudo depende de como as ES forem definidas.

Existem infinitas formas de se interagir e de se observar um jogo. (N)O jogador está (n)o jogo e (n)o jogo está (n)o jogador (SCHELL, 2008). Definir como apresentar o Mundo de Jogo ao jogador é papel do designer: as ES encapsulam a Lógica de Jogo para fornecer uma experiência imersiva e acessível ao usuário.

Ou seja, especializando-se o Mundo Abstrato de Jogo em um Mundo Concreto de Jogo, não se obtém um jogo – é possível obter quantos jogos com a mesma lógica quanto se deseje. Isto sugere que o Mundo Abstrato de Jogo define

[48] Como as primitivas ator, componente e evento podem determinar o Mundo Concreto e o Mundo Abstrato de Jogo, então sugere-se que a simulação de mundo de jogo seja fechada para estas primitivas, ou seja, todas as relações resultantes do jogo podem ser criadas à partir dela.
[49] Isto decorre da observação anterior.

um Meta-Jogo: um molde, um modelo para a criação de jogos com as mesmas regras e mecânicas, nas quais o único elemento a ser alterado quando da inclusão de um usuário é a interação.

Portanto, dados suficientes eventos e componentes, o modelo proposto nesta dissertação permite a criação de infinitos clones de um mesmo jogo apenas alterando suas entradas e saídas. Do ponto de vista do usuário, todos são jogos diferentes. Do ponto de vista do desenvolvedor, existem infinitas formas de personalizá-lo[50].

As especializações recaem sobre as três primitivas de jogo: o ator, o componente o evento. Como o ator não representa nada por ele mesmo, componentes e eventos devem ser explorados para apresentar todo o jogo ao jogador. No Capítulo 3, considerou-se eventos ao nível da Lógica do Jogo, como protocolo de comunicação (comando de jogo) e como resultado da interação de atores.

Assumindo-se que esta especialização possa ser acessível a um público de jogadores, com suficientes iterações do design e na implementação das ES, obter-se-á um Mundo Concreto de Jogo acessível ao público considerado – o Meta-Jogo tornar-se-á um Jogo.

4.2.1 Saída: Especialização Modal para a Apresentação do Meta-Jogo

Com a abordagem adotada, o jogo pode ser apresentado ao usuário por qualquer representação sensorial, desde que exista ao menos um componente ou um evento que possa converter os dados em um estímulo. Para a saída, é permitido a Camada de Visão apenas ler valores de componentes usados pela Camada de Lógica. Deve-se, portanto, apresentar o jogo ao usuário sem modificações nos componentes da Lógica – o acesso aos dados da simulação deve ser feito de forma somente-leitura.

Por exemplo, se o ator possuir um componente para saída gráfica, ele será apresentado graficamente pelo subsistema gráfico por meio da representação armazenada no componente. É permitido ao sistema gráfico obter o valor da

[50] Um exemplo ilustrativo desta afirmação pode ser encontrado em: <https://github.com/francogarcia/uge/wiki/Game-Tech-Demo-Tutorial>.

transformação de mundo do ator correspondente para apresentar a saída, desde que os valores do componente não sejam modificados.

O mesmo vale para saídas explorando outros órgãos do sentido. Se o ator possuir um componente para saída aural, o subsistema de áudio reproduzirá o som armazenado no componente e usará quaisquer outros componentes de que precisar com permissão apenas de leitura. Como sempre é possível definir e adicionar um novo componente a um ator, a solução torna-se extensível. Por exemplo, caso se deseje fornecer saída táctil, pode-se criar um componente correspondente ao estímulo táctil desejado e definir-se um subsistema para tratá-lo.

Entretanto, a apresentação do jogo não se restringe aos componentes. Eventos também são uma primitiva do Mundo Abstrato de Jogo. As arquiteturas *event-driven* permitem definir tratamentos (por meio de *handlers*) para eventos.

Um evento pode possuir quantos tratamentos quanto se deseje. A Camada de Lógica trata os eventos segundo as regras definidas. A Camada de Visão pode apresentar os eventos sensorialmente ao usuário. Novamente, decidir a modalidade e o sentido a ser explorados é papel do designer para a especialização do jogo.

Com estas especializações, o Mundo Abstrato de Jogo torna-se Concreto em relação a saída: dependendo da criatividade do designer, o jogo pode se tornar observável, audível, tateável. Com o avanço da tecnologia, talvez até mesmo saboreável ou odorífero.

4.2.2 Entrada: Especialização Modal para a Interação com o Meta-Jogo

Como o mecanismo de comunicação do Mundo Abstrato de Jogo é um comando de jogo (evento), quaisquer seres que possam utilizar eventos como mecanismo de comunicação podem jogar. Ou seja, um ator controlado por uma IA pré-definida poderia jogar. Um arquivo de entrada contendo um *script* com comandos de jogo poderia jogar. Finalmente, o mais importante para esta seção: dado que exista uma tradução entre o valor obtido por um dispositivo de entrada para um comando de jogo, um ser humano pode jogar.

Da mesma forma, não importa se um comando de jogo é gerado por um ator dotado de IA, por um ser humano ou por geração espontânea. Se a Camada de Lógica recebe um comando de jogo, o evento é tratado e a simulação de jogo prossegue.

Convém lembrar que um comando de jogo é um evento. Ou seja, pela Seção 4.2.1, é possível tratá-lo na Camada de Visão para fornecer feedback sensorial ao usuário sobre sua ocorrência.

4.3 Necessidades e Habilidades de Interação

Com base na Seção 4.2 para entrada e saída, faz-se necessário elencar necessidades de interação às quais especializarão o Mundo Abstrato de Jogo do Meta-Jogo. Considerando-se um usuário ou um grupo de usuários com a necessidade de interação considerada, caso exista ao menos uma forma acessível para permitir a interação do usuário com o jogo, então é possível especializar o Mundo Abstrato de Jogo e especializar o Meta-Jogo em um Jogo para o público.

Para isto, considerar-se-á as habilidades de interação disponíveis na Literatura e elencadas nos jogos acessíveis consideráveis. Sempre que possível, será feita uma analogia com o modelo de interação descrito no Capítulo 2.

4.3.1 Usuário Médio

O usuário médio é o usuário considerado como padrão. Como jogos convencionais atendem às suas necessidades de interação, espera-se que exista ao menos uma possível especialização do Mundo Abstrato de Jogo que seja acessível a estes usuários.

De fato, o uso de eventos é uma estratégia empregada para o desenvolvimento de jogos convencionais (GREGORY, 2009; MCSHAFFRY; GRAHAM, 2012) e, assim como em software tradicional, o uso de arquiteturas *event-driven* podem ser exploradas para a construção de sistemas complexos.

4.3.2 Deficiência Visual

Os principais problemas de interação para usuários com deficiência visual estão compreendidos no Passo 1 do modelo de interação, ou seja, afetam o recebimento do estímulo. Para tornar um jogo acessível a estes usuários, é

necessário, portanto, que existam especializações para eventos e componentes de forma a permitir ao usuário compreender o estímulo.

Esta dissertação considera três casos:

a) Daltonismo;
b) Baixa visão;
c) Cegueira.

O daltonismo é uma deficiência visual que impede o usuário de diferenciar algumas cores – normalmente, usuários daltônicos possuem dificuldade em distinguir tons vermelhos de tons verdes. Uma estratégia utilizada por jogos para incluir usuários daltônicos é fornecer um modo acessível utilizando texturas ou imagens que não utilizem cores inadequadas às esses usuários (por exemplo, vermelho e verde) (ELLIS et al., 2013; YUAN; FOLMER; HARRIS, 2011).

O uso de componente permite resolver este problema. É possível definir um componente com saída gráfica que armazene uma textura acessível para especializar a saída. Desta forma, é possível criar uma especialização do Mundo Abstrato de Jogo acessível para usuários daltônicos.

Para a baixa visão, algumas estratégias exploradas são melhorias gráficas (como o uso de modelos com tamanho grande e de alto contraste), saída por fala, efeitos sonoros, feedback *haptic* e técnicas de sonificação (YUAN; FOLMER; HARRIS, 2011). O primeiro caso é abrangido pelo componente de transformação de mundo: um dos valores armazenados na transformação de mundo é a escala, que pode ser aplicada ao componente de saída gráfica para o desenho de um objeto com tamanho maior.

Para os demais casos, podem ser explorados tanto componentes quanto eventos. No caso de componentes, é possível adicionar um componente com saída sonora para a apresentação de um som 3D. No caso de evento, é possível definir-se *handlers* para tratar eventos relevantes e prover-se estímulos sonoros como feedback a ações. O feedback *haptic* segue o mesmo princípio.

Por fim, para a cegueira, além das estratégias não visuais da baixa visão, uma das estratégias utilizadas é a criação de uma apresentação do jogo totalmente aural, utilizando sons em espaços 2D ou 3D. No primeiro caso, exploram-se técnicas como som estéreo e *stereo panning* (INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2004; YUAN; FOLMER; HARRIS, 2011). No segundo caso, utiliza-

se sonificação (YUAN; FOLMER; HARRIS, 2011). Em ambos os casos, é possível construir a apresentação aural utilizando eventos – por exemplo, Top Speed 3 é um *audio-only game* que explora o uso eventos para prover feedback ao usuário[51] (PLAYING IN THE DARK, 2011).

Desta forma, dado que existe ao menos um *áudio-only game* construído empregando-se eventos, é possível concluir que existe ao menos uma possível especialização possível para o Mundo Abstrato de Jogo que possa ser acessível a usuários cegos.

Assim, dado que é possível utilizar eventos e componentes para tornar o jogo acessível a usuários com baixa visão, cegueira e daltonismo, conclui-se que é possível especializar o Mundo Abstrato de Jogo de forma a torna-lo acessível aos usuários com deficiência visual considerados.

4.3.3 Deficiência Motora

Para os usuários com deficiência motora, os principais problemas de interação referem-se ao Passo 3 do modelo de interação – o fornecimento da entrada. A Literatura descreve três estratégias principais para melhorar a interação de pessoas com deficiência motora: redução (*reduction*), automação (*automation*) e escaneamento (*scanning*) (YUAN; FOLMER; HARRIS, 2011).

A estratégia de redução remove parte da interação para simplificar o jogo – neste caso, a parte removida deixa de fazer parte do jogo. Esta é uma decisão de design para a implementação do jogo.

A estratégia de automação consiste em automatizar parte da interação para o usuário. Por exemplo, caso existam os comandos de jogo `Move` e `Jump`, poder-se-ia utilizar uma IA pré-defina para realizar um destes comandos ao invés do usuário, simplificando a interação.

Por fim, a estratégia de escaneamento divide-se em duas: escaneamento sensível ao contexto e escaneamento agnóstico de contexto. A primeira técnica alterna o comando de jogo a ser realizado dependendo dos atores próximos. A segunda consiste em alternar por todas as opções possíveis de uma lista finita de

[51] O projeto tornou-se de código-aberto em meados de 2013 (disponível em: <https://bitbucket.org/playinginthedark/top-speed-3>). No arquivo “Game.h”, é possível observar os eventos definidos e tratados pelo jogo.

itens. O escaneamento normalmente é utilizado por jogos que não dependem de tempo.

Para tornar um jogo acessível a pessoas com deficiência motora, deve existir uma forma de substituir as estratégias de automação e de escaneamento utilizando eventos e componentes. Como apresentado no Capítulo 3, a técnica de automação pode ser substituída por comandos de jogo controlados por uma IA pré-definida que possa atuar pelo usuário.

Resta o escaneamento. O escaneamento depende de um contexto – o contexto pode ser provido por uma regra, resultando em um evento. Isto é suficiente para o escaneamento sensível ao contexto. O escaneamento agnóstico de contexto requer alguns passos adicionais. Dada a ocorrência do primeiro evento, pode-se definir um `OrdenableComponent` para se definir uma sequência finita de atores a serem alternados em uma ordem específica. A transição entre cada uma das opções é marcada por um novo evento.

Dada a possibilidade de se substituir as técnicas para acessibilidade para usuários com deficiência motora, pode-se assumir que exista ao menos uma especialização possível para tornar o Mundo Abstrato de Jogo acessível a estes usuários.

4.3.4 Deficiência Auditiva

Assim como a deficiência visual, a deficiência auditiva acarreta em problemas de interação no Passo 1 (recebimento de estímulo) do modelo de interação. Yuan *et al.* (2011) afirmam que, como poucos jogos utilizam sons como o estimulo primário, a experiência de jogo de usuários com deficiência auditiva, embora seja comprometida, não é impossibilitada.

As estratégias comumente empregadas consistem de substituição por áudio por outro estímulo. A substituição pode ser feita por meio de texto (utilizando legendas e *closed captions*) ou por elementos não-textuais (como marcas visuais, radares de som e linguagem de sinais) (YUAN; FOLMER; HARRIS, 2011).

Substituições por meio não textuais exploram os mesmos recursos anteriores para serem decompostas em eventos e componentes: marcas visuais e linguagem de sinais podem explorar componentes gráficos e eventos; radares de som, componentes sonoros.

Substituições textuais não fazem, necessariamente, parte do Mundo Abstrato de Jogo – elas normalmente são externas ao jogo, sendo apresentadas em uma *Head-Up Display* (HUD). Caso se deseje usar eventos e componentes, segue-se o mesmo princípio do escaneamento independente de contexto.

Novamente, considerando-se que, pela argumentação anterior, existem opções válidas de decomposição das estratégias em eventos e componentes, é possível especializar o Mundo Abstrato de Jogo para usuários com deficiência auditiva.

4.3.5 Deficiência Cognitiva

De acordo com o modelo de interação, usuários com deficiência cognitiva apresentam problemas de interação relativos ao Passo 2 (determinação da resposta). Como afirmado por Yuan et al. (YUAN; FOLMER; HARRIS, 2011), as necessidades de interação deste perfil de usuário são complexas e variáveis.

Algumas recomendações gerais para aumento da acessibilidade para este nível são reduções de restrições de tempo para comandos de jogo, redução de estímulos sensoriais na tela e redução de entrada. A primeira pode ser obtida por componentes de movimento – basta-se reduzir a velocidade; a segunda explora componentes de saída, utilizando representações simples; por fim, a terceira segue os mesmos princípios da deficiência motora.

Como não existe uma estratégia que garanta uma solução acessível, o Mundo Abstrato de Jogo não pode ser especializado para todos os usuários com deficiência cognitiva e, portanto, quaisquer especializações possíveis serão restritas a grupos específicos de usuários. Entretanto, para os casos em que as recomendações gerais forem aplicáveis, pode ser possível a obtenção de, ao menos, melhorias paliativas na interação.

4.3.6 Múltiplas Deficiências

As seções anteriores consideraram uma única deficiência por usuário. Uma expansão futura do modelo pode começar a considerar mais de uma.

Assim como o Design Unificado apresenta uma análise das alternativas de design para a melhor interação, teoricamente, a proposta obtida também pode

combinar as estratégias identificadas para as deficiências envolvidas como possível forma de se permitir o uso do jogo.

A efetividade disto, entretanto, é tema para pesquisa futura.

4.4 Síntese das Estratégias

Analisando-se a Figura 6 apresentada no Capítulo 2 e considerando-se a argumentação deste capítulo, é possível perceber como componentes e eventos contribuem com cada uma das estratégias definidas. Componentes permitem apresentar o jogo sensorialmente (estratégias relacionadas estímulos) e alterar o valor dos dados (redução de restrições de tempo). Por outro lado, eventos permitem realizar a substituição de estímulos, a redução de estímulos e a substituição de entrada (redução, automação de entrada).

É interessante notar que, como o conjunto universo para a construção de um Mundo Abstrato de Jogo possui apenas as primitivas ator, componente e evento, então, se for possível converter qualquer uma das estratégias da Figura 6 em um conjunto não-vazio das primitivas, é possível permitir que o usuário interaja ou perceba sensorialmente um Mundo Concreto de Jogo correspondente.

Mais interessante ainda é notar que, se for possível criar um arquivo de configuração externo que permita ativar ou desativar estas mudanças, dado suficientes atores, componentes e eventos, é teoricamente possível configurar qualquer tipo de interação para especializar o Mundo Abstrato de Jogo de forma acessível. A construção do motor proposto por esta dissertação partirá, no próximo capítulo, deste pressuposto.

4.5 Considerações Finais

No Capítulo 3, agrupou-se as abordagens de Desenvolvimento de Jogos escolhidas no Capítulo 2 para a construção de um modelo de Mundo Abstrato de

Jogo capaz de definir toda a lógica de um jogo por meio de três elementos considerados como primitivos: o ator, o componente e o evento.

Neste capítulo, argumentou-se, informalmente, que é possível combinar os elementos primitivos para especializar o Mundo Abstrato de Jogo utilizando construções disponíveis na Literatura de Acessibilidade em Jogos para especializar o modelo definido. Isto sugere que é possível existir ao menos um Mundo Concreto de Jogo para quatro dos cinco perfis de habilidade de interação considerados: usuário médio, deficiência visual (baixa visão, cegueira e daltonismo), deficiência motora e deficiência auditiva. Ou seja, para estes grupos, deve existir uma forma de se criar um Jogo a partir do Meta-Jogo. Isto permitirá à implementação de um Mundo Abstrato de Jogo no nível da Camada de Lógica e diversos Mundos Concretos de Jogo no nível da Camada de Visão.

Infelizmente, não foi possível, neste momento, definir estratégias para incluir usuários com deficiências cognitivas. Como as estratégias identificadas na Literatura não provêm a garantia de acessibilidade, as possíveis especializações serão capazes de contribuir apenas parcialmente para melhorar a acessibilidade a esses usuários.

Capítulo 5

UGE: UM MOTOR PARA JOGOS DIGITAIS UNIVERSAIS

"Kindness is the language which the deaf can hear and the blind can see"

(A bondade/gentileza é a linguagem que permite aos surdos ouvirem e aos cegos enxergarem)

Mark Twain

5.1 Considerações Iniciais

Neste capítulo, explora-se a flexibilidade das abordagens elencadas no Capítulo 2 para apresentar um motor para jogos digitais universais. *UA-Game Engine* (UGE) (GARCIA, 2014c) é um motor para jogos universais de código-aberto[52], destinado ao desenvolvimento de jogos universais flexíveis e personalizáveis. UGE explora o conceito de *tailoring* para permitir aos desenvolvedores a criação de perfis de usuário destinados a atender às necessidades de interação dos usuários. Por meio dos perfis, é possível definir configurações gerais de jogo e alterar todas as especializações de entrada e saída para o jogo, definindo e personalizando a interação.

[52] O código-fonte do motor UGE está disponível em: <https://github.com/francogarcia/uge>.

Além de permitir o desenvolvimento de jogos, UGE serve como prova de conceito para este projeto, combinando as diferentes abordagens escolhidas de forma coesa e mostrando como cada uma delas contribui para o desenvolvimento de jogos universais. Por ser um projeto de código-aberto, outros desenvolvedores podem modificá-lo de acordo com suas necessidades ou usar o código como referência para a construção de um novo motor para jogos acessíveis.

Conforme será discutido ao longo deste capítulo, provê-se no motor UGE um modelo estruturado para a criação do Mundo Abstrato Jogo para permitir aos designers dedicarem-se à criação do jogo. Este modelo é concebido por meio da união de arquiteturas *data-driven*, *event-driven* e *entity-component* para possibilitar a criação de uma simulação de jogos independente de entrada e saída específica (ES – usando atores, componentes e eventos), com a comunicação mediada por um protocolo bem definido (o evento).

5.2 Partes Interessadas

O principal público-alvo para o motor UGE é constituído por designers e desenvolvedores de jogos. Um possível diagrama de partes interessadas está ilustrado na Figura 16[53].

[53] Um diagrama de partes interessadas (KOLKMAN, 1993; NERIS, 2010) permite o levantamento das partes envolvidas no uso de um sistema e indica como elas afetam o design do sistema. O diagrama é um artefato da Semiótica Organizacional, um referencial teórico que apresenta um conjunto de métodos e artefatos para a análise de sistemas de informação.

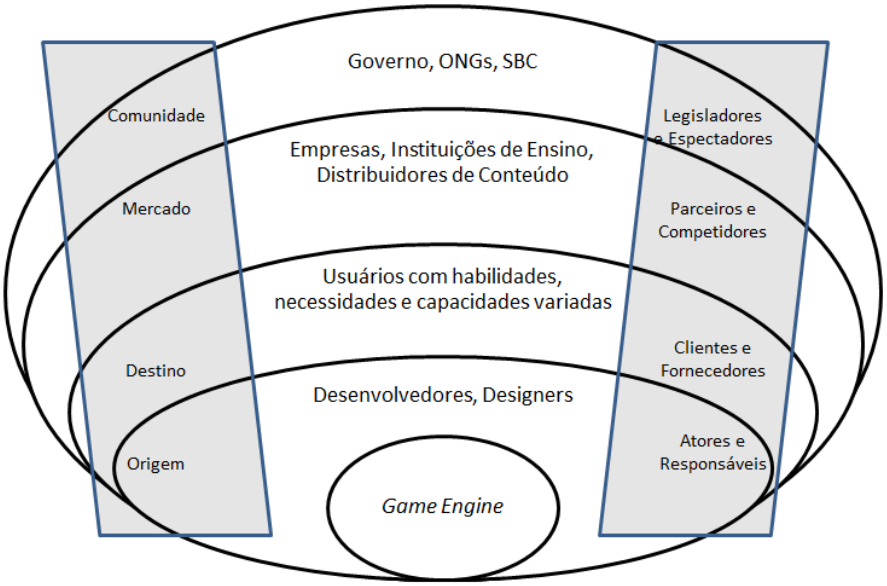


Figura 16. Diagrama de partes interessadas.

Pelo diagrama, pode-se observar que, embora em um primeiro momento apenas desenvolvedores e designers sejam beneficiados pelo motor criado, os artefatos resultantes do estudo conduzido ou do uso do motor podem beneficiar outras parcelas da sociedade.

Os estudos realizados podem auxiliar pessoas interessadas a desenvolver aplicações ou jogos mais acessíveis ou universais. Neste sentido, além de contribuir diretamente para o desenvolvimento de novos jogos, o motor pode contribuir com o 4º Desafio da Sociedade Brasileira de Computação – principalmente ao se considerar os artefatos resultantes do uso do motor (jogos).

O motor possibilita aos desenvolvedores a oportunidade de especializar a implementação para a adequar a usuários com diferentes capacidades e necessidades. Estes usuários compreendem pessoas normalmente impossibilitadas de jogar jogos convencionais, como usuários com deficiência visual. Assim, os jogos produzidos podem contribuir para a inclusão social e digital, além de fornecer uma opção de entretenimento a essas pessoas.

5.3 O Motor UGE

O motor UGE é implementado buscando-se permitir criar-se e definir-se os modelos de mundo definidos no Capítulo 3 e no Capítulo 4. Para o motor UGE, a

lógica é uma simulação independente de ES: ela é considerada sistema autônomo que reage a comandos de alto nível e gerencia alterações em seus dados.

Com isto, UGE permite a implementação de Meta-Jogos que podem ser especializados em Jogos por meio do uso de um Perfil de Interação, denominado *Player Profile* (perfil de usuário). O perfil de usuário contém as definições desejadas de ES para, em tempo de execução, especializar o Mundo Abstrato de Jogo de forma a transformá-lo em um Mundo Concreto de Jogo.

Isto permite aos desenvolvedores a possibilidade de prover uma experiência de jogo acessível e adequada aos usuários: o motor personaliza a entrada e saída para o jogo conforme a especificação do perfil, permitindo às interações especializaram definições estabelecidas da lógica do jogo para a obtenção do jogo final.

Ou seja, para o usuário, após escolhido o perfil desejado, a interação ocorre com um Jogo completo. Se existir um perfil adequado às suas necessidades de interação, tem-se a possibilidade de se oferecer um jogo acessível a ele.

Do ponto de vista do desenvolvedor, entretanto, existe um Meta-Jogo que pode ser especializado para outros perfis – quantos se desejar. Para isto, é necessário criar alternativas de especializações de ES aos atores, componentes e eventos definidos no Mundo Abstrato de Jogo. Ou seja, sempre que for possível especializar as primitivas de jogo segundo as necessidades de interação de um usuário, é possível criar um novo perfil de usuário para as necessidades consideradas. Se a especialização puder ser feita de forma acessível, então o jogo resultante também será acessível.

5.3.1 Considerações de Arquitetura e Design

O design e a arquitetura do motor UGE tem como objetivo prover flexibilidade para a prática de *tailoring* sobre a interação entre usuário e jogo em tempo de execução. Busca-se aumentar o potencial de personalização do jogo e a diminuição do tempo de iteração em detrimento de definições rígidas e dados imutáveis. Para atingir estes objetivos, diversas das decisões de design favorecem a flexibilidade e a adaptação do jogo em tempo de execução em detrimento do desempenho.

Algumas das principais inspirações para as decisões de design são os motores Unity[54], *Gameplay*[55] e GCC4[56] (MCSHAFFRY; GRAHAM, 2012; MCSHAFFRY, 2009). GCC4, proposto por McShaffry e Graham (2012)) é, inclusive, adotado como base para o motor UGE[57]. UGE reimplementa e estende o motor GCC4, aumentando suas funcionalidades e flexibilidades para um contexto de acessibilidade e de Design Universal.

Contrariamente à maioria dos motores de jogos disponíveis, UGE não tem como objetivo ter a melhor apresentação sensorial para uma determinada modalidade. De fato, os subsistemas de entrada e saída do motor são abstratos. Parte-se do pressuposto de que é mais importante possibilitar aos desenvolvedores a escolha das tecnologias mais adequadas para o público-alvo do jogo do que prendê-lo a uma específica. Os desenvolvedores podem implementar as interfaces abstratas disponíveis para cada um dos subsistemas e escolher a implementação desejada.

O motor UGE adota como dependências apenas projetos de código-aberto. Por exemplo, o motor usa Object-Oriented Graphics Rendering Engine[58] (OGRE) como o motor de renderização para o subsistema gráfico padrão (cobrindo OpenGL and DirectX); YSE[59] e TinyOAL[60] como opções para o subsistema de áudio (ambas utilizando OpenAL); e Object Oriented Input System[61] (OIS) para o subsistema de entrada. Entretanto, como mencionado anteriormente, o desenvolvedor é livre para implementar as interfaces disponíveis e escolher o subsistema desejado em tempo de execução.

5.3.2 Módulos e Camadas

Esta seção descreve a estrutura e a arquitetura dos módulos e das camadas do motor UGE. Usando uma abordagem *top-down*, esta seção inicia a apresentação

[54] Unity (Comercial). http://unity3d.com/
[55] GamePlay. https://github.com/blackberry/GamePlay
[56] GCC4. http://www.mcshaffry.com/GameCode/
[57] Embora tenha sido projetado para o desenvolvimento de jogos convencionais, esse motor contempla boa parte dos requisitos descritos ao longo deste projeto (em especial, modelo Entidade-Componente e arquitetura data-driven).
[58] OGRE. Available at: <http://www.ogre3d.org/>.
[59] YSE. Available at: <http://attr-x.net/yse/ >.
[60] TinyOAL. Available at: <https://code.google.com/p/tinyoal/>.
[61] OIS. Available at: <http://sourceforge.net/projects/wgois/>.

pelo motor propriamente dito e suas camadas. Pretende-se, assim, fornecer-se um panorama geral sobre as demais subseções e como suas funcionalidades funcionam em conjunto com o motor.

Os demais capítulos fornecerão mais detalhes sobre o funcionamento das principais funcionalidades dos módulos mais relevantes para o projeto.

5.3.2.1 UGE Engine

A estrutura de maior nível de abstração do UGE é o motor propriamente dito. O motor agrupa todas as funcionalidades da base de código em algumas classes, abstraindo e reduzindo a complexidade de desenvolvimento. O motor é responsável por gerenciar e por orquestrar a execução do jogo – incluindo o funcionamento de todas as funcionalidades, como eventos, física, lógica e apresentação. Para implementar um jogo, os desenvolvedores devem estender as classes disponibilizadas pelo motor e implementar a lógica e a apresentação do jogo.

Como ilustrado na Figura 17, o motor possui três camadas: Aplicação (`GameApplication`), Lógica (`GameLogic`) e Visão[62] (`GameView`). A implementação da lógica do jogo restringe-se à camada de Lógica (`GameLogic`), ao passo que a Camada de Visão (`GameView`) implementa as operações de ES segundo o perfil ativo.

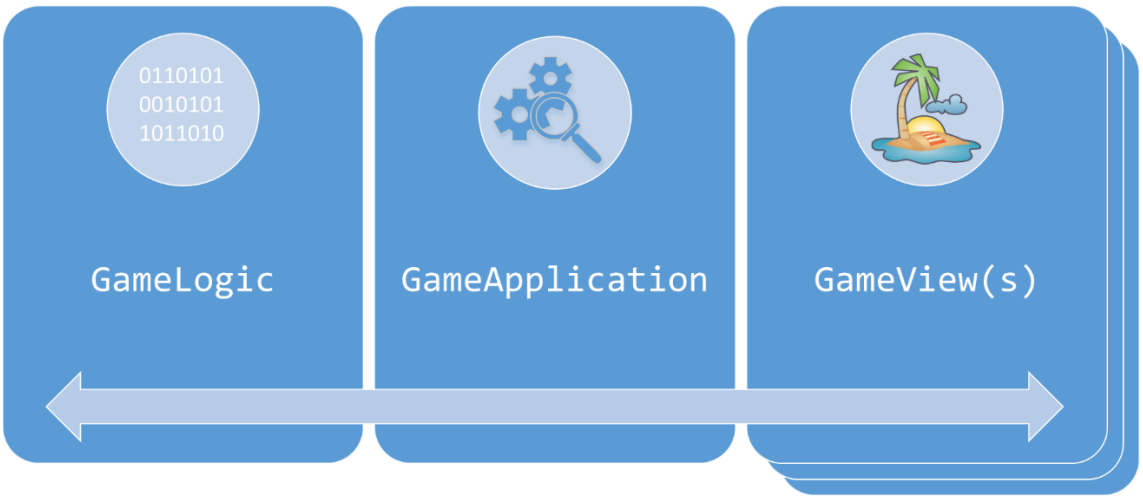


Figura 17. Camadas do motor UGE: Lógica, Aplicação e Visão.

A camada de Aplicação provê uma abstração entre o jogo e o sistema operacional do usuário. Ela cria e gerencia os subsistemas de entrada e saída, os

[62] Como destacado anteriormente, um nome semanticamente mais apropriado seria camada de Interação (*GameInteraction*). O nome camada de Visão será mantido ao longo do texto em referência à implementação.

recursos do jogo, os perfis de jogador e as outras duas camadas – sendo a Camada de Visão criada segundo o perfil desejado. A camada de Aplicação descreve e gerencia o laço de jogo, atualizando e apresentando o jogo e todos os seus subsistemas ao usuário em ordem apropriada.

A camada de Lógica provê uma abstração para a implementação do jogo. Nesta camada, o Mundo Abstrato de Jogo é simulado com todas as suas regras, lógica e comportamentos. A camada gerencia o ciclo de vida (criação, atualização e remoção) de entidades durante todos os *ticks* do jogo.

A camada de Visão abstrai a interação do usuário com o jogo. Ela define como o usuário percebe e interage com o jogo – portanto, controles, câmeras e dispositivos de entrada e saída são criados e escolhidos nesta camada.

O motor depende de todos os outros módulos da base de código – principalmente do núcleo de funcionalidades (UGE *Core*).

5.3.2.2 UGE IO

A camada de Visão provê a especialização para as ES necessárias para a interação do usuário com a camada de Lógica, carregando os subsistemas disponíveis no módulo UGE IO em tempo de execução, conforme definido no perfil.

O motor UGE não restringe as ES a frameworks, APIs, SDKs ou outros motores específicos para gerenciar a entrada e saída utilizada pelo jogo. Pelo contrário, permite-se aos desenvolvedores utilizarem aquelas que eles julgarem mais adequadas para o jogo – ou mesmo explorarem várias implementações para perfis diferentes. Para isto, é possível definir novos subsistemas concretos para o motor por meio da implementação das interfaces abstratas disponíveis (por exemplo, `IAudio` e `IGraphics` para os subsistemas gráficos e de áudio, respectivamente).

O módulo de IO depende dos módulos de utilidades (UGE *Utilities*) e do núcleo de funcionalidades (UGE *Core*). Além disso, depende também de quaisquer APIs, SDKs ou frameworks usados para a sua especialização.

5.3.2.3 UGE *Core*

O núcleo de funcionalidades (UGE *Core*) é o módulo que implementa a maioria das funcionalidades utilizadas nas camadas do módulo UGE *Engine*. As

funcionalidades são implementadas, em sua maioria, de forma modular, dependendo apenas do módulo *UGE Utilities*.

Algumas das funcionalidades do núcleo incluem:

- Ator e Componentes: primitivas do mundo de jogo;
- Cena: estrutura para facilitar a organização de posição de atores de forma hierárquica e facilitar a apresentação pela camada de Visão. O motor UGE fornece um grafo de cena independente de ES, facilitando a integração de tecnologias assistivas;
- Comandos de jogo: abstração para a interação com o usuário realizada por meio de eventos;
- Eventos: primitiva do mundo de jogo;
- Física: subsistema para simulação de física (dinâmica), operando sobre componentes relacionados a transformação de mundo, velocidade, aceleração e corpo rígido do ator;
- Perfil de usuário: o perfil de Interação instanciado para jogo;
- *Scripting*: integração de linguagens de *scripting*, como Lua, ao motor.

As funcionalidades disponíveis abstraem e/ou gerenciam o comportamento e a especialização do jogo em formas definidas e auto-contidas. Isto permite que a implementação do jogo seja realizada em alto-nível – é possível notar a ausência de referências a interações de ES neste módulo.

5.3.2.4 UGE Utilities

O último módulo do motor UGE provê funcionalidades úteis independentes de framework e plataforma. Este módulo é como uma caixa de ferramentas para uso geral, oferecendo abstrações para funcionalidades comuns como tipos, macros para depuração e *logging*, gerenciamento de arquivos, manipulação de *strings*, matemática e informações do sistema.

5.4 O Modelo de Mundo de Jogo

A construção do mundo de jogo em alto nível pode ser encontrada em UGE in a Nutshell (GARCIA, 2014a). Este documento sintetiza em alto nível os conceitos definidos e construído nos Capítulo 3 e Capítulo 4. A simulação de mundo é regida por regras definidas pelos designers. As regras agem sobre componentes específicos, ou seja, apenas afetam atores que possuem os componentes necessários para a regra. Por exemplo, o subsistema de física atua sobre atores que possuem `TransformableComponent`, `CollidableComponent` e `KinematicComponent`. Assim, um ator com estes componentes está sujeito à ação de regras como colisões, forças e impulsos.

O resultado de regras gera eventos, o que permite o desencadeamento de novas regras na camada de Lógica ou, posteriormente, fornecer-se feedback para o usuário por meio de especializações de evento. A comunicação entre atores e o mundo de jogo é regida por comandos de jogo (eventos).

UGE utiliza ambos os mundos de jogos definidos nos capítulos anteriores: o Mundo Abstrato de Jogo é utilizado pela camada de Lógica, ao passo que o Mundo Concreto de Jogo é usado pela camada de Visão. A construção do modelo foi explorada ao longo da Seção 3.2 para a obtenção informal do Mundo Abstrato de Jogo. A especialização do Mundo Abstrato para o Mundo Concreto de jogo adiciona componentes e eventos relacionados a ES – isto é realizado a partir um perfil de interação, um documento de texto que abstrai as necessidades de interação do jogador.

5.5 O Perfil de Interação: Player Profile

Diante da grande diversidade de capacidades e de necessidades de interação e da variedade demográfica dos usuários, um jogo universal deve ser altamente flexível, personalizável e adaptável. Como discutido nos capítulos anteriores, o uso de *tailoring* pode ser explorado para o desenvolvimento de jogos universais. É importante permitir que o jogo adapte-se às necessidades do usuário em tempo de

execução. Entretanto, para isto, é necessário informar ao jogo as alterações que devem ser realizadas.

Este é o objetivo do perfil de usuário oferecido pelo motor UGE. O perfil de usuário oferece uma descrição estruturada que permite modificar o jogo em tempo de execução. Um perfil de usuário pode ser criado para um usuário específico ou para um grupo de usuários (por exemplo, um perfil para uma necessidade de interação específica). Isto possibilita adaptar o jogo às necessidades do usuário e pode fornecer pré-definições acessíveis para a criação de um perfil personalizado.

Por exemplo, a Figura 18 apresenta a especialização de um Meta-Jogo para quatro diferentes perfis de usuário: (1) usuário médio, (2) baixa visão, (3) deficiência motora e (4) cegueira.

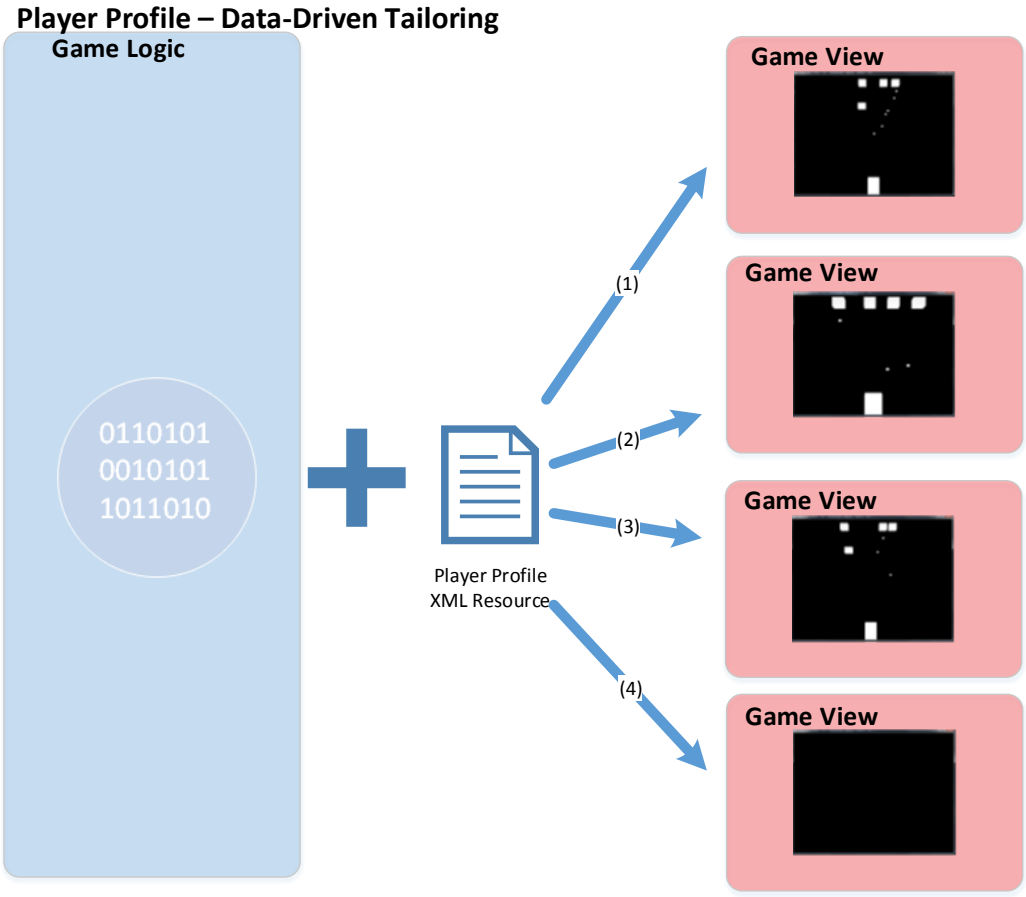


Figura 18. O perfil de usuário define as ES de um Meta-Jogo para torna-lo um Jogo acessível ao público para o qual foi concebido.

O perfil de usuário descrito pelo motor divide-se em dois: um conjunto de recursos externos – arquivos na linguagem de marcação *Extensible Markup Language* (XML) – e uma classe para acesso aos dados dos arquivos em tempo de execução (`PlayerProfile`).

O conjunto de arquivos XML explora a arquitetura *data-driven* do motor, permitindo aos desenvolvedores modificar os parâmetros que servem para a personalização do jogo. Por ser um recurso externo à implementação do jogo, é possível modificar o arquivo XML sem a necessidade de alterações no código-fonte. Como tudo no motor são dados, tudo pode ser definido em um arquivo de configuração externo e instanciado em tempo de execução – desta forma, é possível realizar a especialização da camada de Visão de forma flexível e personalizável.

A camada de Aplicação é a responsável por carregar os perfis disponíveis e gerenciar o perfil de usuário ativo. O perfil ativo é oferecido para as demais camadas adaptarem o jogo de acordo com as especificações definidas.

Um exemplo de uma lista de perfis está disponível na Listagem 1.

Listagem 1. Uma lista de perfis de jogadores.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayerProfiles resource="player_profiles/player_profiles.xml">

  <PlayerProfile name="Average User: Default"
                 resource="player_profiles/average_user/profile.xml"/>

  <PlayerProfile name="Visually Impaired: Blind"
              resource="player_profiles/visually_impaired/profile.xml"/>

  <PlayerProfile name="Perfil do Franco"
              resource="player_profiles/users/franco/profile.xml"/>

</PlayerProfiles>
```

A estrutura de um perfil é esquematizada na Listagem 2. Um perfil padrão contém quatro elementos: `GeneralSettings`, `InputSettings`, `OutputSettings` e `GameplaySettings`.

Listagem 2. Um exemplo perfil de usuário.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayerProfile name="Average User: Default"
            resource="player_profiles/average_user/profile.xml">

  <GeneralSettings>
    <Preferences
         resource="player_profiles/average_user/preferences.xml"/>
  </GeneralSettings>

  <InputSettings>
    <Mapping
        resource="player_profiles/average_user/input_mapping.xml"/>
  </InputSettings>
```

```
  <OutputSettings>
    <PrimaryOutput type="graphical"
        resource="player_profiles/average_user/graphics.xml"
        events="player_profiles/average_user/graphical_events.xml"/>
    <SecondaryOutput type="aural"
        resource="player_profiles/average_user/audio.xml"
        events="player_profiles/average_user/aural_events.xml"/>
  </OutputSettings>

  <GameplaySettings>
    <EntitySpecialization
      resource="player_profiles/average_user/entity/entities.xml"/>
    <EventSpecializations
      resource="player_profiles/average_user/events/events.xml"/>
  </GameplaySettings>

</PlayerProfile>
```

`GeneralSettings` contém um arquivo XML descrevendo configurações gerais para o jogo (por exemplo, idioma). Estas configurações são, em geral, usadas por todo o jogo. `InputSettings` descrevem as configurações para o dispositivo de entrada e mapeamento para os comandos do jogo, sendo usado pela camada de Visão. Isto permite definir, em tempo de execução, como o jogador interagirá com o jogo – por exemplo, caso sejam adicionados dispositivos ou tecnologias assistivas, basta alterar o valor do recurso de elemento para que, durante a interação, o jogo utilize os dispositivos escolhidos.

`OutputSettings`, por outro lado, contém os parâmetros para os dispositivos de saída e apresentação do jogo, também sendo aproveitado pela camada de Visão. Este elemento tem funcionalidade similar ao `InputSettings`: é seu equivalente para a saída, permitindo-se definir qual será o dispositivo de saída a ser utilizado para a apresentação sensorial do jogo. Novamente, podem ser explorados dispositivos padrões ou tecnologias assistivas.

Por fim, o elemento `GameplaySettings` descreve configurações para alterar partes especificas da lógica do jogo, como entidades e eventos. O elemento `EntitySpecializations` define como os atores de jogo serão especializados em tempo de execução – por exemplo, quais componentes de saída serão agregados a eles. O elemento `EventSpecializations` define quais *event handlers* serão ativados ou desativados em tempo de execução para prover feedback ao usuário.

Como será apresentado no Capítulo 6, o perfil de usuário é o recurso mais importante para a realização de *tailoring* em tempo de execução do motor UGE, dado que, a partir dele, é possível definir todas as entradas e saídas desejadas para o jogo, transformando o Mundo Abstrato de Jogo definido na camada de Lógica em um Mundo Concreto de Jogo na camada de Visão. Ou seja, transformando-se o Meta-Jogo definido em um Jogo.

5.6 Primitivas de Jogo e Especializações

O motor UGE utiliza apenas as três primitivas de jogo definidas no Capítulo 3 e no Capítulo 4 para a definição do Meta-Jogo. Todos eles são explorados de forma *data-driven* para permitir para explorar ao máximo os benefícios do perfil de usuário. Como sugerido nesses capítulos, qualquer combinação que possa ser convertida em atores, eventos e componentes pode ser usada para especializar as ES do jogo – como todas estas abordagens são exploradas de forma *data-driven*, então elas podem ser incorporadas ao perfil.

5.6.1 Atores e Componentes

O motor UGE utiliza a implementação de Atores descrito em (MCSHAFFRY; GRAHAM, 2012) com algumas modificações. O ator é um simples identificador (`ActorID`) com uma coleção de componentes. Ou seja, o ator por si próprio não é nada, mas pode se tornar qualquer coisa – isto só depende dos componentes escolhidos. Componentes, por sua vez, são conjuntos puros de dados que são processados por subsistemas específicos. Isto facilita o reuso.

A implementação permite a criação de atores a partir de um recurso externo – no caso, um arquivo XML – por meio de um padrão *Factory* (GAMMA et al., 1994). Por exemplo, o perfil da Listagem 3 é carregado pelo motor, resultando no ator da Figura 19 para a Camada de Lógica. Ou seja, para adicionar ou remover um componente, basta alterar a definição presente no arquivo XML.

Listagem 3. Um modelo de recurso XML para arquétipo atores.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Actor type="Spaceship"
 resource="data/config/player_profiles/average_user/entity/spaceship.xml">

  <TransformableComponent>
    <Position x="0.0f" y="0.0f" z="180.0f"/>
    <!-- YXZ order (yaw, pitch, roll) -->
    <Rotation yaw="3.1415f" pitch="0.0f" roll="0.0f"/>
    <Scale x="15.0f" y ="10.0f" z="30.0"/>
  </TransformableComponent>

  <CollidableComponent>
    <Shape type="Box">
      <Dimension x="1.0f" y="1.0f" z="1.0f"/>
    </Shape>
    <CenterOfMassOffset>
      <Position x="0.0f" y="0.0f" z="0.0f"/>
      <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
    </CenterOfMassOffset>
    <Density type="Pine"/>
    <Material type="Elastic"/>
  </CollidableComponent>

</Actor>
```

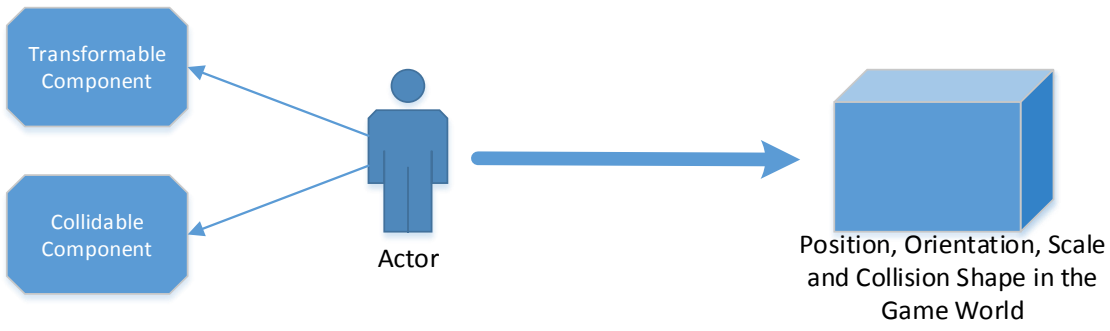


Figura 19. Um ator gerado pelo arquétipo da Listagem 3 para a camada de Lógica.

UGE explora esta funcionalidade para criar o Mundo Abstrato de Jogo usando um recurso sem componentes de ES. Em um segundo passo, a partir do recurso

definido pelo perfil de interação, os atores passam a ter seus componentes especializados para as ES. Isto é feito com um código simples e reusável, apresentado na Listagem 4. O código apresentado na listagem especializa um ator passado como parâmetro – para se especializar todos os atores do jogo, basta iterar por cada um deles.

Listagem 4. *Tailor* em tempo de execução para atores.

```
// Maps an actor's archetype to the resource that specializes it.
std::map<std::string, std::string> actorSpecializationResource;

void TailorActorToProfile(uge::ActorSharedPointer pActor)
{
    uge::ActorID actorID = pActor->GetActorID();

    std::string entityResourceFileName =
             actorSpecializationResource[pActor->GetActorType()];
    uge::XMLFile entityResource;
    entityResource.OpenFile(entityResourceFileName,
                            uge::File::FileMode::FileReadOnly);

    pGameLogic->vModifyActor(actorID, &entityResource.GetRootElement());
    pActor->PostInit();

    // Re-add the actor to the physics simulation (its transform or shape
    // might have changed).
    RemoveActorFromPhysics(actorID);
    AddActorToPhysics(pActor);

    entityResource.CloseFile();
}
```

Assim, por exemplo, se a Listagem 5 for usada para especializar o ator definido na Listagem 3, a representação do ator no Mundo de Jogo Concreto passar a ser como apresentado na Figura 20 – os componentes `DrawableComponent` e `AudibleComponent` são utilizados apenas pela Camada de Visão para, respectivamente, apresentar ao usuário estímulos gráficos e sonoros.

Listagem 5. Especialização para o arquétipo de ator da Listagem 3.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor type="Spaceship"
 resource="data/config/player_profiles/average_user/entity/spaceship.xml">

  <DrawableComponent>
    <NodeName n="Spaceship-Graphics" />
    <MeshFileName m="airplane.mesh" />
    <MaterialFileName n="airplane.material" />
  </DrawableComponent>

  <AudibleComponent>
    <NodeName n="Spaceship-Audio" />
    <FileName n="data/audio/effects/blip16.wav" />
    <Volume v="1.0f" />
    <InitialProgress p="0.0f" />
    <Loop l="true" />
  </AudibleComponent>

</Actor>
```

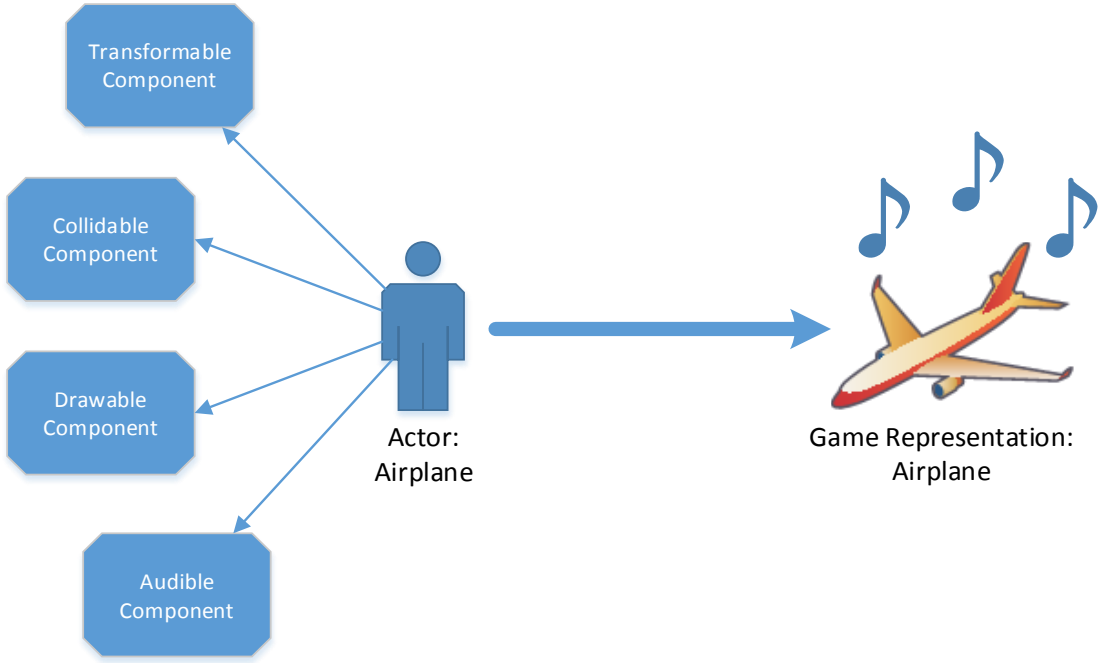


Figura 20. O ator da Listagem 3 para a camada de Visão.

Caso um mesmo componente seja redefinido pelo perfil, o motor sobrescreve os valores padrões para permitir uma configuração ainda mais granular do jogo – por exemplo, oferecer controle sobre a velocidade presente em componentes de física para melhorar a interação de usuários com deficiência motora ou cognitiva.

Ou seja, pode-se adaptar o jogo apenas alterando componentes das entidades já existentes na camada de Lógica – um perfil define os componentes mais adequados para a interação do usuário.

5.6.2 Eventos

Atores e componentes são um excelente recurso para desacoplar a camada de Lógica da camada de Visão. Entretanto, o uso de componentes normalmente não é suficiente para construir toda a camada de Visão – em geral, componentes são excelentes para a apresentação de estímulos contínuos, como gráficos. Infelizmente, para estímulos momentâneos, como sons, eles não são tão eficazes. Para estes casos, eventos são uma excelente alternativa.

O evento é um dos recursos mais polivalentes do motor UGE, sendo usado tanto para tratamento da camada de Lógica do motor quanto para a especialização das ES para o usuário. No caso da camada de Lógica, eventos são tratados para

gerenciar relações entre atores e comandos de jogo. No caso da camada de Visão, eventos são usados para permitir a especialização da entrada realizada pelo usuário e para prover feedback sensorial como saída a ele.

5.6.2.1 Eventos para a Saída

Para o feedback sensorial, a mesma abordagem adotada para atores e componentes é explorada para eventos. O motor UGE explora eventos de forma *data-driven*, permitindo aos desenvolvedores ativares ou desativarem *handlers* de acordo com as necessidades de interação do usuário. Por exemplo, a Listagem 6 explora este aspecto para eventos de saída.

Listagem 6. Eventos *data-driven* para a para a apresentação do jogo.

```
<?xml version="1.0" encoding="UTF-8"?>

<EventSpecialization resource=
      "data/config/player_profiles/average_user/events/aural_events.xml">

  <Events>

    <Event name="OnAlienDestroyed" enabled="false"/>
    <Event name="OnFireProjectile" enabled="false"/>
    <Event name="OnMoveActor" enabled="true">
      <FileName n="data/audio/effects/steps.wav" />
      <Volume v="1.0f" />
      <InitialProgress p="0.0f" />
      <Loop l="true" />
    </Event>
    <Event name="OnStopActor" enabled="true"/>

  </Events>

</EventSpecialization>
```

Desta forma, basta-se alterar um valor de texto para alterar-se o comportamento do jogo – tanto para a entrada, quanto para a saída. Ou seja, com o uso de eventos, pode-se redefinir diversas das saídas do jogo: se um *handler* é

ativado, ele passa a ser tratado pela camada de Visão e provê feedback sensorial; caso contrário, ele é ignorado.

Isto permite que sejam criados *handlers* para apresentar sensorialmente um mesmo evento – por exemplo, a Listagem 6 utilizou estímulos sonoros. Entretanto, poderiam ser providos, dentre outros: estímulos visuais, como efeitos de partículas e onomatopeias; hápticos, como *force feedback*; textuais, como transcrições ou legendas.

Explorando-se a arquitetura *data-driven*, isto possibilita que, para um dado perfil, sejam ativados os eventos mais relevantes para a apresentação sensorial considerando as necessidades de um usuário (Figura 21).

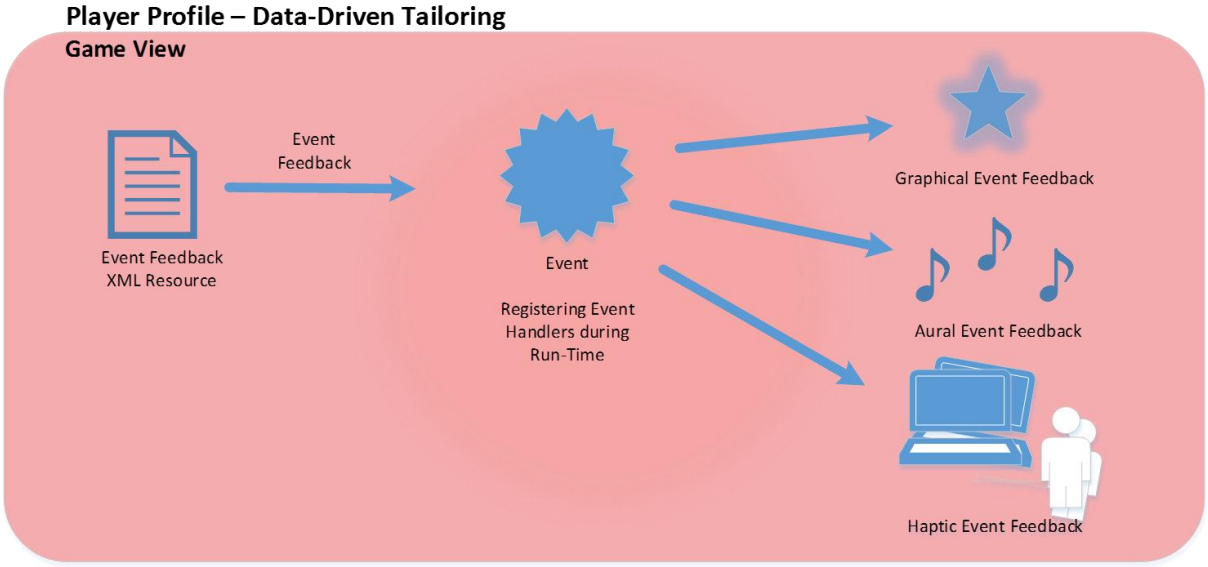


Figura 21. Especializando-se um evento para apresentação sensorial.

Ou seja, para diferentes necessidades de interação, pode-se escolher os tratamentos de evento mais adequados para prover a melhor experiência de jogo possível.

5.6.2.2 Eventos para a Entrada

Eventos são usados como o mecanismo de comunicação do motor UGE na forma de comandos de jogo. Isto promove uma forma unificada de comunicação tanto para o usuário quanto para todos os atores do jogo que realizam uma ação. Ou seja, tanto um usuário, quanto um ator com componente de IA utilizam um comando de jogo para desempenhar uma tarefa. Ou seja, isto permite que, caso o usuário

tenha dificuldades em realizar algum comando de jogo, sempre é possível simplificar a interação por meio de automação de um ou mais comandos de jogo (Figura 22).

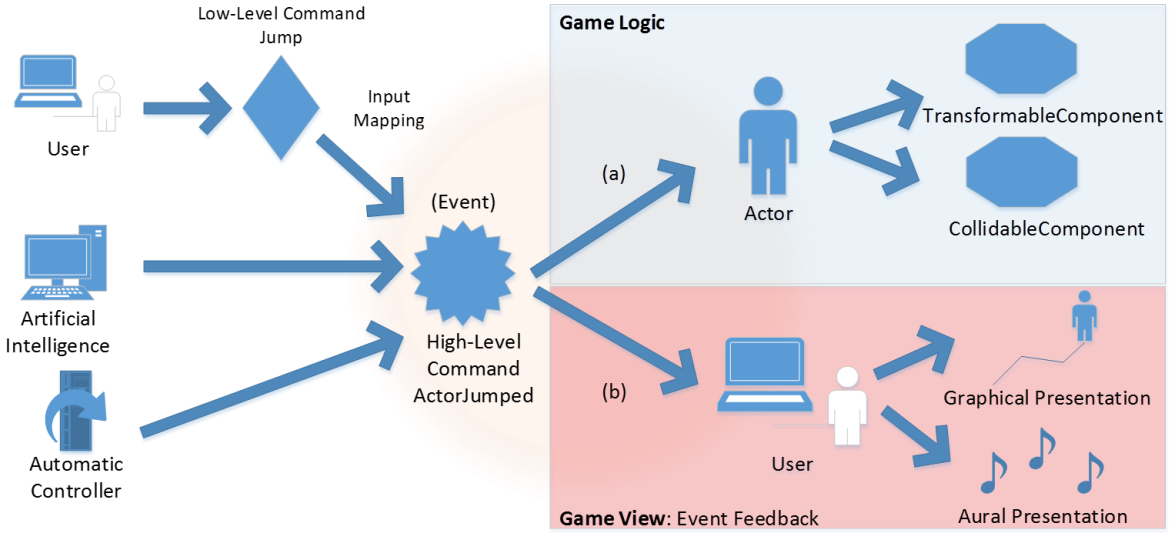


Figura 22. Um comando de jogo pode ser expedido tanto por atores, quanto por usuários.

Da mesma forma, isto simplifica o mapeamento de entrada: para que um dispositivo de entrada possa ser usado como controle do jogo, basta uma conversão dos valores obtidos do dispositivo para um comando de jogo (Figura 23).

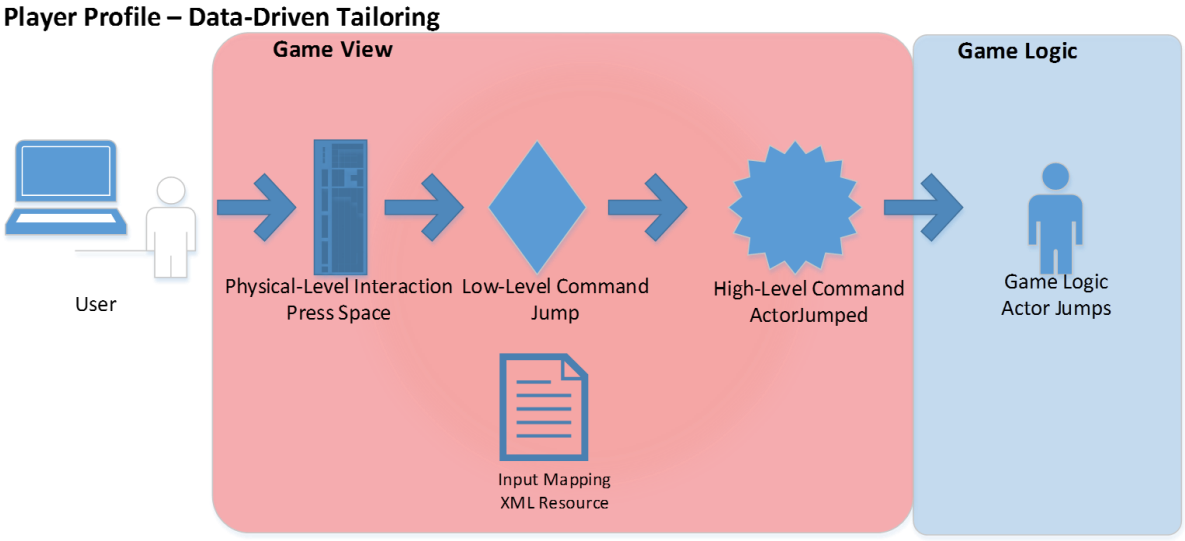


Figura 23. Especializando-se um evento para entrada.

Ou seja, além de se permitir usar dispositivos padrões, é possível integrar com maior facilidade tecnologias assistivas para a interação com o jogo. Da mesma forma, como um comando de jogo é um evento, é possível fornecer feedback para a entrada utilizando-se *handlers* – como definido na Seção 5.6.2.1.

5.7 Primitivas Coesas para Reuso e Acessibilidade

Assim, após implementada a camada de Lógica, a maioria das alterações para a inclusão de usuários ao jogo é realizada na camada de Visão – afinal, o Meta-Jogo é definido pela camada de Lógica e especializado por meio da adição de componentes e eventos para a entrada e saída na camada de Visão (Figura 24).

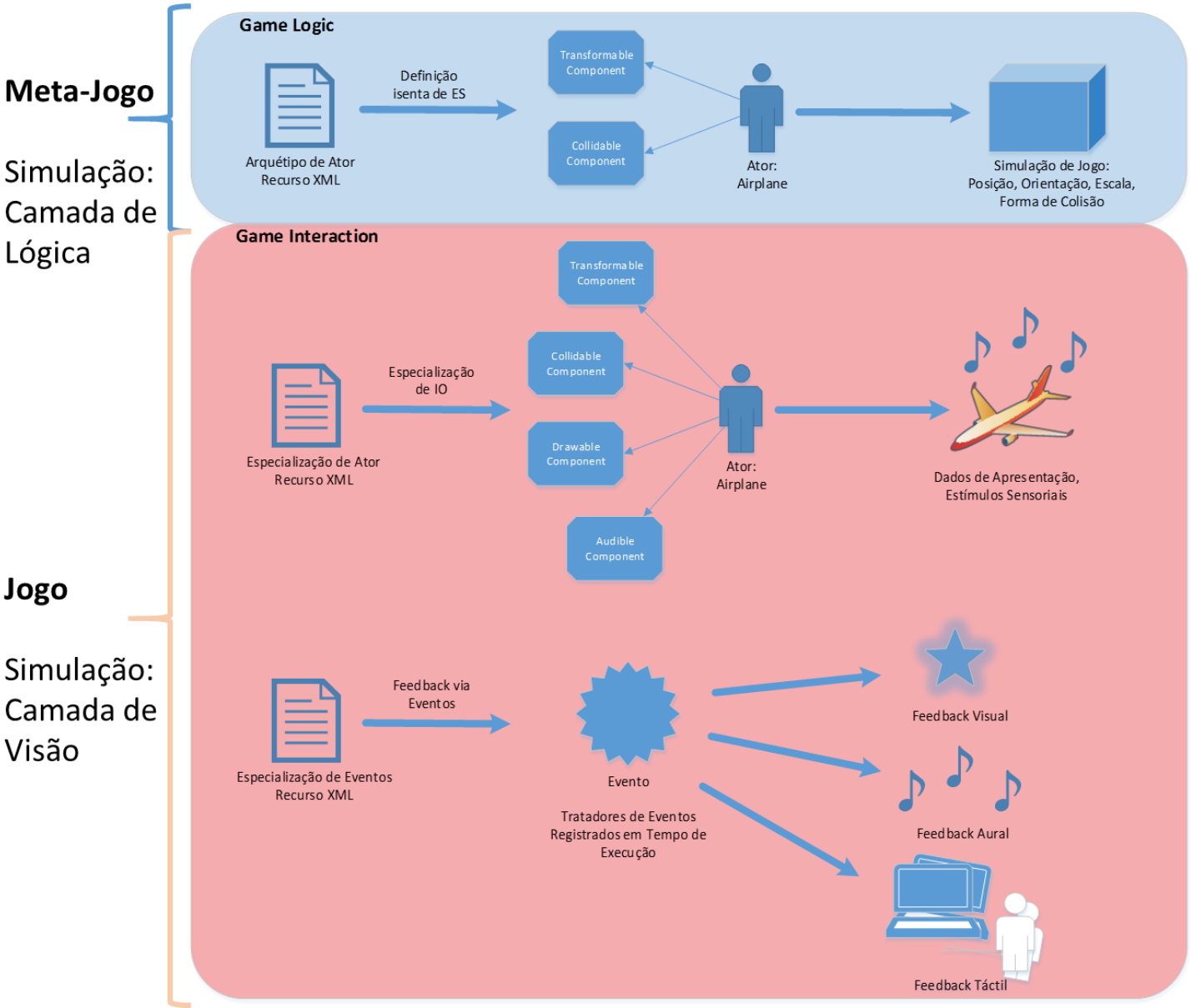


Figura 24. Toda a lógica de jogo é simulada na camada de Lógica. Dessa forma, a camada de Visão pode alterar a interação adicionando-se componentes e eventos para definir a interação.

Desta forma, a maioria das alterações realizadas para promover acessibilidade ao jogo para um determinado usuário ou grupo de usuários é

realizada por meio da implementação de novos componentes e tratamentos de eventos – como será apresentado com mais detalhes no Capítulo 6.

Como todas as especializações para a camada de Visão são *data-driven*, após a implementação da primeira especialização da camada, sempre é possível reusar componentes e eventos definidos anteriormente. Ou seja, todo novo perfil de usuário pode ser construído aproveitando-se as implementações de componentes e *event handlers* dos perfis anteriores. Sempre que é possível reusar um recurso já definido, nada é alterado no código – basta a inclusão do recurso desejado no perfil de usuário.

Além disso, em virtude da arquitetura *event-driven*, os eventos definidos na camada de Lógica atuam como guias para fornecer indícios daquilo que deve ser especializado.

Para o motor UGE, portanto, cada novo componente ou evento definido beneficia e melhora a experiência de jogo um ou mais usuários ou grupos de usuários. Isto permite aperfeiçoar o jogo de forma gradual e iterativa: incluir um perfil para uma nova necessidade de interação melhora o jogo para todos, pois fornece novas formas de se combinar opções de ES para melhor atender a necessidades individuais de usuários[63].

5.8 Processo de Implementação de um Jogo utilizando o motor UGE

Assumindo-se que a síntese realizada na Seção 4.4 seja válida, o processo de implementação de um jogo utilizando o motor UGE segue os seguintes passos para se criar um Meta-Jogo:

1. A camada de Lógica é implementada usando-se apenas atores sem componentes de entrada e saída;

[63] Não assume-se, entretanto, que todo jogo possa ter apresentações equivalentes considerando-se diferentes modalidades e sentidos. Sabe-se da existência de perdas decorrentes de diferentes percepções dentre os sentidos (BELLIK; BURGER, 1994; MILLER; PARECKI; DOUGLAS, 2007; PETRIE; MORLEY; WEBER, 1995). Logo, apresentações estritamente equivalentes em diferentes modalidades tendem, na maioria dos casos, a não existir.

2. Toda comunicação entre atores é realizada via eventos (comando de jogo);
3. Todo relacionamento (regra de jogo) entre um conjunto não vazio de atores resulta em um ou mais eventos;
4. A camada de Lógica trata comandos de jogo, quaisquer que sejam suas origens;
5. A camada de Lógica trata eventos relevantes para seus interesses;
6. Toda a camada de Lógica deve ser implementada seguindo-se os passos 1, 2, 3, 4 e 5.

Ao final dos primeiros passos, obtém-se a simulação do Mundo Abstrato de Jogo independente de ES ou usuário. Ou seja, ao final, obtém-se um Meta-Jogo que simula o *gameplay* desejado – tem-se um modelo para especializar um mesmo tipo de jogo para quaisquer ES.

Após construído o Meta-Jogo, é necessário especializar os atores e eventos de acordo com as necessidades de interação representadas por um perfil de usuário. Os resultados desta especialização manifestam-se na camada de Visão. Para isto:

7. A especialização é definida de acordo com um recurso externo (o perfil de usuário) armazenado em formato de texto. Todos os demais passos devem ser feitos em tempo de execução utilizando os dados do perfil;
    a. Caso se esteja criando o perfil, os resultados obtidos nos passos seguintes devem ser adicionados ao perfil em criação.
8. Adiciona(m)-se componente(s) de saída sensorial descritos no perfil de usuário para o(s) ator(es) do jogo de forma a obter-se estímulo(s) desejado(s) para o usuário.
    a. Se não existir um componente adequado, deve-se criá-lo;
9. Ativa-se ou desativa-se *event handler(s)* para evento(s) relevante(s) descrito(s) no perfil de usuário de forma a obter-se estímulo(s) desejado(s) para o usuário.
    a. Se não existir um *handler* adequado, deve-se criá-lo;

   b. Se não existir um evento relevante necessário, deve-se criá-lo e adicioná-lo à camada de Lógica. Não é permitido ao subsistema alterar componentes da camada de Lógica.
10. Define-se o mapeamento de comandos de jogo descritos no perfil de usuário para a realização da entrada e controle do ator que representa o jogador no Mundo Concreto de Jogo;
   a. Define-se uma tradução entre o(s) dispositivo(s) de entrada considerado(s) e os comandos de jogo necessário.
   b. Se for necessário reduzir-se o número de comandos de jogo, então utiliza-se um ator com componente de IA para realiza-lo pelo usuário.
11. Caso não existam subsistemas de ES adequados para gerenciar os componentes e/ou eventos criados, deve-se implementá-los;
   a. Neste caso, deve-se criar um subsistema de ES sensorial para processar o elemento (componente ou evento) e adicioná-lo à implementação. Não é permitido ao subsistema alterar componentes da camada de Lógica.
12. Toda a camada de Visão deve ser implementada usando-se os passos 7, 8, 9, 10 e 11. O resultado obtido é armazenado no perfil de usuário.

Ao final do processo, obtém-se um perfil de usuário que, ao ser lido e analisado em tempo de execução, configurará o jogo para atender às necessidades de interação do público para o qual foi concebido. Como os passos 7 a 12 podem ser realizados quantas vezes quanto se desejar, ao final de algumas iterações pode-se obter um resultado similar ao apresentado anteriormente na Figura 18. Como cada perfil pode ser construído usando-se componentes e *event handlers* já definidos, o processo tende a se tornar cada vez mais fácil e com a necessidade de implementação de poucas novas funcionalidades – em uma situação ideal, este número chegaria a zero e permitir a qualquer usuário jogar.

Desta forma, quando um jogo é iniciado, o Mundo Abstrato de Jogo é criado pelo motor pelos passos 1 a 6; em seguida, a especialização para o Mundo Concreto de Jogo ocorre a partir do perfil escolhido – o motor realiza os passos 7 a 12 para transformar o Meta-Jogo em um Jogo. Finalizado o processo, o usuário interagirá com o Jogo definido.

5.9 Considerações Finais

Desenvolver um jogo universal apresenta inúmeros desafios, envolvendo tanto o design, quanto a implementação. Ao passo que o Design Unificado e os trabalhos com recomendações contribuem para a realização de design de um jogo, poucos são os recursos disponíveis para auxiliar na implementação.

O motor UGE explora esta oportunidade. Este capítulo apresentou uma visão geral da organização do motor, descrevendo seus módulos e suas camadas. Ao final do capítulo, mostrou-se como componentes e eventos são explorados de forma *data-driven* para personalizar o jogo. O próximo capítulo apresenta um exemplo de uso do motor instanciado na implementação de uma demonstração para tornar as explicações mais concretas.

Deve-se notar que, sempre que possível, evitar-se-á detalhes de implementação para priorizar a discussão sobre as decisões tomadas para o desenvolvimento motor. Estes detalhes podem ser encontrados no código-fonte e são discutidos detalhadamente no guia de referência para desenvolvedores disponível em (GARCIA, 2014b).

Capítulo 6

TAILORING POR PERFIL DE INTERAÇÃO

UGE: Code once, ~~deploy everywhere~~ enable everyone.

Franco Eusébio Garcia, 20??

6.1 Considerações Iniciais

O Capítulo 5 apresentou uma visão geral sobre o motor UGE. Para melhor ilustrar como o uso de cada uma das funcionalidades do motor contribui para a criação de um jogo, este capítulo apresenta brevemente a construção de um jogo utilizando o motor. Desta forma, mostra-se como empregar o processo descrito na Seção 5.8 na prática e como o perfil de usuário definido na Seção 5.5 molda a transformação de um Meta-Jogo em um Jogo.

Esta abordagem torna mais clara a importância de cada elemento de jogo para a construção do Mundo Abstrato de Jogo e como este mundo é moldado pelo perfil de usuário para a obtenção do Mundo de Jogo Concreto. Ou seja, como realizar-se a implementação de um Meta-Jogo e como especializá-lo em um Jogo.

Recomenda-se que o acompanhamento desta seção seja realizado com a leitura da Seção 7.2 do UGE: *Developer's Reference* (GARCIA, 2014b)[64]. A seção

[64] Caso o leitor não tenha domínio sobre as arquiteturas *data-driven*, *event-driven* e entidade-componente, este capítulo provavelmente não fará sentido. Uma demonstração da aplicação está disponível em

recomendada descreve passo a passo a implementação do protótipo. A descrição deste capítulo será feita em maior nível de abstração.

6.2 Space Invaders

Space Invaders é um jogo do gênero de ação (subgênero *arcade*) desenvolvido por Tomohiro Nishikado e publicado para Taito Corporation (MOBYGAMES, 2014; WIKIPEDIA, 2014). O objetivo do jogo é proteger o planeta de uma invasão de extraterrestres.

Para isto, o usuário pode mover sua nave espacial em dois sentidos de uma mesma direção e atirar projéteis nos invasores para destruí-los. Caso os invasores atinjam a nave do usuário, o usuário perde parte de sua energia – caso a energia finde, o jogo é encerrado com derrota.

6.3 Design Unificado

O estudo de caso considera o resultado do Design Unificado para um clone do jogo Space Invaders disponível na Literatura (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007). O design a ser considerado está ilustrado na Figura 25.

<https://github.com/francogarcia/uge/wiki/Game-Tech-Demo-Tutorial>. Este protótipo é um *tech demo*. Portanto, não deve ser visto como um jogo – apenas como prova do potencial.
Recomenda-se fortemente o uso do protótipo e a personalização dos perfis alterando-se os arquivos XML conforme proposto. Com isto, é interessante notar que as abordagens adotadas pelo motor UGE reduzem as principais estratégias para aumento de acessibilidade em agregação de eventos aos atores ou à ativação/desativação de *event handlers*. Provido que eles já estejam implementados, isto permite que as melhorias de acessibilidade sejam realizadas apenas se alterando o perfil de usuário – sem modificações no código.
Para se verificar como as alterações nos perfis requerem poucas modificações no código, recomenda-se o acompanhamento das mudanças realizadas em 02/04/2014 disponíveis em <https://github.com/francogarcia/uge/commits/master>. Muitas destas modificações podem ser removidas com a aplicação de mais alguns padrões.

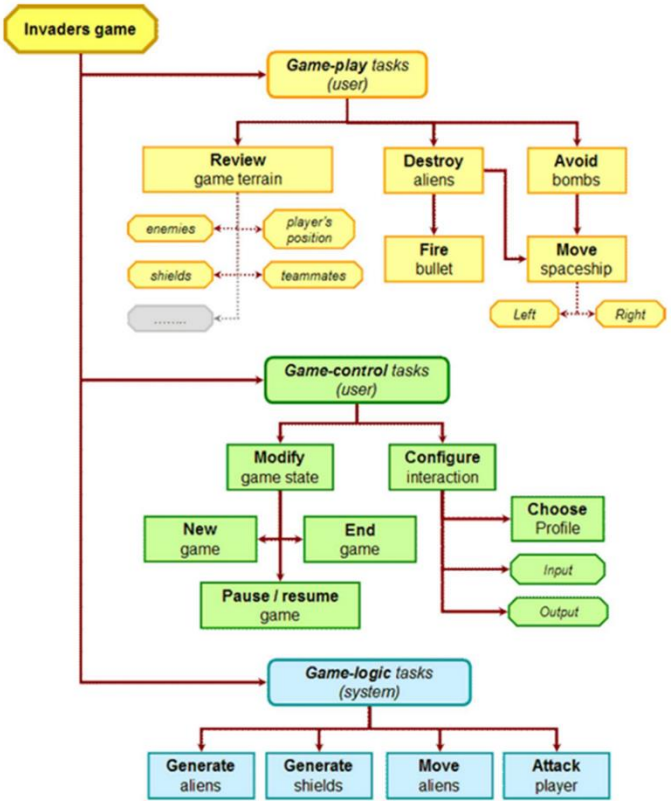


Figura 25. Design Unificado para um clone do jogo Space Invaders. (Extraído de (GRAMMENOS; SAVIDIS; STEPHANIDIS, 2007)).

Este design servirá como base para a implementação de um protótipo de alta fidelidade do jogo. A escolha baseia-se na possibilidade de ilustrar a aplicação do motor para anteder a diferentes necessidades de interação – cada especialização a ser apresentada na Seção 6.4 abordará uma ou mais funcionalidades do motor UGE.

6.4 O Meta-Jogo

Como descrito na Seção 5.8, o Meta-Jogo deve ser definido seguindo-se apenas os passos 1, 2, 3 e 4. Estes passos permitem a criação do Mundo Abstrato de Jogo, tornando possível a posterior especialização de acordo com as necessidades do usuário.

Nas subseções a seguir, apresenta-se a implementação do jogo em alto nível. Embora elas estejam aqui definidas em uma certa ordem, a implementação pode ser feita em ordem diferente. Ao final, obter-se-á a simulação do jogo sem ES.

6.4.1 Atores

O Mundo Abstrato de Jogo inicia-se vazio. Em um mundo vazio não há simulação e, consequentemente, não existe jogo. Para que exista um jogo, são necessários atores com interações mediadas por regras.

Analisando-se o design da Figura 25, pode-se identificar os seguintes arquétipos de atores para o jogo: `Spaceship`, `Alien`, `Bullet` e `Bombs`. Estas são, portanto, as entidades de jogo. Para o motor UGE, todas as entidades de jogo nascem iguais – é necessário defini-las com componentes.

6.4.2 Regras, Mecânicas e Eventos

A Figura 25 apresenta as seguintes ações explícitas: `Fire`, `Move`, `Generate`, `Avoid`, `Review` e `Attack`. Estas são atividades que devem ser realizadas pelo usuário ao jogar – elas definem os comandos de jogo.

Entretanto, existem regras implícitas que podem ser depreendidas das ações: `Fire` envolve a criação de um ator (de uma `Bullet` ou de uma `Bomb`). `Move` envolve uma translação da transformação de mundo (afetando todos os atores). `Avoid` também envolve uma translação, de forma similar a `Move`. Por fim, `Review` está relacionado à apresentação – caso se tenha um motor totalmente voltando a eventos, ele seria um evento importante. Entretanto, como o motor UGE utiliza para a camada de Visão para as ES, ele pode ser ignorado.

Realizando-se uma terceira análise, `Fire` e `Attack` sugerem a existência de energia para os atores. Isto sugere a necessidade de uma regra para verificação de energia e remoção de atores destruídos.

Assim, pode-se obter o seguinte possível conjunto inicial de regras e mecânicas: `FireProjectile`, `MoveActor`, `AvoidProjectile`, `AttackActor`, `CreateActor`, `DestroyActor`, `ApplyDamage`, `IsActorAlive` e `CheckCollision`. Algumas destas tarefas são realizadas apenas pela camada de Lógica (`CheckCollision`, `CreateActor`, `DestroyActor`, e `IsActorAlive`). Outras são comandos de jogo gerados por atores e tratados pela camada de Lógica (`FireProjectile` e `MoveActor`). As restantes são utilizadas pela IA (`AvoidProjectile` e `AttackActor`) para a posterior geração de comandos de jogo.

6.4.3 Componentes

Os componentes a serem definidos para o jogo estão intimamente relacionados com as regras e as mecânicas desejadas. Para a criação de componentes, deve-se analisar o conjunto obtido na Seção 6.4.2, considerar-se o design da Figura 25 e verificar-se quais dados são necessários. Neste momento, consideram-se apenas componentes livres de ES – ou seja, componentes livres de representações para estímulos sensoriais como imagens e sons.

Por exemplo, `FireProjectile`, `IsActorAlive`, `CheckCollision`, `ApplyDamage` e `DestroyActor` sugerem a existência de dano e energia quando de uma colisão. Portanto, deve-se criar um `HealthComponent` para armazenar a energia atual e a energia máxima, um `DamageInflictingComponent` para armazenar o dano resultante de uma colisão e um `DamageSoakingComponent` para reduzir o dano. A colisão supõe uma transformação de um corpo rígido e dados cinemáticos, sugerindo o uso de um `TransformableComponent`, de um `CollidableComponent` e de um `KinematicComponent` – e assim por diante.

O raciocínio estende-se para todas as demais regras e mecânicas. Um possível conjunto de componentes para a criação do Mundo Abstrato de Jogo é: `TransformableComponent`, `CollidableComponent`, `KinematicComponent`, `DamageInflictingComponent`, `DamageSoakingComponent`, `HealthComponent` e `AIScriptComponent`. Todos estes componentes possuem uma classe correspondente na implementação que armazena obtém dados em arquivos XML. Para este capítulo, o principal é o aspecto *data-driven* do motor – os dados de cada componente são apresentados na Listagem 7.

Listagem 7. Componentes para o perfil de usuário.

```
<TransformableComponent>
  <Position x="20.0f" y="0.0f" z="-180.0f"/>
  <!-- YXZ order (yaw, pitch, roll) -->
  <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
  <Scale x="10.0f" y ="10.0f" z="10.0"/>
</TransformableComponent>

<CollidableComponent>
  <Shape type="Box">
```

```
    <Dimension x="1.0f" y="1.0f" z="1.0f"/>
  </Shape>
  <CenterOfMassOffset>
    <Position x="0.0f" y="0.0f" z="0.0f"/>
    <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
  </CenterOfMassOffset>
  <Density type="Pine"/>
  <Material type="Elastic"/>
</CollidableComponent>

<KinematicComponent>
  <LinearFactor x="1.0f" y="1.0f" z="1.0f"/>
  <AngularFactor x="0.0f" y="0.0f" z="0.0f"/>
  <MaxVelocity v="15.0f"/>
  <MaxAngularVelocity v="0.0f"/>
</KinematicComponent>

<HealthComponent>
  <InitialHealthPoints value="100"/>
  <MaximumHealthPoints value="100"/>
</HealthComponent>

<DamageSoakingComponent>
  <InitialProtection value="50"/>
  <MaximumProtection value="100"/>
</DamageSoakingComponent>

<DamageInflictingComponent>
  <DamageOutput value="5"/>
</DamageInflictingComponent>

<AIScriptComponent>
  <ScriptObject constructor="AddActor" destructor="RemoveActor" />
  <ScriptData actorType="ActorType" />
</AIScriptComponent>
```

Na Listagem 7, é importante notar que os componentes são todos descritos em formato de texto. Ou seja, pode-se alterar qualquer um destes valores – em uma nova execução do jogo, o valor será refletido em tempo de execução. Internamente, o motor armazena os dados em uma classe; um objeto desta classe é agregado ao ator.

Por exemplo, alterando-se os valores do elemento `Position` do `TransformableComponent` de (20,0f; 0,0f; -180,0f) para (0,0f; 0,0f; 0,0f), ao se executar novamente a aplicação, o ator com este componente passaria a estar localizado na origem do mundo de jogo (pois, internamente, o valor teria sido pelo convertido motor em um objeto da classe da Listagem 8).

Listagem 8. A classe interna para o componente de transformação.

```
/**
 * @class TransformableComponent
 *  Component used to provide spatial position information to actors.
 */
class TransformableComponent : public ActorComponent
{
public:
    /// The name of the component.
    static const char* g_ComponentName;

    TransformableComponent();
    ~TransformableComponent();

    /**
     *  Instantiates the game object and loads the default data.
     * @param pInitXMLData : pointer to a XML element containing
     *             the object's initial position.
     * @return : true if the if initialization was successful;
     *             false otherwise.
     */
    bool vInit(XMLElement* pInitXMLData) override;

    /**
     *  Initializes the game object after it was instantiated.
```

```
     */
    void vPostInit() override;

    const std::string vGetName() const override;

    const Vector3 GetPosition() const;
    void SetPosition(const Vector3& position);

    const Quaternion GetRotation() const;
    const Vector3 GetRotationVector() const;
    void SetRotation(const Quaternion& rotation);

    const Vector3 GetScale() const;
    void SetScale(const Vector3& scale);

    const Matrix4& GetTransform() const;
    void SetTransform(const Matrix4& transform);

private:
    /// The position, rotation and translation of the actor
    /// this component is attached to.
    Matrix4 m_Transform;
};
```

Um ator definido com um ou mais destes componentes é criado por meio do padrão *Factory* pelo motor de jogo e pode participar da simulação do Mundo Abstrato de Jogo.

6.4.4 Criando-se o Mundo Abstrato de Jogo

Definidos os atores, os componentes e os eventos necessários, é possível implementar a lógica do Mundo Abstrato de Jogo. Para isto, implementa-se a classe `IGameLogic` ou `BaseGameLogic` do motor UGE. A primeira é uma interface, o que permite aos desenvolvedores reimplementar a camada de Lógica completamente,

caso desejarem. A segunda é uma implementação padrão, que pode ser estendida para a criação de jogos acessíveis. Desta forma, em geral, pode-se partir da `BaseGameLogic` para a criação do jogo usando as definições-padrão. Qualquer jogo criado utilizando o motor UGE pode tem um ou mais estados; um estado é denominado `GameState`. Por simplicidade, o restante deste capítulo assume que o estado do jogo apresentado seja `Running` (em execução).

Inicialmente, define-se os arquétipos dos atores – eles serão a base para a posterior especialização de componentes via perfil de usuário. A Listagem 9 apresenta um exemplo para o arquétipo `Spaceship`.

Listagem 9. O arquétipo de ator `Spaceship`.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor type="Spaceship" resource="data/game/actors/spaceship.xml">

  <TransformableComponent>
    <Position x="0.0f" y="0.0f" z="180.0f"/>
    <!-- YXZ order (yaw, pitch, roll), in radians. -->
    <Rotation yaw="3.1415f" pitch="0.0f" roll="0.0f"/>
    <Scale x="15.0f" y ="10.0f" z="30.0"/>

  <CollidableComponent>
    <Shape type="Box">
      <Dimension x="1.0f" y="1.0f" z="1.0f"/>
    </Shape>
    <CenterOfMassOffset>
      <Position x="0.0f" y="0.0f" z="0.0f"/>
      <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
    </CenterOfMassOffset>
    <Density type="Pine"/>
    <Material type="Elastic"/>
  </CollidableComponent>

  <KinematicComponent>
    <LinearFactor x="1.0f" y="1.0f" z="1.0f"/>
    <AngularFactor x="0.0f" y="0.0f" z="0.0f"/>
```

```
    <MaxVelocity v="15.0f"/>
    <MaxAngularVelocity v="0.0f"/>
  </KinematicComponent>

  <HealthComponent>
    <InitialHealthPoints value="100"/>
    <MaximumHealthPoints value="100"/>
  </HealthComponent>

  <DamageSoakingComponent>
    <InitialProtection value="50"/>
    <MaximumProtection value="100"/>
  </DamageSoakingComponent>

</Actor>
```

Pela Listagem 9, qualquer ator criado pela camada de Lógica a partir deste arquétipo terá um `TransformableComponent`, um `CollidableComponent`, um `KinematicComponent`, um `HealthComponent` e um `DamageSoakingComponent` – todos estes componentes são isentos de ES[65]. O mesmo se aplica aos demais arquétipos:

- `Bomb` e `Bullet` possuem: `TransformableComponent`, `CollidableComponent`, `KinematicComponent`, `DamageInflicting Component`;
- `Alien` possui: `TransformableComponent`, `CollidableComponent`, `BulletPhysicsComponent` e `HealthComponent`.

Internamente, o ator da Listagem 9 é convertido pelo motor em um objeto da classe `Actor` (Listagem 10).

Listagem 10. A classe `Actor` para o motor UGE.

```
/**
 * @class Actor
 *  Defines actors to be used by the game.
```

[65] Novamente, é importante ressaltar que todos os atores são construídos como texto e carregados em tempo de execução. Isto significa que um ator construído com os componentes da Listagem 8 terá os valores iniciais para os atributos de jogo definidos como os atributos do arquivo de texto.

```
 */
class Actor
{
    /// Actors can only be created by the ActorFactory class.
    friend class ActorFactory;

public:
    static const ActorID NULL_ACTOR_ID = 0u;

    /**
     *  Constructor.
     * @param id : the identifier which will be used for the actor.
     */
    explicit Actor(const ActorID id);

    /**
     *  Destructor.
     */
    ~Actor();

    /**
     *  Instanciates the game object.
     * @param pInitXMLData : pointer to a XML element containing
     *          the object's load data.
     */
    void Init(XMLElement* pInitXMLData);

    /**
     *  Initializes the game object after it was instantiated.
     */
    void PostInit();

    /**
     *  Destroys the object, freeing the memory used by the actor.
     */
    void Destroy();
```

```
/**
 *  Returns the actor's unique identifier.
 * @return : the actor ID.
 */
const ActorID GetActorID() const;

/**
 *  Fetches the actor's archetype.
 * @return : a string containing the name of the actor.
 */
const std::string GetActorType() const;

/**
 *  Retrieves a components associated with this actor by
 *  its identifier.
 * @return : a pointer to the requested component.
 */
template <class ComponentType>
std::weak_ptr<ComponentType> GetComponent(
                                    const ComponentID componentID);

/**
 *  Retrieves a components associated with this actor by its name.
 * @return : a pointer to the requested component.
 */
template <class ComponentType>
std::weak_ptr<ComponentType> GetComponent(
                                    const std::string& componentName);

/**
 *  Gets all the components associated to the actor.
 * @return : a pointer to a map containing all the components
 *        of which the actor is composed.
 */
const ActorComponentMap* GetActorComponents();
```

```
    /**
     *  Converts all the data of the actor object to XML.
     * @return : a string with the XML data.
     */
    std::string SerializeActorToXML();

private:
    /**
     *  Adds a new component to this actor.
     * @param pComponent : a pointer to the new component.
     */
    void AddComponent(ActorComponentSharedPointer pComponent);

    /// Unique identifer for an actor object.
    ActorID m_ActorID;
    /// The name of the actor archetype.
    std::string m_ActorType;
    /// The components that are attached to the actor.
    ActorComponentMap m_ActorComponents;
    /// The name of the file this actor was loaded from.
    std::string m_ActorResourceFileName;
};
```

Na Listagem 10, é importante notar que os únicos atributos do ator são seu identificador, nome do arquétipo, uma coleção de componentes[66] e o arquivo gerador. Ou seja, por meio de uma *Factory*, o arquivo de arquétipo contido na Listagem 9 criaria um ator com os componentes e valores iniciais estabelecidos no recurso XML. Para alterar quaisquer valores destes valores, basta alterar o recurso – nenhuma mudança é necessária no código.

[66] O ator tem um componente – relação *has-a*. Não é necessária, portanto, uma classe para os arquétipos de atores – basta adicionar os componentes desejados para se criar o ator desejado. Por exemplo, uma Pessoa é um Ator que tem um nome, data de nascimento, etc. Tudo em um ator são componentes adicionados em tempo de execução. Componentes são dados.
O leitor deve estar ciente disto para prosseguir a leitura – a especialização do jogo adiciona novos componentes aos atores do Mundo de Jogo Abstrato para lhes tornar Concretos.

As regras (`CheckCollision`, `CreateActor`, `DestroyActor`, e `IsActorAlive`) realizadas apenas pela camada de Lógica tornam-se métodos de um estado de jogo pertencente à classe `GameLogic`. Elas operam sobre os dados dos componentes necessários para a implementação do jogo. Por exemplo, `DestroyActors` pode gerar o método `RemoveDestroyedActors()`, que opera sobre `HealthComponents` e remove do jogo atores destruídos (Listagem 11).

Listagem 11. Implementando-se uma regra de jogo.

```
void Running::RemoveDestroyedActors()
{
    std::vector<uge::ActorID> destroyedActorIDs;

    sg::GameLogic* pGameLogic =
                    dynamic_cast<sg::GameLogic*>(m_pGameLogic);
    for (const auto& actorIt : pGameLogic->m_Actors)
    {
        uge::ActorID actorID = actorIt.first;
        uge::ActorSharedPointer pActor = actorIt.second;
        std::weak_ptr<Component::HealthComponent> pComponent =
                pActor->GetComponent<sg::Component::HealthComponent>(
                      sg::Component::HealthComponent::g_ComponentName);

        // Check if the actor has a HealthComponent.
        if (!pComponent.expired())
        {
            // Provided it does, check whether it is alive.
            Component::HealthComponentSharedPointer pSharedComponent =
                                                        pComponent.lock();
            if (pSharedComponent->GetHealthPoints() <= 0)
            {
                destroyedActorIDs.push_back(actorID);
            }
        }
    }

    // Remove dead actors from the game world simulation.
```

```
    uge::IPhysicsSharedPointer pPhysics = m_pGameLogic->vGetPhysics();
    for (uge::ActorID actorID : destroyedActorIDs)
    {
        pPhysics->vRemoveActor(actorID);
        pGameLogic->vRemoveSceneNode(actorID);
        pGameLogic->vDestroyActor(actorID);

        // Dispatch an event.
        std::shared_ptr<sg::AlienDestroyed> pEvent(
                                LIB_NEW sg::AlienDestroyed(actorID));
        uge::IEventManager::Get()->vQueueEvent(pEvent);
    }
}
```

É importante notar que, ao final da Listagem 11, é expedido o evento `AlienDestroyed`. Isto será explorado, mais tarde, pelo perfil de usuário para possibilitar uma forma de prover feedback sobre a destruição de um ator `Alien`[67].

Os comandos de jogo são gerados por atores e tratados pela camada de Lógica por meio de *event handlers* (`FireProjectile` e `MoveActor`). Da mesma forma que na Listagem 11, para se utilizar um comando de jogo basta despachar o evento ao qual ele está atrelado. Na Listagem 12, é apresentado um exemplo para o comando de jogo `FireProjectile`.

Listagem 12. Utilizando-se um comando de jogo

```
// A game command used to allow actors to fire.
class FireProjectile : public uge::BaseEventData
{
public:

    enum class Type : char
    {
        Bullet,
        Bomb
```

[67] É importante notar que isto não ocorre neste momento. O mundo do Meta-Jogo é o Mundo Abstrato de Jogo, ou seja, não possui nenhuma interação de nível físico definida – apenas relações livres de ES entre atores, componentes e eventos.

```
    };

    // A Global Unique Identifier (GUID) for the event.
    static const uge::EventType sk_EventType;

    // An event must have only raw data – no pointer or references
    // are allowed.
    explicit FireProjectile(uge::ActorID actorID,
                            FireProjectile::Type type)
        : m_ActorID(actorID), m_Type(type)
    {

    }

    // ...

    // The data must be read-only.
    uge::ActorID GetActorID() const
    {
        return m_ActorID;
    }

    FireProjectile::Type GetType() const
    {
        return m_Type;
    }

private:
    // The actor that fired the projectile.
    uge::ActorID m_ActorID;
    // The type of the projectile (Bullet or Bomb).
    FireProjectile::Type m_Type;
};

// Dispatch the created game command somewhere in the application.
void Foo()
```

```
{
     // ...
     std::shared_ptr<FireProjectile> pEvent(
                               LIB_NEW FireProjectile(actorID, type));
    uge::IEventManager::Get()->vQueueEvent(pEvent);
    // ...
}

// The Game Logic's game command handler.
// The command is sent as a pointer with its data (the data is raw).
void Running::FireProjectileHandler(
                                 uge::IEventDataSharedPointer pEventData)
{
    std::shared_ptr<FireProjectile> pData =
             std::static_pointer_cast<FireProjectile>(pEventData);

    // Create the desired projectile according to its type.
    std::string actorResourceFile("");
    if (pData->GetType() == FireProjectile::Type::Bullet)
    {
        actorResourceFile = "data/game/actors/bullet.xml";
    }
    else
    {
        actorResourceFile = "data/game/actors/bomb.xml";
    }

    uge::ActorSharedPointer pActorProjectile =
                          CreateActor(actorResourceFile);

    // Tailor the new actor to the profile's definition.
    TailorActorToProfile(pActorProjectile);

    // Set the required component data, add actor the game world...
    // ...
}
```

Em qualquer parte da implementação, é possível criar e enviar o comando de jogo desejado. Na Listagem 12, a classe `FireProjectile` encapsula o comando como um evento – como tudo no motor UGE, um evento são apenas dados puros (ou seja, não permitidos ponteiros ou referências). Em seguida, a função `Foo()` envia um o comando `FireProjectile` de alguma parte da aplicação.

O comando é tratado por um *handler* (no caso, `FireProjectileHandler`), que trata o evento recebido de acordo com a definição da regra de jogo estabelecida. Isto é interessante porque, efetivamente, esta construção eliminou todas as referências a possíveis entradas: se for possível gerar um comando de jogo, é possível fazer qualquer dispositivo de entrada registrado funcionar como um controle.

Dado não existir usuário até este momento, estes comandos são gerados apenas por atores com `AIScriptComponent` – estes atores também implementam suas regras particulares de IA (`AvoidProjectile` e `AttackActor`) para a simulação do jogo[68].

As demais regras de jogo são implementadas de forma similar: elas operam somente sobre os componentes necessários, enviam e tratam eventos conforme necessário. Ao final da implementação, o Mundo de Jogo Abstrato torna-se completo – o Meta-Jogo foi definido (Figura 26).

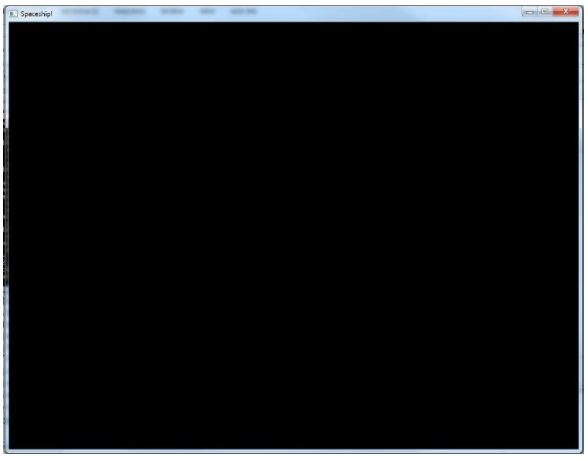


Figura 26. O Meta-Jogo: uma simulação livre de ES.

É importante notar que, na Figura 26, existe um jogo em execução. Toda a simulação de jogo está ativa – apenas não existe entrada e saída para mostra-las.

[68] Na aplicação de demonstração apresentada, isto é feito por um `GameController` ou por um método em `Running`. A situação descrita é a ideal e está em fatoração.

Para que a queda de uma árvore seja vista ou ouvida, é necessário transformar o Meta-Jogo em um Jogo – para isto, faz-se necessário o uso do perfil de usuário.

6.5 O Jogo Criado a Partir de Perfis de Jogador

Pela proposta apresentada na Seção 5.8, após criado o Meta-Jogo, é possível definir quantos jogos se desejar especializando as ES para o Mundo Abstrato de Jogo Obtido. Nesta seção, explora-se esta possibilidade como prova de conceito para criar uma demonstração para cinco perfis de usuários: usuário médio, usuário com deficiência visual (baixa visão), usuário com deficiência visual (cegueira), usuário com deficiência motora e usuário com deficiência cognitiva.

Isto permitirá mostrar o potencial de reuso da solução: para os jogos simples considerados, após criado o jogo base, a maioria das mudanças ocorrerá no perfil de usuário (devido à arquitetura *data-driven*) ou envolverá adições simples na implementação, como, por exemplo, tratamento para os eventos definidos na camada de Lógica. Um grande benefício é que, como todas as ES são definidas em tempo de execução, cada nova funcionalidade introduzida por meio de componentes ou eventos pode contribuir para melhorar o jogo para outros usuários. Ou seja, incluir novos usuários ao jogo cria uma experiência potencialmente melhor para todos.

6.5.1 Usuário Médio

A criação de um Mundo Concreto de Jogo a partir de um Mundo Abstrato de Jogos para usuários médios tende a ser intuitiva, dado que este é o público padrão para jogos digitais. Isto torna um perfil para usuários médios uma boa alternativa para o desenvolvimento inicial do jogo: pode-se criar o jogo ao mesmo em que se cria o Mundo Concreto de Jogo para este público.

Como o motor UGE fornece subsistemas padrão para apresentação de gráficos e sons para a saída e interação com mouse e teclado para a entrada, então a especialização torna-se facilitada e ocorre no nível do perfil de usuário.

Assim como no caso da camada de Lógica, o motor UGE fornece duas classes para a criação da camada de Visão: a `IGameView` e a `HumanGameView`. A `IGameView` é a interface para criação de camadas de Visão, ao passo que a `HumanGameView` oferece uma implementação padrão para a criação da apresentação do jogo ao usuário.

6.5.1.1 Especialização de Atores e Componentes

Um dos principais recursos da abordagem definida pelo motor UGE é a possibilidade de especializar os componentes definidos para cada ator do jogo. A especialização adiciona componentes de ES para criar as representações sensoriais dos atores definidos na Seção 6.4.1. Isto está ilustrado na Figura 27.

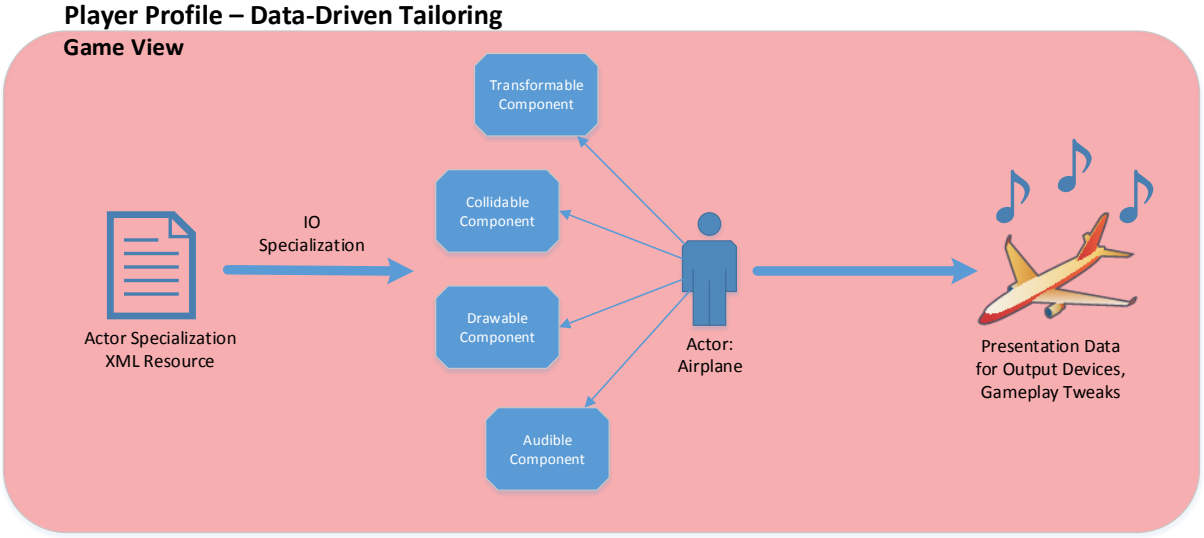


Figura 27. Especialização de atores por meio de componentes de saída.

Por exemplo, no caso de perfil de usuário médio, a especialização é feita com o arquivo XML da Listagem 13.

Listagem 13. Especialização para o perfil usuário médio da Listagem 9.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor                                                    type="Spaceship"
resource="data/config/player_profiles/average_user/entity/spaceship.xml">

  <TransformableComponent>
    <Position x="0.0f" y="0.0f" z="180.0f"/>
    <!-- YXZ order (yaw, pitch, roll) -->
```

```
    <Rotation yaw="3.1415f" pitch="0.0f" roll="0.0f"/>
    <Scale x="15.0f" y ="10.0f" z="30.0"/>
  </TransformableComponent>

  <HealthComponent>
    <InitialHealthPoints value="500"/>
    <MaximumHealthPoints value="500"/>
  </HealthComponent>

  <DrawableComponent>
    <NodeName n="Spaceship-Graphics" />
    <MeshFileName m="box.mesh" />
    <MaterialFileName n="" />
  </DrawableComponent>

  <AudibleComponent>
    <NodeName n="Ball-Audio" />
    <FileName n="data/audio/effects/blip16.wav" />
    <Volume v="1.0f" />
    <InitialProgress p="0.0f" />
    <Loop l="true" />
  </AudibleComponent>

</Actor>
```

Na Listagem 13, foram adicionados dois novos componentes: `AudibleComponent` para representação sonora e `DrawableComponent` para apresentação gráfica[69]. Ou seja, a partir deste momento, qualquer ator representado pelo arquétipo `Spaceship` possui saída gráfica e sonora.

Os valores dos componentes `TransformableComponent` e `HealthComponent` foram redefinidos – neste caso, estas alterações são realizadas apenas para este

[69] Esferas e cubos são artes de programador e servem como *place-holder* para a arte do jogo. É importante notar que os componentes de saída possuem campos para a alteração do modelo (`MeshFileName` em `DrawableComponent`) e do som (`FileName` em `AudibleComponent`) a ser utilizado. Caso eles sejam alterados com novos arquivos e o jogo for reexecutado, a nova arte corresponderá às representações contidas no arquivos escolhidos.

perfil. Ou seja, caso não sejam redefinidos nos demais perfis, estas mudanças serão realizadas apenas para o perfil do usuário médio. Aplicando-se o mesmo procedimento para todos os outros atores, pode-se obter um resultado semelhante ao da Figura 28.

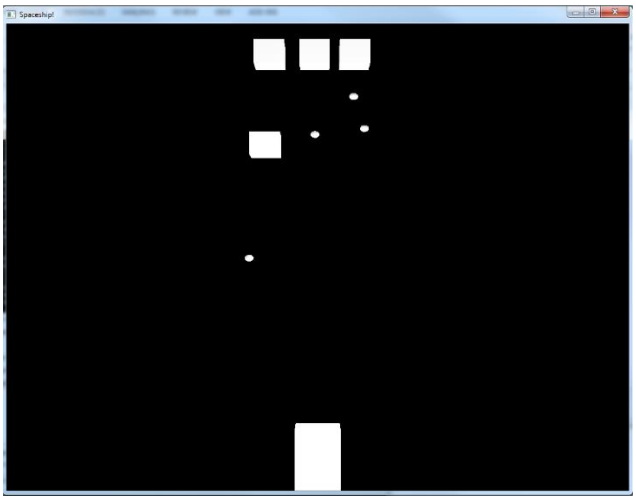


Figura 28. Saída do jogo após a especialização por componentes.

Na Figura 28, a `Spaceship` controlada pelo usuário é apresentada na parte de baixo como um paralelepípedo; os inimigos (`Aliens`), na parte superior e os projéteis (`Bombs` e `Bullets`) são as esferas.

6.5.1.2 Especialização de Comandos de Jogo

A entrada é definida por um `GameController` – seu objetivo é converter a entrada gerada pelo usuário em um comando de jogo. Isto ocorre como definido na Figura 23. O motor UGE suporta mapeamento de dispositivos de entrada e teclas pelo perfil, isto é, é possível customizar as entradas de acordo com a necessidade de interação. O formato do arquivo de mapeamento é apresentado na Listagem 14.

Listagem 14. Mapeamento de entrada.

```
<?xml version="1.0" encoding="UTF-8"?>
<Commands resource=
         "data/config/player_profiles/average_user/input_mapping.xml">
  <Actions>
    <State game_command_name="FireBullet" device="mouse"
           input_value="MB_LEFT"/>
    <State game_command_name="FireBomb" device="mouse"
           input_value="MB_RIGHT"/>
```

```
  </Actions>
  <States>
    <State game_command_name="PlayerMoveLeft" device="keyboard"
           input_value="KC_LEFT"/>
    <State game_command_name="PlayerMoveRight" device="keyboard"
           input_value="KC_RIGHT"/>
  </States>
  <Ranges>
  </Ranges>
</Commands>
```

Se os valores do Listagem 14 forem trocados, eles serão refletidos automaticamente no jogo contato que o dispositivo de entrada esteja registrado. Na Listagem 15, por exemplo, redefine-se o comando `FireBullet` para usar o teclado ao invés do mouse. A ação, que na Listagem 14 exigia o pressionamento do botão esquerdo do mouse, passou a usar a tecla de espaço de um teclado.

Listagem 15. Alterando-se o mapeamento de um comando de jogo.

```
    <!-- Original
    <State game_command_name="FireBullet" device="mouse"
           input_value="MB_LEFT"/>
    -->
    <!-- Novo -->
    <State game_command_name="FireBullet" device="keyboard"
           input_value="KB_SPACE"/>
```

Como os valores são convertidos para comandos de jogo pelo motor UGE, isto permite que, efetivamente, qualquer dispositivo seja usado para controlar o jogo – basta registrar o dispositivo (exige implementação) e realizar a conversão entre os valores do dispositivo para um comando de jogo (exige implementação simples). A partir deste ponto, todo o mapeamento pode ser feito sem implementação – diretamente no perfil.

A partir deste momento, tem-se todas as ES definidas para tornar o Mundo Abstrato de Jogo e um Mundo Concreto de Jogo – ou seja, o Meta-Jogo foi transformado em um Jogo. Do ponto de vista do usuário, a interação ocorre com um

jogo pronto, adequado às suas necessidades. Do ponto de vista do desenvolvedor, entretanto, é possível adequar o jogo para quantos outros públicos se desejar.

6.5.2 Usuário com Deficiência Visual (Baixa Visão)

Como apresentado na seção anterior, os perfis do motor UGE são definidos em texto. Isto significa que tudo pode ser alterado em um perfil. A partir desta seção, considerar-se-á diferentes perfis para atender a diferentes necessidades de interação. Para simplificar as explicações, serão considerados problemas de interação específicos para cada um dos perfis[70].

De acordo com a Figura 6, uma das possíveis estratégias consiste em aumentar os estímulos (*enhance stimuli*). O motor UGE torna este tipo de modificação trivial – ela pode ser realizada em nível de perfil, ou seja, nada é alterado no código.

Desta forma, supondo ser este o problema de interação, basta criar um novo perfil e alterar o valor da escala do `TransformableComponent` e alterar as texturas (e talvez o modelo[71]) dos atores no `DrawableComponent` para melhorar a interação (Listagem 16).

Listagem 16. Especialização de componentes para baixa visão.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor type="Alien" resource=
           "data/config/player_profiles/low_vision/entity/alien.xml">

  <TransformableComponent>
    <Position x="20.0f" y="0.0f" z="-180.0f"/>
    <!-- YXZ order (yaw, pitch, roll) -->
    <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
    <!-- Original do arquétipo:
```

[70] Deve-se observar que as alterações que serão feitas ilustram o potencial da abordagem empregada pelo motor. Elas permitem a alteração do jogo em tempo de execução de forma simples, totalmente configurável e permitem aos desenvolvedores tornarem um jogo acessível ao público com menor esforço. Mais detalhes no Capítulo 7.

[71] O *tech demo* usa um cubo para a demonstração. Caso fosse usado um modelo de arte, este perfil poderia alterá-lo para o cubo de forma a simplificar a saída e melhorar a acessibilidade do jogo para usuários com baixa visão.

```
        <Scale x="10.0f" y ="10.0f" z="10.0"/>
    -->
    <!-- Novo valor. -->
    <Scale x="25.0f" y ="25.0f" z="25.0"/>
  </TransformableComponent>

  <DrawableComponent>
    <NodeName n="Alien-Graphics" />
    <MeshFileName m="box.mesh" />
    <MaterialFileName n="" />
  </DrawableComponent>

  <AudibleComponent>
    <NodeName n="Alien-Audio" />
    <FileName n="data/audio/effects/kick.wav" />
    <Volume v="1.0f" />
    <InitialProgress p="0.0f" />
    <Loop l="true" />
  </AudibleComponent>

</Actor>
```

Adicionando-se o novo perfil aos perfis existentes e executando-se novamente o jogo, a interação foi adaptada (Figura 29): na parte superior da imagem, os atores inimigos tiveram seu tamanho ampliado. Isto poderia ser feito a todos os outros atores do jogo – bastaria alterar as especializações do perfil.

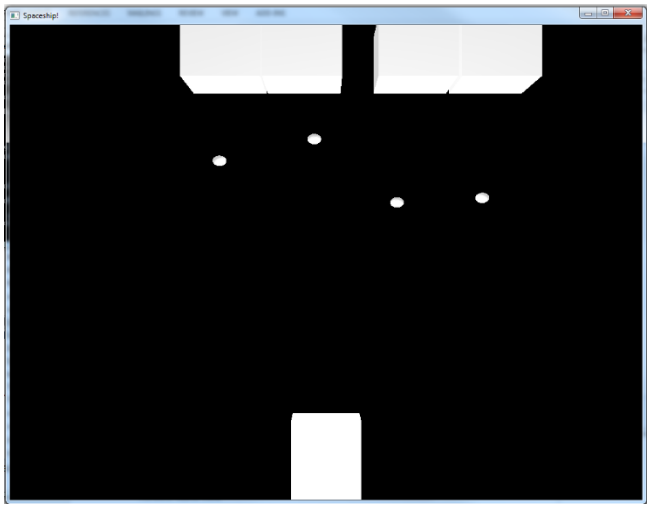


Figura 29. O jogo após a especialização da Listagem 16.

Em outras palavras, se for possível modificar apenas o perfil para melhorar a interação, UGE permite que o jogo seja adaptado sem alterações no código – basta apenas adicionar o perfil[72].

6.5.3 Usuário com Deficiência Motora

De acordo com a Figura 6, uma das formas de aumentar a acessibilidade em jogos para usuários com deficiência motora consiste na automação de entrada. Para o motor UGE, comandos de jogo são eventos, ou seja, podem ser despachados de qualquer parte de código. Isto permite que seja criado um controle na camada de Visão para enviar comandos de jogo pelo usuário, ou seja, permitir que ele tenha que usar menos botões ou similares para jogar.

Com o motor UGE, este tipo de construção também é bastante simples, como ilustrado na Listagem 17: basta alterar o método `vUpdate()` de um `GameController` e enviar os comandos desejados a cada frame de jogo.

Listagem 17. Simplificando a interação: enviando um comando de jogo pelo usuário

```
bool MotorImpairmentGameController::vUpdate(
                                    unsigned long timeElapsed)
{
    if (!uge::GameController::vUpdate(timeElapsed))
    {
```

[72] No momento, isto exige uma pequena modificação no código. Entretanto, ela pode ser facilmente refatorada com o uso de scripts e um padrão *Factory*.

```
        return false;
    }

    if (m_ActorID != uge::Actor::NULL_ACTOR_ID)
    {
        const unsigned int kProbabilityToMove = 30u;
        const unsigned int kProbabilityToStop = 20u;
        if (std::rand() % 100 <= kProbabilityToMove)
        {
            MoveActor::Direction direction = (std::rand() % 2) ?
                                        MoveActor::Direction::Left :
                                        MoveActor::Direction::Right;
            std::shared_ptr<sg::MoveActor> pEvent(
                      LIB_NEW sg::MoveActor(m_ActorID, direction));
            uge::IEventManager::Get()->vQueueEvent(pEvent);
        }
        else if (std::rand() % 100 <= kProbabilityToStop)
        {
            std::shared_ptr<sg::StopActor> pEvent(
                                LIB_NEW sg::StopActor(m_ActorID));
            uge::IEventManager::Get()->vQueueEvent(pEvent);
        }
    }

    return true;
}
```

No caso da Listagem 17, automatizou-se os comandos de jogo para movimentação para simplificar a interação[73] – o usuário, que antes precisava de quatro botões para jogar, agora precisa de apenas duas. Caso se desejasse, poder-se-ia seguir a mesma estratégia para reduzir a um único comando.

No momento, isto é feito adaptando-se o `GameController` e exige algumas mudanças no código. Entretanto, é possível explorar scripts e componentes de IA

[73] O algoritmo usado é muito simples, para fins de prova de conceito. Poder-se-ia sofisticá-lo o quanto se desejasse.

para tornar possível incorporar a mudança diretamente na especialização de entidades do perfil – ou seja, neste caso, bastará implementar o código para a IA desejada em script ao invés de se precisar alterar a implementação do jogo.

6.5.4 Usuário com Deficiência Cognitiva

A Figura 6 descreve que algumas formas exploradas para aumentar a acessibilidade de usuários com deficiência cognitiva a jogar são a redução de estímulo (explorada na 6.5.2), a redução da entrada (explorada na 6.5.3) e a redução do tempo de interação.

Uma das formas de se aumentar o tempo de interação consiste na redução da velocidade máxima que os atores podem atingir – uma mudança realizada nos valores de dados do componente (Listagem 18).

Listagem 18. Aumentando-se o tempo de interação.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor type="Alien" resource=
      "data/config/player_profiles/cognitive_impairment/entity/alien.xml">

  <TransformableComponent>
    <Position x="20.0f" y="0.0f" z="-180.0f"/>
    <!-- YXZ order (yaw, pitch, roll) -->
    <Rotation yaw="0.0f" pitch="0.0f" roll="0.0f"/>
    <Scale x="15.0f" y ="15.0f" z="15.0"/>
  </TransformableComponent>

  <OgreGraphicalComponent>
    <NodeName n="Alien-Graphics" />
    <MeshFileName m="box.mesh" />
    <MaterialFileName n="" />
  </OgreGraphicalComponent>

  <BulletPhysicsComponent>
    <LinearFactor x="1.0f" y="1.0f" z="1.0f"/>
    <AngularFactor x="0.0f" y="0.0f" z="0.0f"/>
```

```
    <!-- Original
         <MaxVelocity v="15.0f"/> -->
    <MaxVelocity v="3.0f"/>
    <MaxAngularVelocity v="0.0f"/>
  </BulletPhysicsComponent>

</Actor>
```

Ou seja, mais uma vez, a mudança pode ser realizada no nível de perfil. É interessante notar também que toda nova funcionalidade criada pode ser reusada – caso se desejasse, poder-se-ia partir dos perfis de baixa visão e deficiência motora para criar o perfil de deficiência cognitiva. Ou, caso este tivesse sido implementado primeiro, simplificar-se-ia a criação dos outros dois.

6.5.5 Usuário com Deficiência Visual (Cegueira)

Os perfis anteriores exploraram especializações em nível de perfil. De acordo com a Figura 6, uma das principais estratégias para aumentar a acessibilidade para usuários com deficiência visual é o uso de substituição sensorial. Infelizmente, o uso de apenas componentes não permite uma apresentação completa da interface de jogo.

Entretanto, propositalmente, um dos principais recursos do motor UGE ainda não foi explorado: a especialização de eventos (Figura 30).

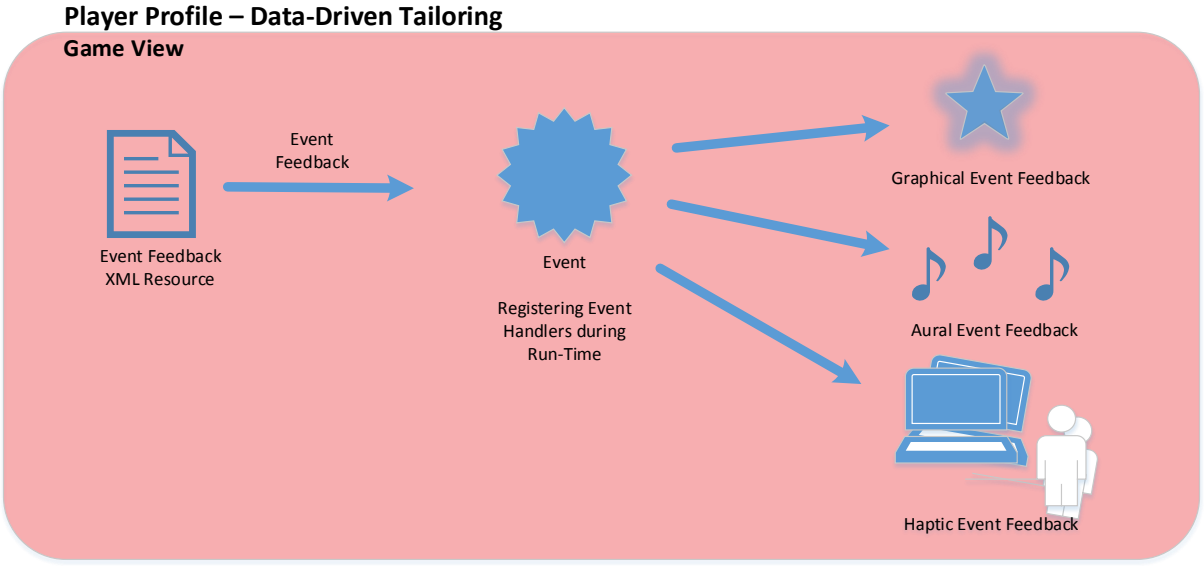


Figura 30. Especialização de eventos.

Se os recursos anteriores contribuíram com quase todas as recomendações da Figura 6, a especialização de eventos é responsável por completa-la. Eventos são excelentes para prover feedback imediato ao jogador.

Como não existem especializações de evento no momento, será necessário implementá-las. A implementação, entretanto, é facilitada dado que, pelo processo descrito na Seção 5.8, já existem diversos eventos disponíveis na implementação – basta trata-los. Uma primeira implementação pode ser tão simples quanto a da Listagem 19, que não explora som 3D.

Listagem 19. Exemplo de tratamento de evento.

```
void OnAlienDestroyed(uge::IEventDataSharedPointer pEventData)
{
    std::shared_ptr<sg::AlienDestroyed> pData =
            std::static_pointer_cast<sg::AlienDestroyed>(pEventData);

    PlaySoundEffect("data/audio/effects/aural-feedback/explosion.ogg",
                    0.1f, false, uge::Vector3(0.0f, 0.0f, 0.0f));
}

// Outros tratamentos...
```

Para utilizar o novo tratamento, basta adicioná-lo e ativá-lo no perfil de usuário (Listagem 20).

Listagem 20. Habilitando o tratamento de eventos para o peril de usuário cego.

```
<?xml version="1.0" encoding="UTF-8"?>

<EventSpecialization
resource="data/config/player_profiles/blindness/events/aural_events.xml">

  <Events>
    <Event name="OnAlienDestroyed" enabled="true"/>
    <Event name="OnFireProjectile" enabled="true"/>
    <Event name="OnMoveActor" enabled="true">
      <FileName n="data/audio/effects/spaceship.wav" />
      <Volume v="1.0f" />
      <InitialProgress p="0.0f" />
      <Loop l="true" />
    </Event>

    <Event name="OnStopActor" enabled="true"/>
  </Events>

</EventSpecialization>
```

Da mesma forma que nos casos anteriores, especializa-se, em seguida, os atores e componentes desejados para compor o jogo. Desta vez, além dos eventos de áudio, os atores que deverão ser localizados por meio do som terão componentes aurais – por exemplo, os projéteis.

Listagem 21. Um exemplo de especialização para o projétil `Bullet`.

```
<?xml version="1.0" encoding="UTF-8"?>

<Actor type="Bullet"
       resource="data/config/player_profiles/blindness/entity/bullet.xml">

  <!--<DrawableComponent>
    <NodeName n="Bullet-Graphics" />
    <MeshFileName m="sphere.mesh" />
    <MaterialFileName n="" />
```

```
  </DrawableComponent>-->

  <AudibleComponent>
    <NodeName n="Bullet-Audio" />
    <FileName n="data/audio/effects/laser.wav" />
    <Volume v="1.0f" />
    <InitialProgress p="0.0f" />
    <Loop l="true" />
  </AudibleComponent>

</Actor>
```

Alterando-se a posição da câmera para primeira pessoa (em uma nova `HumanGameView`), o resultado da combinação seria similar ao da Figura 31: um *audio-only* game[74]. Os sons são explorados em 3D por conversões realizadas pelo motor – a apresentação dos sons supõe a `Spaceship` do usuário como a posição do ouvinte.

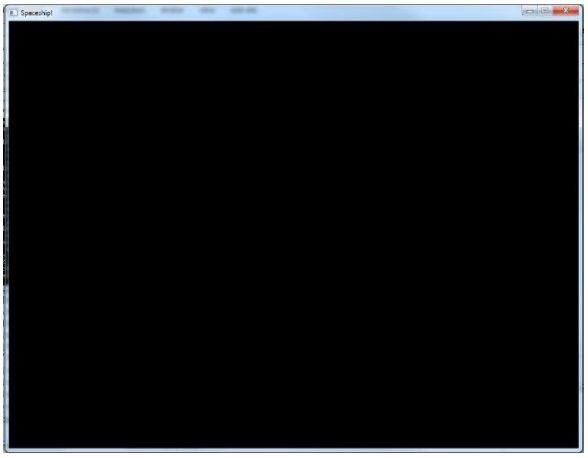


Figura 31. Uma versão sonora do jogo.

Após a primeira implementação, é possível realizar testes com usuários para melhorar a qualidade. Na Listagem 21, foi deixado um componente gráfico comentado. Caso o comentário fosse removido para todas as especializações de atores e o jogo fosse reiniciado, obter-se-ia o Jogo ilustrado na Figura 32: todos os atores teriam representação gráfica.

[74] Embora a imagem seja similar à apresentada na Figura 26, a Figura 31 apresenta um Jogo. Infelizmente, a versão em papel não permite a reprodução de sons – mas a demonstração sugerida no início de capítulo permite. Isto ficará evidente a seguir, na Figura 32, quando se remover os comentários dos componentes gráficos.

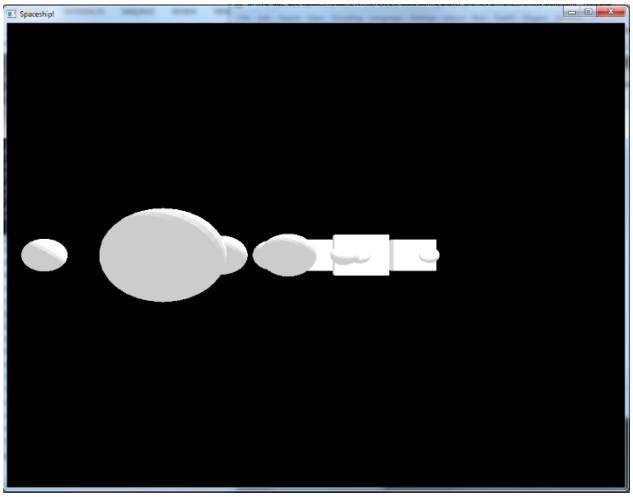


Figura 32. Habilitando-se componentes gráficos no Jogo da Figura 31.

Ou seja, após definidos novos componentes e eventos, sempre é possível criar novos Mundos Concretos de Jogo com mudanças apenas nos perfis.

6.6 *Code Once, Enable Everyone*

Antes da Seção 6.5.5, havia suporte apenas primitivo para som no jogo. Entretanto, isto foi alterado com a implementação do perfil para usuários cegos. Assim como componentes, UGE explora a arquitetura *data-driven* para eventos. Isto significa que, agora, é possível habilitar som para todos os perfis existentes – com uma simples modificação do perfil de usuário (Listagem 22).

Listagem 22. Habilitando o tratamento de eventos para os perfis.

```
<?xml version="1.0" encoding="UTF-8"?>

<EventSpecialization resource=
      "data/config/player_profiles/low_vision/events/aural_events.xml">

  <Events>

    <Event name="OnAlienDestroyed" enabled="true"/>
    <Event name="OnFireProjectile" enabled="true"/>
    <Event name="OnMoveActor" enabled="true">
      <FileName n="data/audio/effects/step.wav" />
```

```
      <Volume v="1.0f" />
      <InitialProgress p="0.0f" />
      <Loop l="false" />
    </Event>
    <Event name="OnStopActor" enabled="true"/>

  </Events>

</EventSpecialization>
```

Ou seja, basta executar a versão do jogo para usuários de baixa visão que, a partir deste momento, o jogo também terá sons. Como é possível habilitar e desabilitar sons pelo perfil, também é possível escolher apenas os desejados.

Da mesma forma, pode-se considerar que o próximo perfil implementado incorpore as habilidades de interação para a deficiência auditiva e adicione os seguintes tratamentos para `OnAlienDestroyed`:

- Onomatopeias;
- Efeitos de partículas;
- Feedback hético;
- Transcrições textuais;
- Legendas.

Os tratamentos para estes eventos poderiam ser habilitados para todos os outros perfis que poderiam ser beneficiados por eles. Novamente, poder-se-ia escolher quais são os desejados, um a um. Com o motor UGE, cada novo evento, *handler* ou componente adicionado melhora o jogo para alguém – afinal, como as ES do jogo são definidas de forma totalmente *data-driven*, sempre é possível escolher o que se deseja ativar no perfil.

6.7 Considerações Finais

Neste capítulo, apresentou-se como a abordagem do motor UGE contribui para melhorias de acessibilidade em jogos. Embora tenha sido apresentado um

estudo de viabilidade muito simples, é possível ver como as abordagens definidas ao longo desta dissertação colaboram para a produção de jogos universais: elas incluem, em apenas três primitivas, as principais recomendações de acessibilidade presentes na Literatura.

Ilustrou-se também o potencial de reuso da solução e que, potencialmente, a solução apresentada é escalável – ao menos para jogos simples, como proposto. Ainda é necessária a implementação de um jogo complexo para a validação do projeto, um possível trabalho futuro.

Capítulo 7

AVALIAÇÃO

"Hi Franco!

Your project is amazing. I'm really proud that you decided to use the practical game accessibility guidelines of Includication. Many try to adhere to these guidelines already, therefore, an engine that could meet the entire level I of accessibility for all disabilities with very little coding effort would be a project the entire game accessibility movement would be behind.

In particular, AbleGamers would love to help any way we can. We have some in-house accessibility experts who are some of the top developers in the industry. Shoot me some details and I will get you in contact with those developers.

Thanks again for your great work!"

Steve Spohn (autor de Includification, fundador e presidente da fundação AbleGamers[75])

7.1 Considerações Iniciais

O Capítulo 5 e o Capítulo 6 apresentaram o motor UGE. Embora a demonstração apresentada no Capítulo 6 esteja incompleta, o motor obteve alguns resultados positivos até este momento. Esta seção relata alguns deles.

[75] AbleGamers (http://www.ablegamers.org/) é a maior instituição sem-fins lucrativos do mundo dedicada à melhorar a acessibilidade em jogos digitais. Diversos dos projetos apoiados pela instituição buscam promover o aumento da qualidade de vida de pessoas com diferentes necessidades de interação por meio de jogos digitais.

7.2 Prova de Conceito

O primeiro jogo implementado utilizando o motor UGE foi um clone do jogo Pong (alguns detalhes de implementação estão descritos em (GARCIA; NERIS, 2013a)). O jogo Pong original, publicado pela Atari Inc. em 1972, baseia-se na simulação de um jogo simplificado de *ping-pong* entre dois jogadores. Os jogadores controlam raquetes para, assim como em um jogo de *ping-pong*, rebater uma bola.

Cada raquete pode ser movimentada nos dois sentidos possíveis de uma mesma direção. Caso a bola colida com a raquete, a bola é rebatida para o lado oposto e o jogo prossegue. Caso contrário, o jogador adversário marca um ponto.

Embora simples, o jogo apresenta a maioria dos elementos presentes em todos os jogos, como um laço para a repetição do jogo (*game loop*), manipulação de entrada do usuário, física (colisões e movimentação) e apresentação da interface.

O protótipo de jogo criado para Pong é tão personalizável quanto o protótipo definido no Capítulo 6. Entretanto, as alterações nos perfis necessitam de recompilações no código, razão pela qual ele não foi adotado para fornecer a demonstração do motor.

Por ser um jogo mais simples que Space Invaders, para o jogo Pong foram criados apenas dois perfis: usuário médio e usuário com deficiência visual (cego). A versão usuário médio utiliza gráficos e sons; a versão para deficiência visual apresenta o jogo com uma interface totalmente aural (Figura 33).

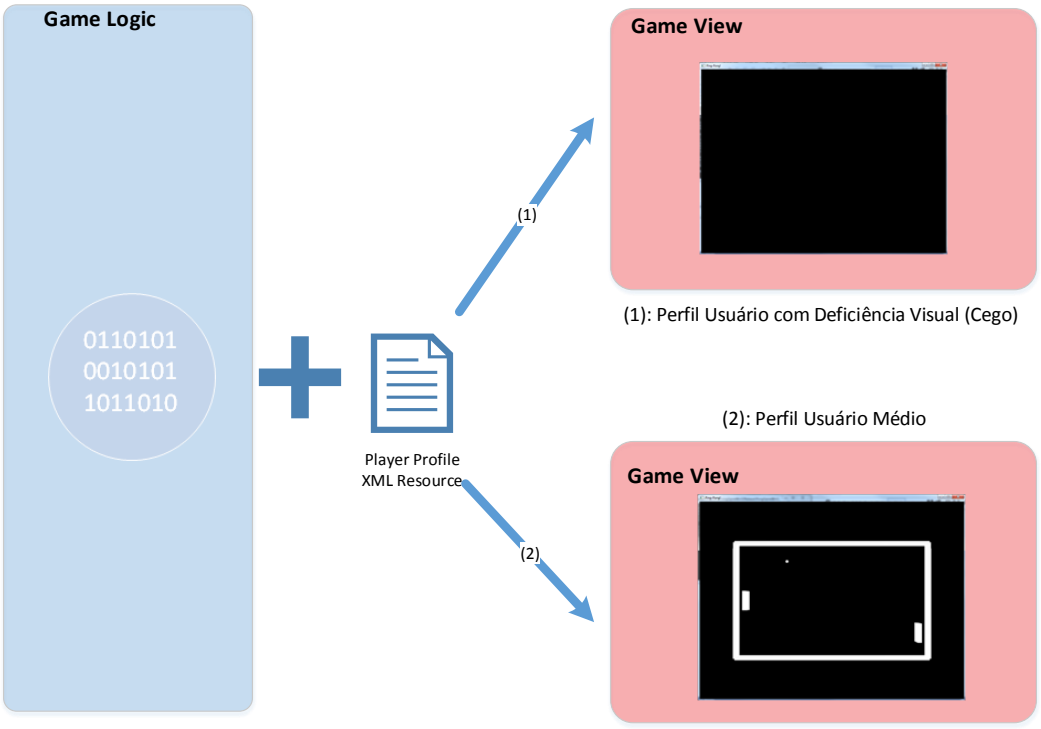


Figura 33. Os perfis criados para Pong.

No Quadro 4, enumera-se os componentes utilizados por cada uma das entidades para a criação do Meta-Jogo. Todos estes componentes são de entrada e saída.

Quadro 4. Relação de atores e componentes para o Mundo Abstrato de Jogo.

Entidade	Componentes

Raquete	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

Bola	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

Campo	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

É interessante notar que, para o caso de Pong, todos os componentes são iguais – o que diferencia o comportamento de um e de outro são seus valores. Por exemplo, a Raquete pode ser movimentada, ao passo que o Campo não pode (possui massa infinita).

Os eventos definidos para o jogo Pong são: `RestartGame`, `PlayerScored`, `BallWallCollision`, `BallPaddleCollision`, `PaddleWallCollision`, `MovePaddle`, `StopPaddle`. Os dois últimos (`MovePaddle` e `StopPaddle`) são comandos de jogo. Eles refletem as regras de um jogo Pong padrão – colisão, placar, movimentação.

No Quadro 5, enumera-se os componentes utilizados por cada uma das entidades para a criação do Meta-Jogo. Todos estes componentes são isentos de ES e permitem a simulação do Mundo Abstrato de Jogo.

Quadro 5. Relação de atores e componentes para o Mundo Abstrato de Jogo.

Entidade	**Componentes**

Raquete	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

Bola	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

Campo	`TransformableComponent`
	`CollidableComponent`
	`KinematicComponent`

No Quadro 6, apresenta-se a especialização de atores e eventos para o perfil de usuário médio definidos no respectivo perfil de usuário. Para esta versão do jogo, foram usados apenas componentes para definir-se às saídas do jogo. A entrada é realizada por teclado, usando as teclas ‘ ↑ ’ e ‘ ↓ ’ para mover a raquete.

Quadro 6. Especialização para o perfil usuário médio do jogo Pong.

Entidade	**Componentes**	**Eventos**
Raquete	`DrawableComponent`	-
Bola	`DrawableComponent` `AudibleComponent`	
Campo	`DrawableComponent`	

Desta forma, quando a aplicação de jogo é utilizada com o perfil escolhido segundo as definições do Quadro 6, o usuário interage com o Jogo da Figura 34.

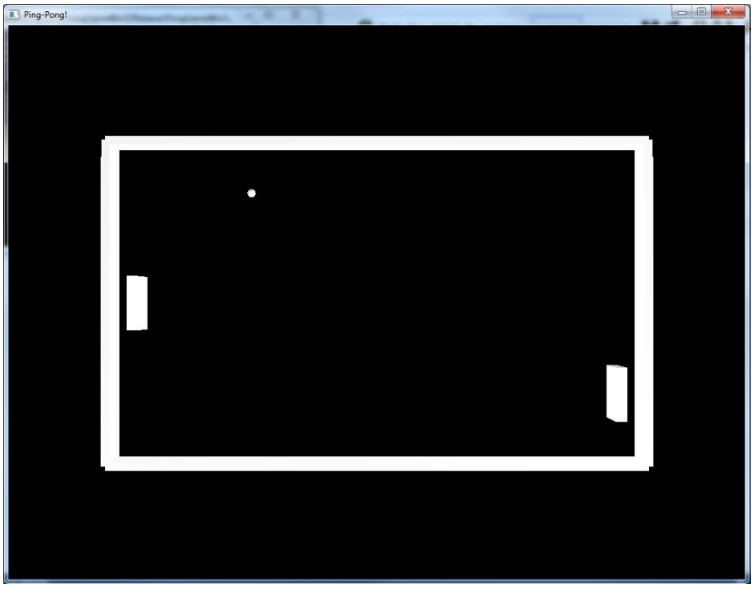


Figura 34. O Jogo de ping-pong adaptado segundo as definições do perfil usuário médio.

No Quadro 7, por sua vez, apresenta-se a especialização de atores e eventos para o perfil de usuário com deficiência visual definidos no respectivo perfil de usuário. Para esta versão do jogo, apenas a bola utiliza um componente para especialização – como o áudio é repetido, utiliza-se componentes nos demais usuários tende a confundir o usuário. A entrada é realizada por teclado, usando as teclas '←' e '→' para mover a raquete.

Quadro 7. Especialização para o perfil usuário com deficiência visual do jogo Pong.

Entidade	**Componentes**	**Eventos**
Raquete	-	`OnRestartGame`
Bola	`AudibleComponent`	`OnPlayerScored`
Campo	-	`OnBallWallCollision`
		`OnBallPaddleCollision`
		`OnPaddleWallCollision`
		`OnMovePaddle`
		`OnStopPaddle`

No caso do Quadro 7, pode-se observar que a maioria do feedback provêm de eventos. Para cada um destes eventos, foi adicionado um *handler* para reproduzir um som após sua ocorrência. Para que estes sons fossem apresentados segundo a posição da raquete do jogador, a câmera foi alterada para primeira pessoa.

Desta vez, ao interagir com o Jogo criado a partir das definições do perfil, o usuário interage com o Jogo resultante ilustrado na Figura 35.

Figura 35. O Jogo obtido do uso do perfil de usuário com deficiência visual.

É interessante notar que, caso fossem adicionados `DrawableComponents` a todos os atores do Quadro 7, o usuário estaria interagindo com o Jogo mostrado na Figura 36.

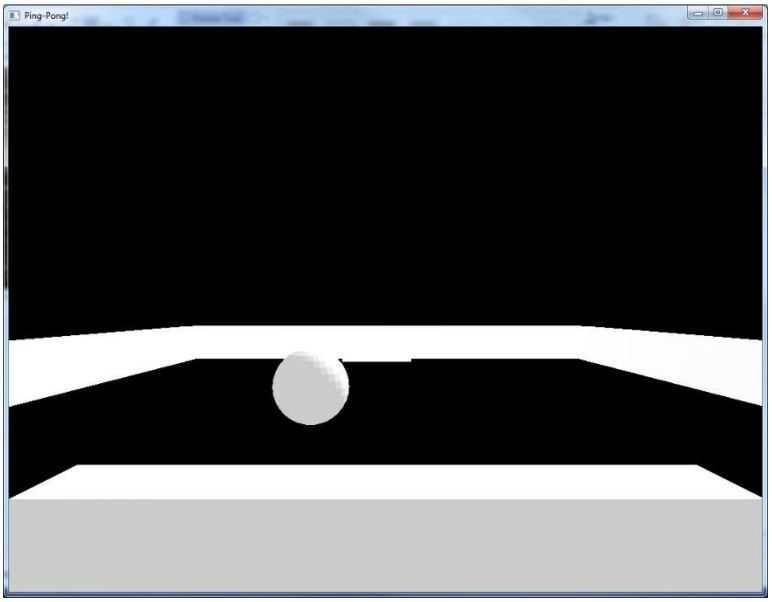


Figura 36. O Jogo obtido da adição de `DrawableComponents` ao perfil de usuário com deficiência visual.

Também é interessante notar que, caso se desejasse, os *handlers* poderiam ser reusados na versão do usuário médio para adicionar som ao jogo – bastaria se adicionar os valores no perfil de usuário, da mesma forma que foi feito no Capítulo 6.

7.3 Recomendações de Acessibilidade

Na Seção 4.4, indicou-se que as abordagens utilizadas pelo motor UGE cobrem as principais estratégias de acessibilidade indicadas pela survey de Yuan *et. al* (2011). Esta seção analisa algumas outras compilações de recomendações para verificar a adequação do motor UGE.

7.3.1 Includification

Includification fornece um conjunto de diretivas para aumentar a acessibilidade em jogos digitais. Na Tabela 2, apresenta-se uma relação das recomendações que são apoiadas pelo motor UGE para tornar o desenvolvimento de um jogo mais simples.

7.3.2 Game Accessibility Guidelines

Assim como *Includification*, *Game Accessibility Guidelines* fornece diretivas para tornar jogos mais acessíveis. Na Tabela 3, indica-se em quais delas o motor UGE pode contribuir para tornar o desenvolvimento do jogo mais simples.

Tabela 2. Suporte às recomendações de *Includification* pelo motor UGE.

Categoria	**Nível**	**Recomendação**	**Suporte**
Cognitive	*3 – Best*	*Enemy Marking*	Possível no futuro
Cognitive	*2 – Better*	*Training levels*	Não se aplica (jogo)
Cognitive	*2 – Better*	*Intuitive menus*	Não se aplica (jogo)
Cognitive	*1 – Good*	*Tutorial*	Não se aplica (jogo)
Cognitive	*1 – Good*	*Sandbox mode*	Não se aplica (jogo)
Cognitive	*1 – Good*	*Difficult levels*	Perfil
Cognitive	*-*	*Perspective*	Perfil
Cognitive	*-*	*Auto-Pass*	Não se aplica (jogo)
Cognitive	*-*	*Reward System Balance*	Não se aplica (jogo)
Hearing	*3 – Best*	*Options to include ambient noise as text output*	Perfil – eventos
Hearing	*3 – Best*	*Alternative reactionary input*	Não se aplica (jogo)
Hearing	*-*	*Subtitles are present*	Não se aplica (jogo)
Hearing	*-*	*Ambient noise is included*	Perfil – eventos
Hearing	*-*	*Identifies speaker*	Possível no futuro
Hearing	*-*	*All audio cues are accompanied by visual cues*	Perfil – eventos
Hearing	*-*	*Game can be successfully completed without sound*	Perfil
Mobility / Visual /	*3 – Best*	*Speed Settings*	Perfil – componentes

Cognitive			
Mobility	*3 – Best*	*Input Devices*	Perfil
Mobility	*2 – Better*	*On-screen keyboard functions properly*	Possível no futuro
Mobility	*2 – Better*	*No button mashing*	Não se aplica (jogo)
Mobility	*2 – Better*	*Can play with only the mouse*	Perfil
Mobility	*2 – Better*	*Can play with only the keyboard*	Perfil
Mobility	*2 – Better*	*Can move user interface elements*	Possível no futuro
Mobility	*2 – Better*	*No mandatory quick time events*	Não se aplica (jogo)
Mobility	*2 – Better*	*Timing of movement / button pressing not important*	Não se aplica (jogo)
Mobility	*2 – Better*	*Difficulty levels*	Perfil
Mobility	*2 – Better*	*Game assists*	Perfil – Game commands, componentes
Mobility	*2 – Better*	*Third party access*	Não se aplica (jogo)
Mobility	*2 – Better*	*Macroability*	Possível no futuro
Mobility	*2 – Better*	*Save points*	Não se aplica (jogo)
Mobility	*2 – Better*	*Sensitivity slides*	Perfil
Mobility	*1 – Good*	*Remappable keys*	Perfil
Mobility	*1 – Good*	*Camera / mouse sensibility*	Perfil
Mobility	*1 – Good*	*Alternate controls*	Perfil
Visual	*3 – Best*	*Text-to-Speech Input*	Possível no futuro
Visual	*2 – Better*	*Map recoloring options / alternative views*	Perfil – componentes,

			eventos
Visual / Hearing	*2 – Better*	*Customizable Fonts*	Possível no futuro
Visual	*1 – Good*	*Colorblind options are present or not needed*	Perfil – componentes
Visual	*1 – Good*	*Font size can be changed*	Possível no futuro
Visual	*1 – Good*	*High-contrast reticle*	Perfil - componentes
Visual	*1 – Good*	*Enemy-marking*	Perfil - componentes
Visual / Hearing	*1 – Good*	*Font color can be changed*	Possível no futuro
Visual	*-*	*No key elements of the game are identified by red and green*	Perfil – componentes
Visual	*-*	*Game presented in high contrast*	Perfil – componentes
Visual	*-*	*Subtitles are easy to read*	Possível no futuro
Visual	*-*	*Subtitles are letterboxed*	Possível no futuro
Visual	*-*	*Game menus are easy to see / read / use*	Não se aplica (jogo)

Tabela 3. Suporte às recomendações de *Game Accessibility Guidelines* pelo motor UGE.

Categoria	**Nível**	**Recomendação**	**Suporte**
General	*Basic*	*Provide details of accessibility features on packaging and/or website*	Não se aplica (jogo)
General	*Basic*	*Offer a wide choice of difficulty levels*	Perfil
General	*Basic*	*Ensure that all settings are saved/remembered*	Perfil
General	*Intermediate*	*Allow difficulty level to be altered during gameplay, either through settings or adaptive difficulty*	Possível no futuro

General	*Intermediate*	*Include some people with impairments amongst play-testing participants*	Não se aplica (jogo)
General	*Intermediate*	*Offer a means to bypass gameplay elements that aren't part of the core mechanic, via settings or in-game skip option*	Perfil
General	*Intermediate*	*Include assist modes such as auto-aim and assisted steering*	Perfil
General	*Intermediate*	*Provide a manual save feature*	Não se aplica (jogo)
General	*Intermediate*	*Provide an autosave feature*	Não se aplica (jogo)
General	*Intermediate*	*Allow a preference to be set for playing online multiplayer with/without others who are using accessibility features that could give a competitive advantage*	Não se aplica (jogo)
General	*Advanced*	*Include every relevant category of impairment (motor, cognitive, etc) amongst play-testing participants, in representative numbers based on age / demographic of target audience*	Não se aplica (jogo)
General	*Advanced*	*Allow gameplay to be fine-tuned by exposing as many variables as possible*	Perfil
General	*Advanced*	*Allow settings to be saved to different profiles, at either game or platform level*	Perfil
Motor	*Basic*	*Allow controls to be remapped / reconfigured*	Perfil
Motor	*Basic*	*Ensure that all areas of the user interface can be accessed using the same input method as the gameplay*	Perfil – game command
Motor	*Basic*	*Include an option to adjust the sensitivity of controls*	Perfil
Motor	*Basic*	*Ensure controls are as simple as possible, or provide a simpler alternative*	Perfil
Motor	*Intermediate*	*Ensure interactive elements / virtual controls are large and well spaced, particularly on small or touch screens*	Perfil

Motor	*Intermediate*	*Support more than one input device*	Perfil
Motor	*Intermediate*	*Make interactive elements that require accuracy (eg. cursor/touch controlled menu options) stationary*	
Motor	*Intermediate*	*Ensure that multiple simultaneous actions (eg. click/drag or swipe) are not required, and included only as a supplementary / alternative input method*	Perfil
Motor	*Intermediate*	*Ensure that all key actions can be carried out by digital controls (pad / keys / presses), with more complex input (eg. analogue, speech, gesture) not required, and included only as supplementary / alternative input methods*	Perfil
Motor	*Intermediate*	*Include an option to adjust the game speed*	Perfil
Motor	*Intermediate*	*Avoid repeated inputs (button-mashing/quick time events)*	Não se aplica (jogo)
Motor	*Intermediate*	*If producing a PC game, support windowed mode for compatibility with overlaid virtual keyboards*	Possível no futuro
Motor	*Advanced*	*Do not make precise timing essential to gameplay – offer alternatives, actions that can be carried out while paused, or a skip mechanism*	Não se aplica (jogo)
Motor	*Advanced*	*Include a cool-down period (post acceptance delay) of 0.5 seconds between inputs*	Possível no futuro
Motor	*Advanced*	*Provide very simple control schemes that are compatible with assistive technology devices, such as switch or eye tracking*	Possível no futuro
Cognitive	*Basic*	*Allow the game to be started without the need to navigate through multiple levels of menus*	Não se aplica (jogo)
Cognitive	*Basic*	*Use an easily readable default font size*	Possível no futuro

Cognitive	*Basic*	*Use simple clear language*	Não se aplica (jogo)
Cognitive	*Basic*	*Use simple clear text formatting*	Não se aplica (jogo)
Cognitive	*Basic*	*Include tutorials*	Não se aplica (jogo)
Cognitive	*Basic*	*Allow players to progress through text prompts at their own pace*	Não se aplica (jogo)
Cognitive	*Basic*	*Avoid flickering images and repetitive patterns*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Include contextual in-game help/guidance/tips*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Indicate / allow reminder of current objectives during gameplay*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Include a means of practicing without failure, such as a practice level or sandbox mode*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Employ a simple, clear narrative structure*	Não se aplica (jogo)
Cognitive	*Intermediate*	*If using a long overarching narrative, provide summaries of progress*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Ensure no essential information (especially instructions) is conveyed by text alone, reinforce with visuals and/or speech*	Perfil
Cognitive	*Intermediate*	*Give a clear indication that interactive elements are interactive*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Provide an option to turn off / hide background movement*	Perfil
Cognitive	*Intermediate*	*Support voice chat as well as text for multiplayer games*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Provide gameplay thumbnails with game saves*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Provide separate volume controls or mutes for effects, speech and background/music*	Perfil
Cognitive	*Intermediate*	*Ensure sound / music choices for each key objects / events are distinct from each other*	Não se aplica (jogo)

Cognitive	*Intermediate*	*Include an option to adjust the game speed*	Não se aplica (jogo)
Cognitive	*Intermediate*	*Provide a choice of text colour, low/high contrast choice as a minimum*	Possível no futuro
Cognitive	*Advanced*	*Provide pre-recorded voiceovers for all text, including menus and installers*	Não se aplica (jogo)
Cognitive	*Advanced*	*Avoid any sudden unexpected movement or events*	Não se aplica (jogo)
Cognitive	*Advanced*	*Allow all narrative and instructions to be replayed*	Não se aplica (jogo)
Cognitive	*Advanced*	*Use symbol-based chat (smileys etc)*	Perfil
Cognitive	*Advanced*	*Provide an option to turn off / hide all non interactive elements*	Perfil
Vision	*Basic*	*Ensure no essential information is conveyed by a colour alone*	Perfil
Vision	*Basic*	*If the game uses field of view (3D engine only), set an appropriate default for expected viewing environment*	Código (pode ir para perfil)
Vision	*Basic*	*Use an easily readable default font size*	Possível no futuro
Vision	*Basic*	*Use simple clear text formatting*	Possível no futuro
Vision	*Basic*	*Provide high contrast between text and background*	Perfil
Vision	*Intermediate*	*If the game uses field of view (3D engine only), allow a means for it to be adjusted*	Código (pode ir para perfil)
Vision	*Intermediate*	*Avoid (or provide option to disable) any difference between controller movement and camera movement, such as weapon/walk bobbing or mouse smoothing*	Não se aplica (jogo)
Vision	*Intermediate*	*Use surround sound*	Perfil
Vision	*Intermediate*	*Provide an option to turn off / hide background animation*	Possível no futuro
Vision	*Intermediate*	*Ensure screenreader support for mobile devices*	Sem suporte a dispositivos

			móveis no momento
Vision	*Intermediate*	*Provide an option to adjust contrast*	Perfil
Vision	*Intermediate*	*Ensure sound / music choices for key objects / events are distinct from each other*	Perfil
Vision	*Intermediate*	*Provide a choice of cursor / crosshair colours / designs*	Possível no futuro
Vision	*Intermediate*	*Give a clear indication that interactive elements are interactive*	Perfil
Vision	*Intermediate*	*Ensure manual / website are provided in a screenreader friendly format*	Não se aplica (jogo)
Vision	*Intermediate*	*Provide separate volume controls or mutes for effects, speech and background/music*	Perfil
Vision	*Intermediate*	*Avoid placing essential temporary information outside the player's eye-line*	Não se aplica (jogo)
Vision	*Advanced*	*Allow the font size to be adjusted*	Possível no futuro
Vision	*Advanced*	*Provide a pingable sonar-style audio map*	Possível no futuro
Vision	*Advanced*	*Provide pre-recorded voiceovers for all text, including menus and installers*	Não se aplica (jogo)
Vision	*Advanced*	*Provide a voiced GPS*	Possível no futuro
Vision	*Advanced*	*Allow easy orientation to / movement along compass points*	Possível no futuro
Vision	*Advanced*	*Ensure that all key actions can be carried out by digital controls (pads / keys / presses), with more complex input (eg. analogue, gesture) not required, and included only as supplementary / alternative input methods*	Perfil
Vision	*Advanced*	*Ensure screenreader support, including menus & installers*	Possível no futuro
Vision	*Advanced*	*Use distinct sound / music design for all objects and events*	Perfil
Vision	*Advanced*	*Simulate binaural recording*	Possível no futuro

Hearing	*Basic*	*Provide separate volume controls or mutes for effects, speech and background / music*	Perfil
Hearing	*Basic*	*Ensure no essential information is conveyed by audio alone, reinforce with text / visuals*	Perfil
Hearing	*Basic*	*If any subtitles / captions are provided, use an easily readable default font size, simple clear text formatting and provide high contrast between text and background*	Possível no futuro
Hearing	*Intermediate*	*Keep background noise to minimum during speech*	Não se aplica (jogo)
Hearing	*Intermediate*	*Provide a text alternative for all speech (subtitles / captions)*	Perfil
Hearing	*Intermediate*	*Allow text alternatives to be displayed before any sound is played*	Possível no futuro
Hearing	*Intermediate*	*Provide a text description of narratively / atmospherically significant background noises*	Possível no futuro
Hearing	*Intermediate*	*Provide a visual indication of who is currently speaking*	Perfil
Hearing	*Intermediate*	*Support text chat as well as voice for multiplayer*	Não se aplica (jogo)
Hearing	*Intermediate*	*Support visual means of communicating in multiplayer*	Não se aplica (jogo)
Hearing	*Intermediate*	*Allow a preference to be set for playing online multiplayer with players who will only play with / are willing to play without voice chat*	Não se aplica (jogo)
Hearing	*Intermediate*	*Ensure that all important supplementary information (eg. the direction you are being shot from) conveyed by audio is replicated in text / visuals*	Perfil
Hearing	*Intermediate*	*Provide a stereo/mono toggle*	Possível no futuro
Hearing	*Advanced*	*Ensure that subtitles/captions are cut down to and presented at an*	Possível no futuro

		appropriate words-per-minute for the target age-group	
Hearing	*Advanced*	*Provide signing*	Possível no futuro
Speech	*Basic*	*Ensure that speech input is not required, and included only as a supplementary / alternative input method*	Perfil
Speech	*Intermediate*	*Allow a preference to be set for playing online multiplayer with players who will only play with / are willing to play without voice chat*	Não se aplica (jogo)
Speech	*Intermediate*	*Support text chat as well as voice for online multiplayer*	Não se aplica (jogo)
Speech	*Intermediate*	*Support visual means of communicating in multiplayer*	Não se aplica (jogo)
Speech		*Base speech recognition on individual words from a small vocabulary (eg. 'yes' 'no' 'open') instead of long phrases or multi-syllable words*	Não se aplica (jogo)
Speech	*Advanced*	*Base speech recognition on hitting a volume threshold (eg. 50%) instead of words*	Não se aplica (jogo)

7.4 Avaliação Externa

Até o momento da escrita desta dissertação, três processos de avaliação externa haviam sido conduzidos, com diferentes níveis de sucesso. Para todos os convites, buscou-se a opinião de desenvolvedores e designers de jogos profissionais.

Todos os processos foram realizados à distância, pela Internet, buscando-se voluntários. Em virtude destas restrições, infelizmente o processo iniciou-se tardiamente – para que fosse possível o uso do motor, a documentação deveria estar completa o suficiente para a realização da tarefa.

7.4.1 Primeira Avaliação

Para a primeira avaliação, foi solicitada a implementação de um protótipo simples, nos moldes do apresentado no Capítulo 6. O convite foi realizado apenas no Brasil para membros da área acadêmica[76], com o prazo definido para realização da tarefa de 13 de março à 1º de abril.

Neste primeiro momento, dez pessoas manifestaram interesse no projeto. Destes participantes, um solicitou mais tempo para a realização da tarefa. Outros dois ofereceram feedback sobre o projeto por meio de e-mails. Um informou dificuldades sobre o uso das linguagens envolvidas. Outros dois relataram ter interesse no futuro para parcerias.

Os comentários sobre a primeira avaliação estão descritos no Quadro 8. Os principais comentários acerca do projeto relacionaram-se à dificuldade de implementação do design do protótipo devido à falta de experiência com acessibilidade em jogos digitais. Pelas mensagens, foi possível constatar-se a maior dificuldade dos desenvolvedores foi em relação a preocupações com o design – muitos não iniciaram o processo de implementação.

[76] A descrição da primeira avaliação pode ser visualizada em https://github.com/francogarcia/uge-evaluation/tree/5fd09c773ff7c130efef38b225a85a3298b291bd.

Quadro 8. Alguns comentários sobre a primeira avaliação.

"Sua documentação está muito bem escrita, mas minha falta de experiencia prejudicou muito. (...)"
" (...) um jogo universalmente acessivel depende muito de game design. Um space invaders, como você citou, seria dificil indicar por som a posição e velocidade de cada inimigo para um cego. Alguns são treinados para conseguirem explorar ambiente por ecolocação (sim, humanos são golfinhos e morcegos), mas a viabilidade de explorar essa habilidade em um jogo é extremamente dependente do game design. Uma coisa importante a se levar em conta, que muita gente esquece de considerar, é que programadores de jogos não são game designers, mas sim programadores - e programadores possuem péssimo senso de design, interface e usuário. Eu poderia tentar testar sua engine fazendo um super mario que escala paredes e no final descobrir que apenas 'usuários médios' ou com deficiencia auditiva (e nenhuma fisica que dificulte o input) podem jogar - se o jogo for jogavel. O jogo existiria, mas possuiria valor nulo para o que parece ser o seu objetivo."
"Eu não tenho experiencia com o Visual C++ da Microsoft, e após várias tentativas criando o projeto, eu não consegui."
"(...) o Guia de referência para o desenvolvedor. Este último tem cerca de 200 páginas (mais até). Eu não sei se terei tempo hábil (provavelmente, não) para fazer a avaliação completa da Engine (...)"
"o material está muito bom! Parabéns!"
"Recebi um e-mail através de um forward falando sobre testar uma engine de jogos em c++. Estou cético quanto a 15h (as competições mais agressivas quanto a tempo dão 72h e muita gente não chega a um nivel de testes)."

A maioria dos desenvolvedores que enviaram comentários elogiaram a documentação do projeto. Um deles relatou que o tempo solicitado para a criação do protótipo (estimado em 15 horas) era inviável. Este é, por sinal, um comentário justo, sobretudo considerando-se que todo o processo foi realizado à distância. O tempo necessário para se familiarizar com um motor e configurar o ambiente de trabalho necessário tende a ser significativo e, portanto, o tempo estimado seria maior[77].

Os comentários obtidos foram considerados para o planejamento de um segundo processo de avaliação.

[77] Embora seja um motor simples, no momento da avaliação, o motor UGE possuía mais de 45.000 (mais de 50.000 neste momento) linhas de código – um valor baixo se comparado aos principais motores existentes, mas significativo para a familiarização.

7.4.2 Segunda Avaliação

Com base nos comentários recebidos pela primeira avaliação, foi elaborado o guia ilustrado *UGE in a Nutshell* (GARCIA, 2014a). O convite estendeu-se à comunidade brasileira e à internacional, incluindo-se o fórum de desenvolvimento de jogos GameDev.net[78] e um convite para o *International Game Developers Association* (IGDA) *Game Accessibility Special Group* (SIG)[79] [80]. Este processo iniciou-se em 4 de abril e está atualmente em andamento.

Para a segunda atividade de avaliação, ofereceu-se um protótipo iniciado (o mesmo apresentado no Capítulo 6) solicitando-se a opinião sobre as abordagens e arquiteturas obtidas sobre o uso do motor para a construção de jogos acessíveis.

Embora o número de visualizações tenha aumentado significativamente, o número de participantes continuou restrito. A provável razão foi a falta de disponibilidade de uma versão binária do protótipo: como a tarefa solicitada incluía programação, apenas foi fornecido o código-fonte para o protótipo.

A recepção do projeto no SIG, entretanto, foi extremamente positiva. Cinco dos participantes mostraram-se muito interessados no projeto (três dos quais são as principais referências para a escrita desta dissertação) e manifestaram apoio ao projeto – como apresentado no início deste capítulo (Quadro 9).

Quadro 9. Alguns comentários recebidos.

"(…) Steve Spohn"
"Welcome to the SIG, Franco! I hope that your project is extremely successful and that others as well, like Steve already has, can comment and help where needed! Sounds like you are off to a great start! :)"
"I've just skimmed through the information and I think it is great to have you here (…) I will look into your presentations during the week and give some better feedback. (…)"
"Hi Franco, nice to meet you!

[78] GameDev.net. http://www.gamedev.net/
[79] IGDA Game Accessibility SIG Page. http://igda-gasig.org/
[80] A descrição da segunda avaliação pode ser encontrada em https://github.com/francogarcia/uge-evaluation.

(…) you still really need to have as widely accessible a default design as possible and use customisation as an enhancement, rather than rely on it exclusively. Or in other words, strive towards getting as close to the unobtainable goal of universal design as possible, and combine that with carefully chosen customisation.
(...)
But that aside, you're absolutely right, it's a really important and effective principle and makes a huge difference. Not just for simple games but in AAAs too. The fact that it is even possible is something that makes accessibility in games pretty unique in the wider field of accessibility.
(…)"

Um dos integrantes (professor em construção de motores para jogos digitais) está, neste momento, realizando uma análise da arquitetura do motor (Quadro 10).

Quadro 10. Comentários sobre o motor UGE.

"I find your project very promising (…)

A few general comments:
- I think the component approach is very good (as opposed to inheritance)
- The layered approach to I/O and game logic is what I teach my students when they make their own engines in courses
- The data driven approach is good coding practice in general
- Customizable IO, yes that rings very familiar (keymaps or high-contrast modes etc)

(…)

I think having the UGE profile becoming a standard format would be the greatest outcome of your engine. That is, if we can get other engine devs to adopt your model and profile format, it would make a big difference. And given that your engine is built on a solid ground with good programming practices, I think it has great potential. May I suggest that you submit a presentation about this for GDC[81] 2015? I will be there in any case and if you need any help with the presentation, please just let me know, I'll be happy to help you out."

Desta forma, os resultados obtidos até o momento classificam o motor UGE como promissor e com excelente potencial, construído sobre boas práticas de programação, e elogiaram a arquitetura e as abordagens escolhidas.

[81] A Game Developers Conference (GDC) é a maior e mais importante conferência de jogos digitais do mundo.

7.5 Considerações Finais

Neste capítulo, apresentou-se as avaliações realizadas para o motor UGE até o momento da escrita desta dissertação. Os resultados obtidos comparando-se o potencial do motor em relação das principais recomendações de acessibilidade descritas na Literatura é promissor: muitas recomendações podem (ou tem potencial) de serem tratadas ao nível de perfil de usuário.

A implementação de jogo simples de *ping-pong* atendeu aos resultados esperados deste trabalho – após a primeira implementação, acrescentar outras tornou-se mais simples. O protótipo de *Space Invaders* apresentado no Capítulo 6 reforça este resultado.

Em relação à avaliação externa, embora os resultados obtidos até este momento sejam poucos, deve-se ressaltar que as principais respostas foram obtidas de especialistas em acessibilidade de jogos digitais – muitos dos quais inspiraram a realização deste trabalho.

Capítulo 8

CONCLUSÃO

"Amateurs sit and wait for inspiration, the rest of us just get up and go to work"

(Amadores sentam e esperam a inspiração; o resto de nós se levanta e vai trabalhar)

Stephen King (*On Writing*)

8.1 Síntese das Contribuições

As principais contribuições obtidas ao longo deste projeto são:

1. A arquitetura resultante da aplicação do roteiro da Seção 5.8;
2. A possibilidade de se reduzir um mundo de jogo digital qualquer em apenas três elementos: ator, componente e evento;
3. O motor UGE propriamente dito;
4. A documentação do projeto:
    a. UGE in a Nutshell (GARCIA, 2014a): 57 slides ilustrados, em inglês;
    b. UGE *Developer's Reference* (GARCIA, 2014b): 280 páginas, em inglês;
    c. Github *wiki*: tech demo comentado (em inglês);
    d. Protótipo de Pong.
5. Os conceitos de Meta-Jogo e Jogo;
6. Os conceitos de Mundo Abstrato de Jogo e Mundo Concreto de Jogo;

7. Os protótipos de jogos universais criados como prova de conceito;
8. O uso de eventos de forma *data-driven* para personalização da interação.

Paradoxalmente, o resultado mais importante deste trabalho não foi o motor UGE propriamente dito – foi o roteiro da Seção 5.8, que reduziu mais de 50.000 linhas de código utilizadas para a implementação do motor UGE em um breve guia de como se realizar *tailoring* em tempo de execução por meio de um perfil de interação. O roteiro agrupa todo o *tailoring* realizado em tempo de execução usando apenas três elementos (ator, componente e evento) e uma fonte de dados externa, que pode variar desde um arquivo até um complexo banco de dados.

Com estes quatro itens, é possível realizar-se a implementação de quaisquer sistemas interativos de forma a torna-los independentes de entrada e saída específica. O princípio é análogo às transformações de Meta-Jogos em Jogos e de Mundos Abstratos de Jogos em Mundo Concretos de Jogos.

Assim, de maneira análoga ao motor UGE, pode-se especializar as entradas e saídas de acordo com as necessidades de interação do usuário – estas necessidades estarão abstraídas no recurso externo. Com isto, desde que seja mapeado um evento gerado a partir do uso do dispositivo de entrada para a comunicação com o sistema, qualquer dispositivo pode ser usado para prover entrada.

Da mesma forma, qualquer dispositivo de saída pode ser usado para apresentar o jogo ao usuário. Basta que exista, ou seja criado, um evento ou a combinação de um componente e de um subsistema para converter os dados internos do software em um estímulo de saída.

A este resultado somam-se todos os benefícios apresentados ao longo desta dissertação sobre o uso do motor UGE – aumentar a flexibilidade para a implementação de jogos digitais e universais. Por meio do perfil de usuário, torna-se possível trocar o foco do jogo para o usuário: as regras do jogo são implementadas uma única vez; o jogo pode ser adaptado para quantas necessidades de interação se desejar.

8.2 Análise Crítica

Há um ano, pretendia-se com este trabalho permitir que um jogo digital pudesse ser apresentado a usuários com diferentes necessidades de interação. Ou seja, apenas que a saída pudesse ser adaptada para satisfazer as necessidades específicas de usuários durante o uso de um jogo. Em seu término, o projeto resultante permite que usuários com diferentes necessidades de interação possam interagir com um jogo – incluindo-se, portanto, entradas e saídas. Ao longo de um ano, passou-se de uma ideia promissora a um projeto promissor, que despertou o interesse daqueles que o inspiraram.

Um projeto que parecia ser impossível de se encaixar no período estabelecido devido à sua complexidade resultou em um sistema que foi além do proposto. Se, por um lado, os protótipos ainda precisam de iterações e avaliações com usuários para tornarem-se jogos completos e acessíveis, por outro, os resultados teóricos obtidos tornaram-se muito mais importantes que os práticos.

O motor UGE é um modelo de que é possível construir um sistema altamente interativo de forma a independer de entradas e saídas. A simulação de jogo ocorre independentemente da existência de um usuário, permitindo-se, posteriormente, especializá-la para adequá-la as interações desejadas ao usuário. Com isto, permite-se a construção de um jogo altamente flexível e completamente configurável.

Com as abordagens adotadas pelo motor UGE, todas as variáveis do jogo são expostas para desenvolvedores e usuários em forma de texto: basta um editor para que sejam alteradas. Como apresentado no Capítulo 6, isto é válido mesmo para versões compiladas do jogo, distribuídas a usuários finais. Todo o *tailoring* para a interação é feito em tempo de execução, não afetando a simulação de jogo – apenas, potencialmente, melhorando a interação ao usuário. Afinal, o elemento mais importante de um software não é o software, mas o usuário.

Ainda é necessário descobrir o quão escalável é a abordagem para aplicações mais complexas. Na pior das hipóteses, obteve-se um motor que avança o estado da arte por permitir que vários dos principais problemas de acessibilidade possam ser ajustados de forma simples – ao menos para jogos simples.

Entretanto, os resultados teóricos são promissores. Os modelos de mundo de UGE são simples, fechados e elegantes. Embora a abordagem tenha sido ilustrada

para alguns perfis, quanto mais componentes, eventos e *handlers* estiverem disponíveis, melhor será o jogo para alguém. UGE trabalha apenas com dados e *tailoring* em tempo de execução. Todo dado não-imutável pode ser alterado. Todo dado de ES pode ser incorporado ou modificado ao perfil desejado.

UGE tornou todo um jogo modificável – desde as partes até as entradas e saídas. As possibilidades de personalização são infinitas. Se a solução for escalável tanto a jogos mais complexos quanto a usuários, obter-se-á resultados ainda mais interessantes: por exemplo, perfis de usuário com deficiência motora poderão ser combinados aos de deficiência auditiva e gerar-se um jogo adequado para múltiplas necessidades de interação. Com um diferencial: grande parte das alterações do jogo são realizadas a partir de um documento de texto.

Resta, então, a pergunta: dados suficientes eventos, *handlers* e componentes, seria possível criar um perfil adequado às necessidades individuais de um usuário?

Se isto for possível, esta dissertação demonstra que o acesso universal em sistemas digitais computacionalmente é viável. Afinal, se existir uma combinação de atores, componentes e eventos que possa permitir a interação e definir as ES, no mínimo, a interação está garantida – basta criar um perfil em texto para combinar os elementos necessários[82]. Talvez, inicialmente, ela não seja ideal – entretanto, sempre é possível editar dados presentes no perfil de usuário para melhorá-la.

Afinal, a solução construída por esta dissertação transforma a acessibilidade em um Lego cujas peças são dados. Entretanto, ao contrário do original, nesta proposta sempre é possível definir novas peças para melhorar o jogo.

8.3 Trabalhos Futuros

Alguns dos trabalhos futuros decorrentes do desenvolvimento deste projeto são:

- Implementação de jogos acessíveis a públicos específicos;

[82] Se existem apenas três primitivas e elas podem possibilitar qualquer entrada e saída, se houver uma conversão que possibilite ao usuário receber o estímulo de entrada e saída, então, na pior das hipóteses, o usuário terá uma péssima experiência de jogo. Não importa – se a interação é possível, ela pode ser melhorada.

- Implementação de jogos universais;
- Avaliações com usuários para melhorias dos protótipos criados e aprimoramento e extensão de funcionalidades existentes no motor;
- Implementação de recursos reusáveis que promovam a acessibilidade para públicos específicos quando da interação com jogos, como sonares e radares em áudio;
- Criação de ferramentas para a edição de perfis de usuário;
- Criação de ferramentas para a edição de atores e componentes;
- Criação de editores de mundo para jogos acessíveis;
- Utilizar linhas de produtos de software para criação de versões do jogo e perfis;
- Integração de tecnologias assistivas ao motor UGE;
- Explorar diferentes técnicas para saídas de subsistemas existentes como, por exemplo, *head-related transfer functions* (HRTF);
- Criação de um editor de mundo acessível, permitindo a usuários finais, com diferentes necessidades de interação, realizarem design e desenvolvimento de jogos (*end-user programming*).

8.4 Considerações Finais

"Amadores sentam e esperam a inspiração; o resto de nós se levanta e vai trabalhar". Assim define Stephen King a arte da escrita: não se obtém resultados sem esforços. Embora esta dissertação tenha avançado o estado da arte e contribuído com o 4º Desafio da Sociedade Brasileira de Computação (acesso participativo e universal do cidadão brasileiro ao conhecimento), ainda há muito a ser feito.

Como apresentado na Seção 5.2, diversas partes envolvidas podem se beneficiar com os resultados. Os trabalhos futuros descritos na seção anterior são apenas parte do que é permitido os resultados obtidos por esta dissertação – poder-se-ia citar muitos mais. Ou seja, embora os resultados obtidos sejam um passo significativo rumo ao acesso universal, eles são um pequeno passo se comparados às possibilidades criadas.

O mais interessante, entretanto, é o último. Com um editor de mundo acessível para a criação de jogos acessíveis, é possível inverter a situação existente: pessoas às quais tinham dificuldades em interagir com jogos podem, agora, tornarem-se desenvolvedores e designers de jogo. Se o acesso participativo e universal do cidadão brasileiro ao conhecimento é um passo extremamente importante, a possibilidade de permitir a criação participativa e universal de conhecimento pelo cidadão brasileiro é um passo ainda mais importante. Passar-se-ia da Inclusão Digital e Social para a criação.

"O resto de nós se levanta e vai trabalhar" – do esforço e a dedicação surgem os resultados. É necessário continuar trabalhando para que se construa um mundo melhor para todos.

REFERÊNCIAS

ADAMO-VILLANI, N.; WRIGHT, K. **SMILE: an immersive learning game for deaf and hearing children**ACM SIGGRAPH 2007 educators program. **Anais**...: SIGGRAPH '07.New York, NY, USA: ACM, 2007Disponível em: <http://doi.acm.org/10.1145/1282040.1282058>. Acesso em: 4 mar. 2013

ADAMS, T.; ADAMS, Z. **Slaves to Armok: God of Blood Chapter II: Dwarf Fortress**. [s.l: s.n.].

ARAÚJO, A. M. L.; GOTO, C. M.; VIOLARO, F. Jogos de Voz. In: **Mobilidade e Comunicação – Desafios à Tecnologia e à Inclusão Social**. Rio de Janeiro: WVA Editora e Distribuidora Ltda., 2000. p. 63–75.

BARLET, M. C.; SPOHN, S. D. **Includification: A Practical Guide to Game Accessibility** (A. Drumgoole, J. T. Mason, Eds.), 2012. Disponível em: <http://includification.com/>. Acesso em: 5 mar. 2013

BELLIK, Y.; BURGER, D. **Multimodal Interfaces: New Solutions to the Problem of Computer Accessibility for the Blind**Conference Companion on Human Factors in Computing Systems. **Anais**... In: CHI '94 - CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS. Boston: ACM Press, 1994Disponível em: <http://dl.acm.org/citation.cfm?id=260482>. Acesso em: 5 set. 2011

BIOWARE. **Dragon Age: Origins**. [s.l: s.n.].

CARVALHO, A. C. P. DE L. F. DE et al. **Grandes Desafios da Pesquisa em Computação no Brasil - 2006 – 2016**. São Paulo: [s.n.]. Disponível em: <http://www.ic.unicamp.br/~cmbm/desafios_SBC/>. Acesso em: 11 set. 2012.

Dark Destroyer. [s.l.] PB-Games, 2004.

DENIS, G.; JOUVELOT, P. **Building the Case for Video Games in Music Education**International Computer Game and Technology Workshop. **Anais**... In: SECOND INTERNATIONAL COMPUTER GAME AND TECHNOLOGY WORKSHOP. 2004Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.7822>. Acesso em: 5 set. 2011

Drive. [s.l.] AudioGames.net, 2003.

ELECTRONIC ARTS. **The Sims Game Portal**. [s.l: s.n.].

ELECTRONIC ARTS. **FIFA 13 - EA SPORTS**. [s.l: s.n.].

ELLIS, B. et al. **Game Accessibility Guidelines: A straightforward reference for inclusive game design**. Disponível em: <http://www.gameaccessibilityguidelines.com/>. Acesso em: 5 mar. 2013.

FAVA, F. Jogando com o ar: o sopro como instrumento de acessibilidade nos jogos eletrônicos. **SBGames '08**, p. 115–121, 2008.

FILGUEIRAS, L. et al. **Personas como modelo de usuários de serviços de governo eletrônico**Proceedings of the 2005 Latin American conference on Human-computer interaction. **Anais**...: CLIHC '05.New York, NY, USA: ACM, 2005Disponível em: <http://doi.acm.org/10.1145/1111360.1111395>. Acesso em: 11 set. 2012

FISCHER, G. **Meta-design: expanding boundaries and redistributing control in design**International Conference on Human-Computer Interaction. **Anais**...: INTERACT'07. In: PROCEEDINGS OF THE 11TH IFIP TC 13 INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION. Berlin, Heidelberg: Springer-Verlag, 2007Disponível em: <http://dl.acm.org/citation.cfm?id=1776994.1777019>. Acesso em: 2 dez. 2011

FREITAS, L. G. DE et al. **Gear2D: an extensible component-based game engine**Proceedings of the International Conference on the Foundations of Digital Games. **Anais**...: FDG '12.New York, NY, USA: ACM, 2012Disponível em: <http://doi.acm.org/10.1145/2282338.2282357>. Acesso em: 7 fev. 2013

GAMES[CC]. **Doom 3 [CC]**. [s.l: s.n.].

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. [s.l.] Addison-Wesley Professional, 1994.

GARCIA, F. E. **UGE in a Nutshell**. Disponível em: <https://github.com/francogarcia/uge>. Acesso em: 5 abr. 2014a.

GARCIA, F. E. **UGE: Developer's Reference**. Disponível em: <https://github.com/francogarcia/uge>. Acesso em: 26 mar. 2014b.

GARCIA, F. E. **UGE: A Game Engine for UA-Games**. Disponível em: <https://github.com/francogarcia/uge>. Acesso em: 26 mar. 2014c.

GARCIA, F. E.; NERIS, V. P. DE A. **Design de Jogos Universais: Apoiando a Prototipação de Alta Fidelidade com Classes Abstrata**Proceedings of the 12th Brazilian Symposium on Human Factors in Computing Systems. **Anais**...: ICH '13.Porto Alegre, Brazil, Brazil: Brazilian Computer Society, 2013aDisponível em: <http://dl.acm.org/citation.cfm?id=2577101.2577120>. Acesso em: 17 fev. 2014

GARCIA, F. E.; NERIS, V. P. DE A. Design Guidelines for Audio Games. In: KUROSU, M. (Ed.). **Human-Computer Interaction. Applications and Services**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2013b. p. 229–238.

GARCIA, F. E.; NERIS, V. P. DE A. **A Data-Driven Entity-Component Approach to Develop Universally Accessible Games**. In: HUMAN-COMPUTER INTERACTION INTERNATIONAL 2014. 2014

GRAMMENOS, D. et al. Access Invaders: Developing a Universally Accessible Action Game. In: MIESENBERGER, K. et al. (Eds.). **Computers Helping People with Special Needs**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. v. 4061p. 388–395.

GRAMMENOS, D.; SAVIDIS, A.; STEPHANIDIS, C. **Unified design of universally accessible games**Proceedings of the 4th international conference on Universal access in human-computer interaction: applications and services. **Anais**...: UAHCI'07.Berlin, Heidelberg: Springer-Verlag, 2007Disponível em: <http://dl.acm.org/citation.cfm?id=1757148.1757218>. Acesso em: 11 set. 2012

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, 1 fev. 2009.

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. 4556p. 607–616.

GREGORY, J. **Game Engine Architecture**. [s.l.] A K Peters, 2009.

IBGE. **Censo Demográfico 2010: Características gerais da população, religião e pessoas com deficiência**. Disponível em: <http://www.ibge.gov.br/home/estatistica/populacao/censo2010/caracteristicas_religiao_deficiencia/default_caracteristicas_religiao_deficiencia.shtm>. Acesso em: 11 set. 2012.

INFOCOM. **Zork Downloads**. [s.l: s.n.].

INTERNATIONAL GAME DEVELOPERS ASSOCIATION. **Accessibility in Games: Motivations and Approaches**. [s.l: s.n.]. Disponível em: <http://www.igda.org/sites/default/files/IGDA_Accessibility_WhitePaper.pdf>.

KOLKMAN, M. **Problem Articulation Methodology**. Tese—Holanda: University of Twente, 1993.

KWEKKEBOOM, B.; VAN WELL, I. **Ilbo**. [s.l: s.n.].

LIN, C.-S. et al. Design of a computer game using an eye-tracking device for eye's activity rehabilitation. **Optics and Lasers in Engineering**, v. 42, n. 1, p. 91–108, jul. 2004.

Lone Wolf. [s.l.] GMA Games, 2000.

MCCRINDLE, R. J.; SYMONS, D. **Audio space invaders**International Conference on Disability, Virtual Reality and Associated Technologies. **Anais**... In: PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON DISABILITY, VIRTUAL REALITY AND ASSOCIATED TECHNOLOGIES. Alghero: 2000Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.4373>. Acesso em: 5 set. 2011

MCSHAFFRY, M. L. **Game Coding Complete, Third Edition**. 3. ed. [s.l.] Charles River Media, 2009.

MCSHAFFRY, M. L.; GRAHAM, D. **Game Coding Complete, Fourth Edition**. 4. ed. [s.l.] Course Technology PTR, 2012.

MILLER, D.; PARECKI, A.; DOUGLAS, S. A. **Finger Dance: A Sound Game for Blind People**International ACM SIGACCESS Conference on Computers and Accessibility. **Anais**... In: 9TH INTERNATIONAL ACM SIGACCESS CONFERENCE ON COMPUTERS AND ACCESSIBILITY. Tempe: ACM Press, 2007Disponível em: <http://dl.acm.org/citation.cfm?id=1296898>. Acesso em: 5 set. 2011

MOBYGAMES. **Space Invaders for Arcade (1978)**. Disponível em: <http://www.mobygames.com/game/space-invaders->. Acesso em: 12 abr. 2014.

Mudsplat. [s.l.] TiM Games, 2005.

NERIS, V. P. DE A. **Estudo e Proposta de um Framework para o Design de Interfaces de Usuário Ajustáveis**. Tese de Doutorado. Disponível em: <http://www.bibliotecadigital.unicamp.br/document/?code=000768432>. Acesso em: 11 set. 2012.

NINTENDO. **New Super Mario Bros. Wii**. [s.l: s.n.].

NYSTROM, R. **Component**. Disponível em: <http://gameprogrammingpatterns.com/component.html>. Acesso em: 11 set. 2012.

OSSMANN, R.; MIESENBERGER, K. Guidelines for the Development of Accessible Computer Games. In: MIESENBERGER, K. et al. (Eds.). **Computers Helping People with Special Needs**. Lecture Notes in Computer Science. [s.l.] Springer Berlin / Heidelberg, 2006. v. 4061p. 403–406.

PALKE, A. **Brainathlon: Enhancing Brainwave Control Through Brain-controlled Game Play**. Dissertação de Mestrado—Estados Unidos: Mills College, 2004.

PETRIE, H.; MORLEY, S.; WEBER, G. **Tactile-based direct manipulation in GUIs for blind users**Conference Companion on Human Factors in Computing Systems. **Anais**... In: CHI '95 - CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS. Denver: ACM Press, 1995Disponível em: <http://dl.acm.org/citation.cfm?id=223769>. Acesso em: 5 set. 2011

PLAYING IN THE DARK. **Top Speed 3**. [s.l.] Playing in the Dark, 2011.

PRESSMAN, R. **Software Engineering: A Practitioner's Approach**. 7. ed. [s.l.] McGraw-Hill Science/Engineering/Math, 2009.

PUC RIO. **Lua**. Disponível em: <http://www.lua.org/about.html>. Acesso em: 26 mar. 2013.

ROCHA, R. V.; ROCHA, R. V.; ARAÚJO, R. B. Selecting the Best Open Source 3D Games Engines. **IX Edition of the Proceedings of the Brazilian Symposium on Computer Games and Digital Entertainment**, n. IX, p. 333–336, 2010.

SCHELL, J. **The Art of Game Design: A book of lenses**. 1. ed. [s.l.] Morgan Kaufmann, 2008.

SCOTT BILAS. **A Data-Driven Game Object System**. In: GAME DEVELOPERS CONFERENCE (2002). , 2002. Disponível em: <http://scottbilas.com/games/dungeon-siege/>. Acesso em: 14 out. 2013

SEGA. **Bayonetta UK**. [s.l: s.n.].

SEHABA, K.; ESTRAILLIER, P.; LAMBERT, D. **Interactive educational games for autistic children with agent-based system**Proceedings of the 4th international conference on Entertainment Computing. **Anais**...: ICEC'05.Berlin, Heidelberg: Springer-Verlag, 2005Disponível em: <http://dx.doi.org/10.1007/11558651_41>. Acesso em: 4 mar. 2013

SOMMERVILLE, I. **Software Engineering**. 9. ed. [s.l.] Addison-Wesley, 2010.

STEPHANIDIS, C. **User Interfaces for All: New Perspectives into Human-Computer Interaction**In C. Stephanidis (Ed.), User Interfaces for All - Concepts, Methods, and Tools (pp. 3-17). Mahwah, NJ: Lawrence Erlbaum Associates (ISBN. **Anais**...2001

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.].

UBISOFT. **XIII**. [s.l: s.n.].

UPS PROJECT. **Guidelines for the development of entertaining software for people with multiple learning disabilities**. Disponível em: <http://www.medialt.no/rapport/entertainment_guidelines/index.htm>. Acesso em: 22 abr. 2012.

UPS PROJECT. **Guidelines for developing accessible games**. Disponível em: <http://web.archive.org/web/20110724181620/http://gameaccess.medialt.no/guide.php>.

WIKIPEDIA. **Space Invaders**, 11 abr. 2014. (Nota técnica).

YUAN, B.; FOLMER, E.; HARRIS, F. Game accessibility: a survey. **Universal Access in the Information Society**, v. 10, n. 1, p. 81–100, 1 mar. 2011.