Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Yohan Duarte Pessanha

Exploratory Software Testing Strategies for Video Games

São Carlos, Brasil
2025

UNIVERSIDADE FEDERAL DE SÃO CARLOS

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

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Yohan Duarte Pessanha, realizada em 27/02/2025.

Comissão Julgadora:

Prof. Dr. André Takeshi Endo (UFSCar)

Prof. Dr. Daniel Lucrédio (UFSCar)

Prof. Dra. Simone do Rocio Senger de Souza (USP)

O presente trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) - Código de Financiamento 2022/13469-6 -, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 88887.801592/2023-00 - e Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Código de Financiamento 444311/2024-6.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Yohan Duarte Pessanha

Exploratory Software Testing Strategies for Video Games

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia 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

Supervisor: Prof. Dr. André Takeshi Endo

São Carlos, Brasil
2025

Acknowledgements

I would like to express my gratitude to everyone who supported me throughout this journey, making the completion of this thesis possible.

First and foremost, I am deeply thankful to my family for their unwavering love, encouragement, and belief in me. To my friends, thank you for your understanding and the moments of joy that helped me throughout this process.

I am deeply grateful to my advisor for his guidance, patience, insights and friendship. If it weren't for him, I would not be in academia. I also extend my thanks to the examining committee for dedicating their time and expertise to evaluate this thesis.

I would like to acknowledge the Federal University of São Carlos (UFSCar) for providing an academic environment and the resources needed to conduct this research. To everyone who contributed, directly or indirectly, to this endeavor.

Finally, to São Paulo Research Foundation (FAPESP) (grant 2022/13469-6), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) (Finance Code 88887.801592/2023-00) and National Council for Scientific and Technological Development (CNPq-Brazil) (grant #444311/2024-6) for funding.

Resumo

A presença de bugs em jogos digitais podem gerar frustração nos usuários, além de impactar negativamente o sucesso comercial dos jogos. Para evitar isso, práticas comuns de desenvolvimento de software podem ser aplicadas para garantir a qualidade geral do produto. Assim como software tradicional, testes também são necessários em jogos, com ênfase no teste manual, especialmente quando considerando a perspectiva do jogador. Um método de teste manual comumente utilizado é o Teste Exploratório - uma abordagem utilizada onde os testadores projetam e executam testes e, ao mesmo tempo, aprendem com os resultados para realizarem uma nova iteração de testes. Apesar da vasta literatura em teste exploratório, existe uma lacuna nas discussões sobre sua aplicação em testes de jogos. Isso pode ser um desafio para novos testadores, por não encontrarem guias sobre utilizar o teste exploratório efetivamente nesse domínio. Esta dissertação investiga o uso de teste exploratório em jogos. Para atingir esse objetivo, dois estudos foram conduzidos. No primeiro estudo, aplicamos sete estratégias de teste exploratório conhecidas na literatura em cinco jogos de plataforma. No segundo estudo, elaboramos um framework para aplicação de teste exploratório em jogos de plataforma, realizando uma avaliação de aplicabilidade em um jogo de plataforma em desenvolvimento por um estúdio independente. Os resultados obtidos foram positivos, sendo diversos bugs identificados, o que fornece evidências iniciais de que esta adaptação é viável e promissora quando considerando o gênero de plataformas.

Palavras-chave: Bugs. Jogos Digitais. Teste de Jogos. Teste Exploratório.

Abstract

The presence of bugs in video games can cause user frustration and negatively impact the commercial success of games. To prevent this, common software development practices can be applied to ensure the overall quality of the product. Similar to traditional software, testing is also necessary in games, with an emphasis on manual testing, especially when considering the player's perspective. A commonly used manual testing method is Exploratory Testing - an approach in which testers design and execute tests while simultaneously learning from the results to perform new test iterations. Despite the extensive literature on exploratory testing, there is a notable gap in discussions about its application in game testing. This can be a challenge for new testers, as they lack guides on how to effectively use exploratory testing in this domain. This thesis investigates exploratory testing for games. To achieve this objective, two studies were conducted. In the first study, we applied seven exploratory testing strategies known in the literature to five platform games. In the second study, we developed a framework for applying exploratory testing to platform games and conducted an applicability assessment using a platform game under development by an independent studio. The results were positive, with various bugs being identified, providing initial evidence that this adaptation is both viable and promising when considering the platformer genre.

Keywords: Bugs. Video Games. Game Testing. Exploratory Testing.

Declaration of Original Authorship and List of Publications

I confirm that this thesis has not been submitted in support of an application for another degree at this or any other teaching or research institution. It is the result of my own work and the use of all materials from other sources has been properly and fully acknowledged. Research done in collaboration is also clearly indicated.

Excerpts of this thesis have been either published or submitted for the appreciation of editorial boards of journals, conferences and workshops, according to the list of publications presented as follows. My contributions to each publication are listed as well.

Journal Paper:

- **Duarte, Y.**; Mandelli, H. C.; Durelli, V.; Nardi, P. A.; Endo, A. T.: "Exploratory testing for platform video games: strategies and lessons learned".

  - **Journal:** Journal on Interactive Systems, 2024.

  - **DOI:** 10.5753/jis.2024.4156.

  - **Level of Contribution:** High - The candidate is the main investigator and conducted the work together with his contributors.

Conference and Workshop Papers:

- **Duarte, Y.**; Durelli, V.; Nardi, P. A.; Endo, A. T.: "Exploratory testing strategies for video games: an experience report".

  - **Event:** 22nd Brazilian Symposium on Games and Digital Entertainment (SBGames 2023).

  - **Level of Contribution:** High - The candidate is the main investigator and conducted the work together with his contributors.

- **Duarte, Y.**; Politowski, C.; Endo, A. T.: “A Framework for Exploratory Testing in Games”.

  - Event: 9th International ICSE Workshop on Games and Software Engineering (GAS 2025).

  - Level of Contribution: High - The candidate is the main investigator and conducted the work together with his contributors.

- Copche, R.; **Duarte, Y.**; Durelli, V.; Eler, M.; Endo, A.: “Can a Chatbot Support Exploratory Software Testing? Preliminary Results”.

  - Event: 26th International Conference on Enterprise Information Systems (ICEIS 2024).

  - Level of Contribution: Medium - The candidate helped in the definition of approach, initial testing and paper writing.

List of illustrations

List of tables

List of abbreviations and acronyms

AI	Artificial Intelligence
ET	Exploratory Testing
GSE	Game Software Engineering
GUR	Game User Research
NPC	Non-Player Characters
SBT	Session-Based Testing
SE	Software Engineering
SUT	System Under Test
UI	User Interface
UX	User Experience

Summary

Chapter 1

Introduction

The video games[1] industry generated $180.3 billion in 2021 (CHUECA et al., 2024) and is projected to reach $363.19 billion by 2027 (CLEMENT, 2023a). Games are formal, rule-based systems with consequences and outcomes attributed to values influenced by player effort (JUUL, 2003). Novak (2017) categorizes games into genres based on the combination and presentation of several elements; the author mentions and explains some game categories, such as: action, where the player should destroy enemies while avoiding destruction; race, where the player competes against adversaries; fight, where the player controls characters to attack or defend; and adventure, where the player explores, collects items and solves puzzles.

Similar to traditional software, games are expected to exhibit minimal failures, necessitating systematic testing. Myers, Sandler and Badgett (2012) define software testing as a process, or series of processes, to ensure the intended behavior and prevent unintended actions. However, game testing differs from traditional software testing both in its steps and priorities (POLITOWSKI; PETRILLO; GUEHENEUC, 2021). There are differences in requirements and scope definitions of the project, game tests focus mainly on overall experience than accuracy, prioritizing feedback based on "enjoyment heuristics", and are mostly based on black-box testing (POLITOWSKI; PETRILLO; GUEHENEUC, 2021). Additionally, technical aspects of testing play a supporting role, while the primary focus is on achieving quality that satisfies the player (KASURINEN; SMOLANDER, 2014).

When testing games, manual testing[2] evaluates user logic (player) and how they adapt with commands and environments. This approach is widely adopted due to the challenges associated with test automation, including code highly integrated with interface,

[1] In this thesis, the term "game" will refer to video games.

[2] Tests crafted and performed manually by a human tester without using automated scripts.

non-determinism, and wide state space to explore (MURPHY-HILL; ZIMMERMANN; NAGAPPAN, 2014). Among the existing software testing approaches, Exploratory Testing (ET) stands out in practice (ITKONEN, 2011; GHAZI et al., 2017; MåRTENSSON et al., 2021; LEVEAU et al., 2022). According to Whittaker (2009), ET is an approach in which a tester, without using scripts, creates scenarios in an attempt to find bugs or issues in a software. These scenarios evolve through various iterations of learning, test design, and test execution on the software (BACH, 2003). Lyndsay and Eeden (2003) state that unscripted exploratory tests can be more effectively managed and controlled through Session-Based Testing technique (SBT), during which a test session is an uninterrupted period where testers investigate one or more test goals.

Knowledge obtained during ET sessions should guide and refine the subsequent sessions. Hendrickson (2013) states that, by carrying out a series of undocumented testing sessions, ET aims at enhancing testers' skills and creativity while allowing a deep exploration of the System Under Test (SUT). From its benefits and objectives, we surmised that ET can be employed to help uncover problems in games.

Games are the most profitable sector of the entertainment industry (POLITOWSKI et al., 2021) and its trajectory of growth and market evolution demonstrates the increase of complexity and necessity of testing practices (SCHULTZ; BRYANT, 2005). Cyberpunk 2077 illustrates the consequences of game-related issues that, according to CD Projekt Red CEO and co-founder Marcin Iwiński, could have been avoided if testing were carried out properly (POLITOWSKI; PETRILLO; GUEHENEUC, 2021). At that time, the game had the largest digital launch in history, with 10.2 million digital copies sold (BANKHURST, 2021), but it had many delays in the release, followed by a large number of bugs involving everything from performance to gameplay issues. These issues generated a frustration among players, which, combined with the game's initial negative reviews, led to approximately 9% of players requesting a refund (DEMARTINI, 2021). Examples like this underscore the relevance of comprehensive testing applied to games, as various bugs could be avoided with better planning and structuring of the tests.

Many companies have already adopted ET in game development. Kasurinen and Smolander (2014) claim that the game companies featured in their study, despite having the budget for conducting technical tests, choose exploratory and usability testing to enhance the user experience (UX). However, studies addressing testing in games primarily focus on automated testing (IFTIKHAR et al., 2015; SRIRAM, 2019; PEREIRA et al., 2021; LI et al., 2022), contributing to a gap in research on manual testing. On the other hand, studies focusing on manual testing typically emphasize the game user research (GUR), UX and usability rather than bugs, issues and more technical aspects (CHOI et al., 2016; DRACHEN; NACKE; MIRZA-BABAEI, 2018; AZIZI; ZAMAN, 2024).

In this context, this master thesis investigates ET for games, to enhance bug detection and improve product quality. We elicited ET practices and strategies from games and

traditional software combined with game concepts, tailoring them to the aspects of game testing. Finally, two studies were conducted to assess the cost and effectiveness of ET in games. In the first study, we applied seven ET strategies to five already released platform games. In the second study, we developed and evaluated a framework for ET in platform games.

The remainder of this text is organized as follows: Chapter 2 covers key concepts related to game development, software engineering in games, software testing, ET strategies; it also reviews the state of the art in game testing. Chapter 3 presents our study adapting Whittaker's (2009) ET strategies to the context of game testing, evaluating five games (two 3D platformers and three 2D platformers) using this approach. Chapter 4 details the development of an ET framework, and its evaluation of applicability in an indie game. Chapter 5 concludes the thesis, revisiting the contributions and future directions.

Chapter 2

Background

2.1 Overview

This chapter presents key concepts to this thesis. Section 2.2 introduces game concepts and terms, its differences with traditional software, development tools and platforms. Section 2.3 examines the application of Software Engineering in the game development context, addressing challenges and project management practices. Section 2.4 explains foundations and terminologies of software testing. Section 2.5 describes Exploratory Testing and some strategies relevant for the understanding of this thesis. Section 2.6 explains foundations of game testing, as well as phases of production and the state of the art of manual and automated playtesting.

2.2 Games

Given that games are formal rule-based systems, in which consequences and outcomes are attributed to different values influenced by player effort (JUUL, 2003), it is crucial for a game to exhibit good gameplay in order to be considered engaging. Desurvire et al. (2004) define gameplay as the set of problems and challenges faced by the user to win a game, and categorize it as one of the four game heuristic categories, alongside with: game story, which encompasses all plot and character development; game mechanics, involving the programming that structure how units interact with the environment; and game usability, which addresses the interface and includes the elements the user utilizes to interact with the game (e.g., controllers, keyboard, mouse).

Game quality is essential to keep players engaged. Aleem, Capretz and Ahmed (2016) argue that only good games can keep players interested and state that the quality of a

game can be defined based on the sounds, graphics and compiled code. However, there are other factors that can impact a game quality, such as level design. Smith and Whitehead (2008) describe level design as a complex task, which requires an understanding of every component of the game and how to merge them. The authors also affirm that a level offers players an interactive environment in which they have the opportunity to explore it within the context of the game world's rules.

Games share some similarities with traditional software, particularly concerning code size and complexity (BORG et al., 2020). However, despite these similarities, Engström et al. (2018) assert that games cannot be merely regarded as software products, but also as creative products, works of art, or cultural expressions. The authors point out that even employing certain software processes and standards, differences exist in the development process. These differences are particularly evident in the need for iterative feedback and UX refinement, which are critical to the success of game development.

Given that game developers need to receive extensive feedback on UX to refine their games, agile methods have been adopted by various game studios (LIU et al., 2019). Agile methods are flexible and adaptable, making them suitable for various types of software projects (PRESSMAN, 2010).

A game genre is defined by the merging of theme, settings, interface, perspective and strategies (NOVAK, 2017), and it is common to fit more than one when combining elements that span to diverse categories. Every genre has its own niche and standout games. Some examples of common genres include action, adventure, strategy, and RPGs. It is possible to further specify a game's classification by using subgenres. Furthermore, the classification may be influenced by problems and challenges faced by the player, storyline and mechanics. These factors, when combined, define playability (AL-AZAWI; AYESH; OBAIDY, 2013).

To deliver a great experience to the player, it is important that a game is well balanced. Schell (2007) defines balance as the adjusting of the elements of the game until they deliver the experience the designers want to. Although game balance is important in various aspects and games, in online gaming communities, this can have great impact and leave players dissatisfied with how it affects the game. An example of this can be seen in "League of Legends" (RIOT GAMES, 2024), a well-known team-based strategy game of multiplayer online battle arena (MOBA) subgenre. In this title, players are divided into two teams of five with the objective of destroying the enemy team's base. The game releases new champions periodically (SATTAR, 2024) and even though it has a public beta environment (PBE), it suffers from balancing a new champion in a way that pleases the players.

Game characters are divided into player characters (PCs) and non-player characters (NPCs) (NOVAK, 2017). NPCs are characters that are not controlled by the players; instead, they are created and controlled by the artificial intelligence (AI) of the game,

ranging from a merchant to a monster the player should defeat (NOVAK, 2017).

There are several digital platforms where games are distributed, such as Steam (VALVE, 2024), Epic Games (EPIC GAMES, 2024), EA app (ELECTRONIC ARTS, 2024) and Battle.net (BLIZZARD ENTERTAINMENT, 2024). In them, players can interact with each others, create communities and modified contents (mods). Steam is the largest platform with an average of over 130 million monthly active users in 2021 (CLEMENT, 2023b) and over 1 billion registered users in 2020 (XPAW, 2020).

Indie games[1] are not always published in these platforms, primarily due to publishing fees and revenue sharing, which can have an impact on the developers outcome. Instead, one of the main platforms used by indie developers to release their games is Itch.io (ITCH, 2024) and it only has revenue share if the seller wants to.

To accelerate the development process of a game, developers use game engines (e.g., Unity, Unreal, Godot), which are development environments that integrate a graphics engine and a physics engine as well as a set of tools that wraps around them (CASAMAYOR et al., 2022). Sometimes big studios also develop their own game engine (e.g., CryEngine, Creation Engine, Fox Engine, Frostbite, RAGE) (CASAMAYOR et al., 2022).

Some game concepts important to the understanding of the next chapters are briefly explained next. Speedrun (SPEEDRUN.COM, 2014) is when players try to complete the game or part of it as fast as possible. Golden Path (DEVELOPER, 2019) is when players make decisions that align with the designers' intended experience. Completionists (CRUZ; HANUS; FOX, 2017) are players that complete every available task and collect all badges the game offers. Secret (BURGUN, 2023) is a piece of content within the game that can be missed by the player.

2.3 Software Engineering for Games

Kanode and Haddad (2009) state that game development, despite being unique, is similar to other software endeavors. This means that a lot of challenges faced in game development can be solved using Software Engineering (SE) practices, adapting and evolving it when necessary. The authors identify challenges in game development and propose SE practices to overcome them. For instance, they suggest optimizing tools and pipelines for integrating diverse assets into the game and investing in training focused on project management practices to address project management issues.

Politowski et al. (2021) point out that the game industry is known for its problems and faces a wide range of them, spanning from technical to management issues, and also including mistreatment of employees and harassment (WOOD, 2023; PARRISH, 2023). They identify the top ten most common problem subtypes, such as: insufficient workforce, environment problems, wrong marketing strategy, underestimation, unclear game design

[1] Indie games are games developed by an independent studio or developer.

vision, lack of fun, platform and technology constraints, game design complexity, inadequate or missing tools, and misaligned teams. The authors also emphasize that these problems are mainly related to people rather than technologies, indicating that this area could improve.

Aleem, Capretz and Ahmed (2016) state that game development is a complex process and its enhancement is directly associated with team configuration and management, game design document management, game engine development, game test management, and programming practices. Additionally, the authors found no strong association between the quality of game architecture and the game development process improvement.

Ampatzoglou and Stamelos (2010) assert that game developers must employ specific SE techniques in order to achieve high quality standards. This is necessary since, to implement a commercially viable game, the complexity of the low level code is hundreds of thousands lines, which demands adaptable design, maintainable implementation and well-written documentation. However, despite the application of various SE techniques in games, one of the requirements is unique to games as opposed to traditional software - games must be fun, also mentioned as *fun factor* - and there is no SE metric that is applicable in this context, as it is a subjective concept (LEWIS; WHITEHEAD, 2011).

Engström et al. (2018) affirm that SE field has a great relevance on software development aspects but has limited applicability in managing creativity. By their definition, games are advanced software products and complex works of creativity and art. The authors also state that game companies have been embracing agile principles, given that game production is not solely led by software and the entire team, formed by composers, animators, graphic artists, game writers and game designers, is important to the production.

Chueca et al. (2024) conducted a systematic literature review of SE for computer games, identifying that researches on Game Software Engineering (GSE) are increasing in interest and the industry may benefit from it. The authors argue that GSE is an independent, mature and growing domain and the gap between industry and academia is narrowing. They also highlight the main similarities and differences between GSE and SE.

They stated the following as main similarities: the design decisions are based on business and commercial influence; technical work in both domains complies with the requirements specified in the project; in project scope, certain features and designs are excluded; project management is similar in both fields, even though team size and roles are not the same, as the agile methodologies from traditional SE are usually applied in game development.

As for the differences, the authors pointed out the following ones: game development teams are multidisciplinary, with the work pipeline requiring the integration of non-engineer specific roles (e.g., art, animation, music); non-agile methodologies is not

common in game development, since game development process is iterative; it is common for game developers to have strict deadlines in a market that postpones can negatively influence the business; players expect large amounts of content as a result of the industry's evolution, and this demands from the developers the need to manage a substantial quantity of assets; testing in games also focuses on UX (e.g., aesthetics, interactive mechanics, storytelling) and game-specific testing (e.g., mechanics, level design, controls, balancing).

The authors also emphasized that developers must test the artistic nature of their products besides the technical testing; as post-release features and content addition are standard practices, the life cycle of a game can be extended years after its release; as opposite to traditional software, user-driven requirements in games have a lot of subjective concepts such as fun, entertainment, aesthetics, flow and creativity; the integration of complex software systems (e.g., sound, rendering, physics, AI, 2D and 3D rendering) is necessary to an understandable and enjoyable game experience; the complexity of games usually makes developers utilize game engines, requiring expertise not only in their usage but also in additional skills beyond software frameworks, including art, music, and design. They often use scripting languages or domain-specific modeling languages that do not necessarily align with the code language of the engine in order to ease the design and implementation process.

Murphy-Hill, Zimmermann and Nagappan (2014) conducted both an interview and a survey to understand the differences between game development and software development. Among the findings obtained by the authors, they mentioned the technical challenges of game development, the problem that is the subjectivity in meeting the "fun" requirement, the variety of skills required to develop a game, the stronger tie gamer developers usually have with their customer since it is difficult to meet "fun" requirement without understanding the customer, game developers have less clear requirements than non-game developers, and creativity is valued more in game development teams.

2.4 Foundations of Software Testing

Testing during software development is essential to ensure the quality of a software product, revealing bugs before it is utilized by users (ITKONEN, 2011). Kaner, Falk and Nguyen (1999) define software testing as the process of seeking errors and posit that testing is directly related to software quality. They assert that, regardless of the planning, time, people, and resources invested, it is not possible to test a software completely.

Schultz, Bryant and Langdell (2005) define a test case as a detailed description of an individual test to be conducted by testers, including what operations to perform in order to meet its objective, being each individual operation a test step. Ammann and Offutt (2016) point out that a test case is composed of the combination of its values, prefix values, postfix values and expected results. They describe test case values as the inputs

used by test designers to directly fulfill test requirements, prefix values as the inputs required to prepare the software for receiving the test case values, and postfix values as the inputs that need to be sent to the software after the test case values.

Myers, Sandler and Badgett (2012) define that software testing has two prevalent techniques: (i) black-box testing, where the software is seen like a black box and the tester should not be concerned about internal structure of the program, focusing the tests in the software functionality; (ii) white-box testing, where the tester can examine the internal structure of the program, focusing in logic, flow and code structure.

The most common types of levels in which testing is applied are:

- **Unit testing:** the lowest level of testing (AMMANN; OFFUTT, 2016). It assesses the units produced by the implementation phase and, sometimes, is done without the knowledge of the encapsulating software application (AMMANN; OFFUTT, 2016);
- **Integration testing:** it ensures correct integration of subcomponents into a larger component, identifying errors that may arise from unexpected interactions among them (AMMANN; OFFUTT, 2016);
- **System testing:** it compares the software to its original objectives, ensuring it satisfies both the functional and non-functional requirements (AMMANN; OFFUTT, 2016).

Manual and automated testing are approaches complementary to each other: while manual testing uses the knowledge of the tester to target the error-prone parts of the SUT, the automated testing can cover a greater area of tests in a shorter period of time (LEITNER et al., 2007). Manual testers are often guided by scripts, which provide input selection and dictate how the software results are checked for correctness (WHITTAKER, 2009).

2.5 Exploratory Testing

When tests are performed without the use of scripts, they are classified as ET, where the tester can test the SUT in any way they want and generate documentation while conducting the tests (WHITTAKER, 2009). The downside is that the lack of tester experience can directly influence the expected outcomes, as the tester's personal knowledge in failure detection is important in this approach (ITKONEN, 2011). Bach (2003) describes ET as a dynamic process that involves learning, designing, and executing tests simultaneously. ET sessions leverage insights gained from each iteration to improve subsequent ones, establishing a continuous enhancement cycle, as illustrated in Figure 1. The author also outlines specific scenarios where ET is beneficial, such as finding the most important

bug in the shortest time, quickly learning about the product, providing rapid feedback, and investigating and isolating a particular defect.

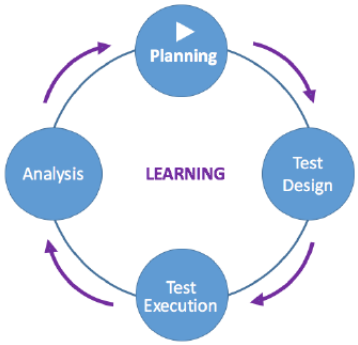

Figure 1 – ET cycle (COPCHE et al., 2021).

The management and control of ET can be achieved through Session-Based Testing (SBT) (LYNDSAY, 2003). This technique involves planning, managing, and tracking the progress of tests in sessions lasting up to a maximum of 12 hours. Its composition is a charter, time limited sessions, results, and questions, with no requirement to detail the techniques or strategies to be used during the sessions (ITKONEN, 2011). Charters are the main objectives of the testing session and present various functionalities that should be explored during the sessions (SOUZA et al., 2019).

Whittaker (2009) proposes several strategies for efficiently conducting ET sessions. These strategies represent methods for testers to employ and have inspired numerous authors. Seven strategies, also utilized by Micallef, Porter and Borg (2016), are discussed next:

1. **Tour Bus Strategy**: the tester takes a "tour" of the system, stopping at any feature as desired for a short period of time, and then returning to the main route. Similarly to a guided tour in a city, the hallmark of this exploration strategy is stopping at places of interest (i.e., important features).

2. **Exploratory Smoke Testing**: the tester randomly checks if the system's features are functioning properly, without following a pattern or rules. The tester uses inputs without regard to what features have and have not been covered (WHITTAKER, 2009).

3. **Crime Spree Tour**: the tester focuses on a specific feature and its neighborhood with the intention of breaking it (i.e., uncovering problems). For example, the tester might try to input negative numbers when the application expects a positive one (WHITTAKER, 2009).

4. **Garbage Collectors Tour**: the tester selects an objective, finds the fastest way to accomplish it, performs tests, and moves on to the next objective. Similarly to a

garbage collector, traveling as fast as possible between neighborhoods and stopping at each house. The tester could select a goal (all menu items) and visit each one of the list adjusting the shortest path possible (WHITTAKER, 2009).

5. **Back Alley Tour**: the tester focuses on exploring the less frequently used features of the system. It is the opposite of a guided tour, where popular places would be visited (WHITTAKER, 2009).

6. **User Interface Exploration**: the tester learns by exploring the user interface (UI), understanding the functionalities of different parts of the UI, and testing their behaviors. The tester could look if the UI elements render properly, look good, are consistent with other buttons and elements, as well as if it violates any convention or standard (WHITTAKER, 2009).

7. **Bad Neighborhood Tour**: the tester examines the "neighboring" features of where a bug was found, aiming to uncover other problems. This strategy is founded upon the analogy drawn from a neighborhood that tourists should avoid due to its proximity to a troubled area (WHITTAKER, 2009).

Itkonen (2011) conducted a study on the ET approach to define and understand its applicability, investigate its benefits and shortcomings, and provide results on how ET is applied in practice. The findings indicate that ET can be as effective as test case-based approaches, or even more so, while requiring less effort; and that ET is an applicable approach to engage domain experts in testing.

There are studies investigating the use of the ET approach in specific areas, such as mobile or web applications (SOUZA et al., 2019; DI MARTINO et al., 2024; LEVEAU et al., 2022). Other authors propose tools to support ET. Copche et al. (2024) developed a chatbot[2] on the Discord social platform to support testers during ET of software applications. The chatbot includes features for testers to manage ET sessions, report bugs and issues, and has features to enhance their ability to gain insights for further exploration of the SUT. The first version was evaluated with six software development professionals. Among the results, the chatbot received positive feedback from the participants and it was provided evidence that it has the potential to be as effective as similar approaches.

Mårtensson et al. (2021) proposed a model to support testers in the industry by optimizing ET. To do so, the authors conducted two series of interviews with professionals from six companies, where they identified key factors to efficient and effective ET. Among the key factors identified by the authors, are: the testers should know how the product is used by the end-user; a well-defined purpose and scope for the tests; testers with different experiences should work together as a team; a well-defined way to report the tests results.

[2] A chatbot is a software program that interacts with users using natural language, replicating a human-like conversation (SHAWAR; ATWELL, 2007).

2.6 Game Testing Foundations

Several games feature long or challenging levels that can take hours to complete. In such instances, bugs that prevent level progression or undermine gameplay can frustrate players. Lin, Bezemer and Hassan (2017) analyzed 2,419 update notes from the 50 most popular games on the Steam platform. Their findings revealed that 80% of these games issued urgent updates; 46% featuring self-admitted hotfixes, with only 10% of the 0-day updates were self-admitted hotfixes, which suggests developers' efforts to mitigate mistakes discreetly; 36% of urgent updates aimed to modify the rules of the game. The most common reasons for releasing urgent updates are feature malfunctions, crashes, and visual bugs.

Schultz, Bryant and Langdell (2005) state that almost all tests performed on games are black-box tests. In such cases, tests are executed based on inputs provided by the tester and the corresponding outputs received, creating a loop, as depicted in Figure 2. The authors also define stages when conducting tests in games:

1. *Pre-production*: the testing planning document begins;
2. *Kickoff*: it addresses critical issues related to the software, serving to improve quality and test execution;
3. *Alpha*: testing is initiated, and all game modules should be tested. However, at this stage, the game is not yet fully finalized, so tests are conducted in a limited way;
4. *Beta*: testers can play without the restrictions imposed in the *Alpha* phase, and the tests conducted prioritize game refinement, addressing bugs that were not identified in the previous stages and new bugs can be discovered;
5. *Gold*: all test suites are executed again on what is considered the final version of the game, delaying the release in case any bugs are found;
6. *Release*: the released game is tested, and any bugs uncovered at this stage are discussed and, if necessary, slated for fixing;
7. *Post-Release*: the identified bugs are resolved through patches or updates. However, when this occurs, the entire bug list must be reviewed, and the changes should be tested for compatibility with the game and previous updates;

Zheng et al. (2019) categorize the bugs found in games in five different types: **Crash bugs**, which leads the entire game to crash and exit; **Stuck bugs**, which freezes the game; **Logical bugs**, which lead to unexpected results but does not break the game; **Gaming balance bugs**, which disturb the balance between human and computer-aided players; and **User experience bugs**, which downgrade the UX.

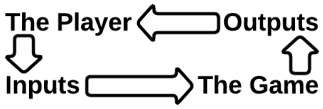

Figure 2 – Player's input and game output loop. Adapted from Schultz, Bryant and Langdell (2005).

Santos et al. (2018) conducted a case study and a survey on the particularities of software testing in game development. The authors pointed out that many aspects can differ from game to non-game testing, such as physics of the game, variety of possible test scenarios and flows, variable requirements, scope definitions of the project and metrics that are not easily measured (e.g., fun, entertainment).

Kasurinen and Smolander (2014) state that game testing results have a more significant impact on the end product than software testing due to focus in soft aspects of the product (e.g., internal mechanics, game rule balance, UX). The authors also emphasize that game developers prioritize different aspects compared to software developers, particularly evident when organizations, despite having resources for technical testing, opt for exploratory and usability testing to refine the UX, setting aside technical concerns.

2.6.1 Manual Playtesting

Game testing tends to be a more human-centered activity due to the difficulty in separating the UI from the rest of the game and the influence of human behavior, which is challenging to automate (SANTOS et al., 2018). Problems in games can be related to coding, designing, balance, real-time event occurrences, object boundaries, and other factors (SANTOS et al., 2018).

As reported by Politowski, Petrillo and Guéhénuec (2021), manual gameplay testing (i.e., a form of system testing also referred to as playtesting) is by far the most used testing technique by game developers. Mirza-Babaei, Moosajee and Drenikow (2016) performed eleven playtesting case studies on six different commercial indie games. They state that there is a need for indie developers to adopt Game User Research (GUR) techniques (e.g., behavioral observation, think-aloud protocols, interviews) into their development pipelines in order to ensure the creation of high-quality products that can effectively compete in the market.

2.6.2 Automated Playtesting

Although manual testing is still the *de facto* approach for gameplay testing, automated solutions have been proposed in the literature. Iftikhar et al. (2015) propose an automated testing strategy for games where a state machine is used to generate and execute tests

automatically. The authors acknowledge that there is an investment of time and resources in developing these test models. However, they assert that once ready, the generation and execution of automated tests save the effort required by manual testing.

AI methods have also been applied in test automation and automated playtesting (JUSTESEN et al., 2019; ARIYUREK; SURER; BETIN-CAN, 2023; SRIRAM, 2019). For instance, Sriram (2019) introduces an AI approach to perform automated testing in 2D platform games. Among the limitations encountered by the author, the created agent does not follow paths similar to those of a human tester, presenting the lack of a human perspective in the testing process. The author also mentions the limitation of the number of mechanics to be tested by the agent, as it was unable to keep up with the variety of mechanics presented by 2D platform games.

Zheng et al. (2019) present an automated testing framework for games, specifically combat games, that combines evolutionary multi-object optimization and Deep Reinforcement Learning. The authors realized the tests in one simple game and two large commercial games, obtaining better effectiveness than other automated strategies using evolutionary algorithms and deep reinforcement learning.

Politowski et al. (2023) propose an approach to support game testing to balance games using autonomous agents. The authors validated their testing process with two platform games, focusing on two game balance types: “challenge vs success” and “skill vs chance”. One of the challenges was to define reward functions for agents to play the game appropriately. The authors state that although it is not possible to replace manual testing, a development pipeline with automated testing can be adopted by the developers to provide quick feedback about the game balance.

Pereira et al. (2021) bring an automated playtesting approach built with the game code, using Unity Test Framework (i.e.: a fully integrated framework for unit testing in Unity). In their approach, they combined unit tests with automated playtesting so the game could be played without any input from a player. The authors limited themselves to the development of a simple 2D game and used Unity as a game engine for the study.

Sestini et al. (2022) propose a reinforcement learning approach designed for the automatic playtesting of intricate 3D scenarios. The authors utilized the Curiosity-Conditioned Proximal Trajectories (CCPT) method, which combines imitation learning and curiosity to train agents that perform exploration in the proximity of demonstrated trajectories. Their outcomes revealed a substantial area of coverage and successful bug discovery in their navigation environment. The authors emphasized the effectiveness of curiosity and imitation blend that CCPT uses, noting how well this combination serves the intended purpose of identifying issues.

Politowski, Guéhénuec and Petrillo (2022) performed an exploratory study investigating the gap between the solutions for automated game testing in the academic literature and what game developers need in practice. In their survey, results indicate that game

developers are skeptical of using automated agents to test games. They also concluded that there is still a long way to go for game testing, especially on how we should test games.

Li et al. (2022) proposed a game bug database and extensive framework designed to enable automated game testing research. They selected 76 bugs from five different industrial games, in collaboration with their respective developers. Additionally, the authors integrated five automated game testing techniques into their framework for comparative study and as baseline for research. Their results suggest automated game testing is still at an early stage and its methods have room for improvements.

Lovreto et al. (2018) conducted a study on automated testing for mobile games, with the aim of assessing the viability and efficiency of employing automated test scripts in this context. The authors randomly selected 16 games from Google Play Store and, for each of them, designed, configured, coded and ran test scripts. The authors pointed some challenges in automated testing mobile games, such as recognition of graphical elements: which caused failed test cases since some buttons, due to how template matching works, were not recognized; loading times and ads: where ad screens popped up on screen requiring an input to exit caused failures in the tests, and loading times affected the performance and execution time of the tests; active player events: complex inputs required by some games was not always accurate to reproduce.

2.7 Concluding Remarks

This chapter introduced games and testing concepts and techniques, accompanied by characteristics that differentiate games from conventional software, especially in Software Engineering. It also elucidated concepts and related studies on game testing, examining its foundations, manual and automated approaches. The next chapter will exhibit the first study, adapting ET strategies into the game context.

Chapter 3

Exploratory testing for platform video games

3.1 Overview

This chapter presents a study, in which five platform games were tested using seven strategies adapted to the gaming context. These strategies, proposed by Whittaker (2009) and Micallef, Porter and Borg (2016), are User Interface Exploration, Garbage Collector's Tour, Tour Bus Strategy, Back Alley Tour, Crime Spree Tour, Exploratory Smoke Testing, and Bad Neighborhood Tour.

The study setup is described in Section 3.2, and the results are detailed in Section 3.3. Section 3.4 discusses the threats to the validity of the study, while the lessons learned are presented in Section 3.5.

This chapter summarizes the main results of the paper "Exploratory testing for platform video games: strategies and lessons learned"[1], Duarte, Y.; Mandelli, H. C.; Durelli, V.; Nardi, P. A.; Endo, A. T., published in Journal on Interactive Systems, 2024, DOI: <https://doi.org/10.5753/jis.2024.4156>.

3.2 Study Setup

In this section, we introduce the study setup, including games under test and testing procedure. As mentioned, ET has been widely used in practice and has a vast literature about its adoption in traditional software, however, little is known about its applicability

[1] This paper is an extension of our previous study (DUARTE et al., 2023), which was invited for submission following its publication at the Brazilian Symposium on Digital Games (SBGAMES 2023).

to games. We undertook a study to evaluate the applicability of ET in the context of games. Thus, we conducted our study to answer the following research question (RQ):

- *How applicable are the ET strategies to games?*

We investigated two key aspects: *(i)* how a subset of established strategies for ET can be applied, and *(ii)* whether or not they are capable of helping game testers to uncover bugs.

3.2.1 Games Under Test

We conducted our study with both 2D and 3D platform games. Despite belonging to the same genre, 2D and 3D games can differ greatly, prompting us to investigate whether any adaptations are necessary.

We selected five games for this study: Little Spy – v1.0.6 (*Game 1*), released in 2021; Diver Down – v1.2 (*Game 2*), released in 2018; Tiny Crate – v11.2022 (*Game 3*), released in 2021; Portal – v12.2014 (*Game 4*), released in 2007; and its sequel Portal 2 – v01.2022 (*Game 5*), released in 2011.

Game 1, Game 2, and Game 3 belong to the 2D platform genre and are available in the *itch.io* platform. These indie games are available as open source projects at GitHub and have been developed using the Godot open source engine (Godot, 2024).

In Game 1, players control a spy air-dropped into enemy territory to collect as many intelligence items as possible; then, they should return to the helicopter within a specified time frame (WINPRESS, 2024). Figure 3 depicts a level from Game 1 where the player (1) navigates platforms, collects items (2), avoids enemies (3), and reaches the helicopter (4).

Game 2 allows players to dive onto solids and use this ability to progress through stages, while avoiding lights (Escada Games, 2024). Figure 4 illustrates a level of Game 2, where the player (1) can dive onto the solid surface in front of him (2), needs to avoid the lights (3), and proceeds to the door (4).

In Game 3, players must lift and toss crates to create platforms, enabling them to reach higher ground (Harmony Honey, 2024). Figure 5 shows a level of Game 3 where the player (1) is lifting a crate (2), those crates can be moved around and help him to reach the end of the level (3).

Game 4 and Game 5, developed by the game developer and publisher Valve Corporation, fit into the 3D platform genre and incorporate elements from diverse genres including action, adventure, and puzzle-solving (Valve Corporation, 2023b; Valve Corporation, 2023a). In these games, the player is a test subject who must progress through levels known as test chambers, either by pressing buttons or avoiding contact with enemies. To accomplish this, the player is provided with a portal gun capable of shooting

Figure 3 – Level example of Game 1. - extracted from Duarte et al. (2024).

Figure 4 – Level example of Game 2. - extracted from Duarte et al. (2024).

two interconnected portals. Upon entering one portal, the player or any object emerges from the other portal.

There are graphical and gameplay differences between Game 4 and Game 5, primarily due to the time frame between their releases. Figure 6 depicts a level from Game 4,

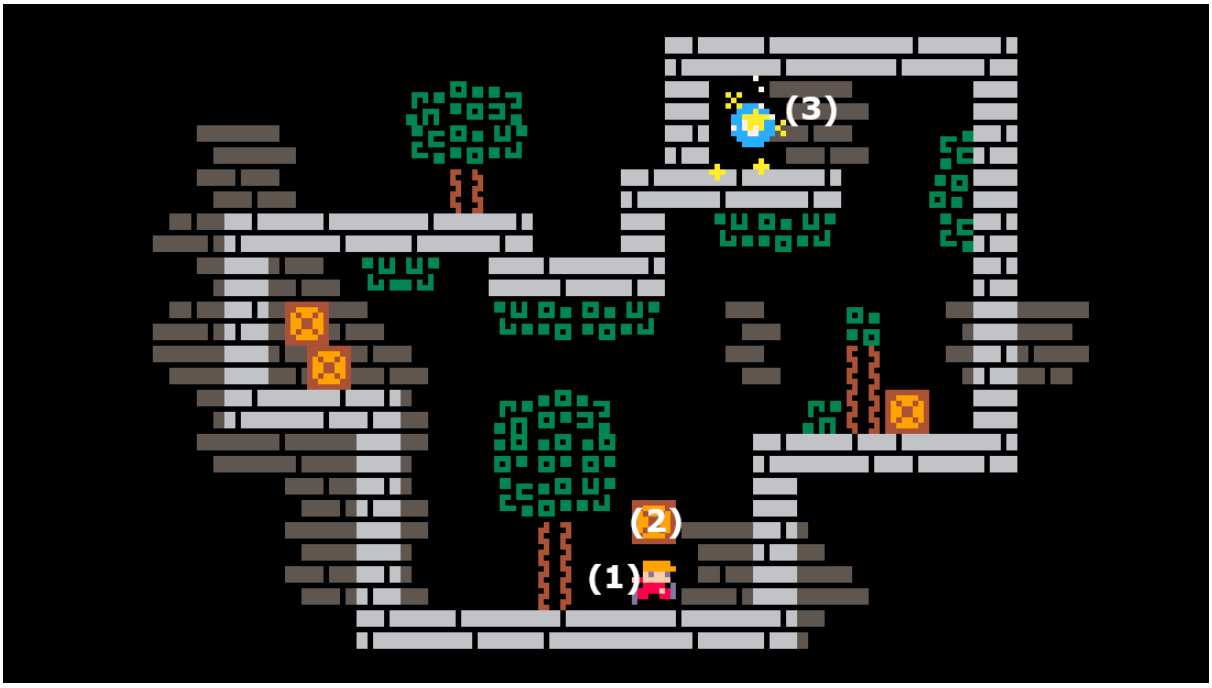

Figure 5 – Level example of Game 3. - extracted from Duarte et al. (2024).

highlighting some key elements: (1) the portal gun, (2) a portal created by the player, (3) the button that needs to be pressed to complete the level, (4) the object used by the player to press the button and move on, and (5) the door indicating the level exit.

Figure 7 shows a level from Game 5, where other elements are presented: (1) an enemy that shoots when the player is spotted, (2) a new liquid mechanic that causes different effects on surfaces and objects it touches, (3) one of the portals placed by the player, (4) a new mechanic of a "gravity tube" that moves the player or objects in the indicated direction, (5) a surface that allows the player to place a portal, and (6) a surface that does accept portals.

We selected these games for several reasons. In settling for Games 1, 2 and 3, we aimed to select 2D platform games crafted by indie developers or studios, specifically those available as open source projects. These games are characterized by their non-trivial nature and showcase a diverse set of mechanics. The level progression increases in difficulty as the player advances, contributing to their complexity. Our search for games meeting these criteria lead us to itch.io. We targeted indie games because our study about ET strategies would particularly help them. Finally, opting for open source games enhances the potential for future studies, as we have more control and access to the underlying code.

For Games 4 and 5, our aim was to include complex 3D games, well-known in their genre and in the game market, and developed by a reputable company. We considered complex games the non-trivial ones, where there are many game elements and mechanics,

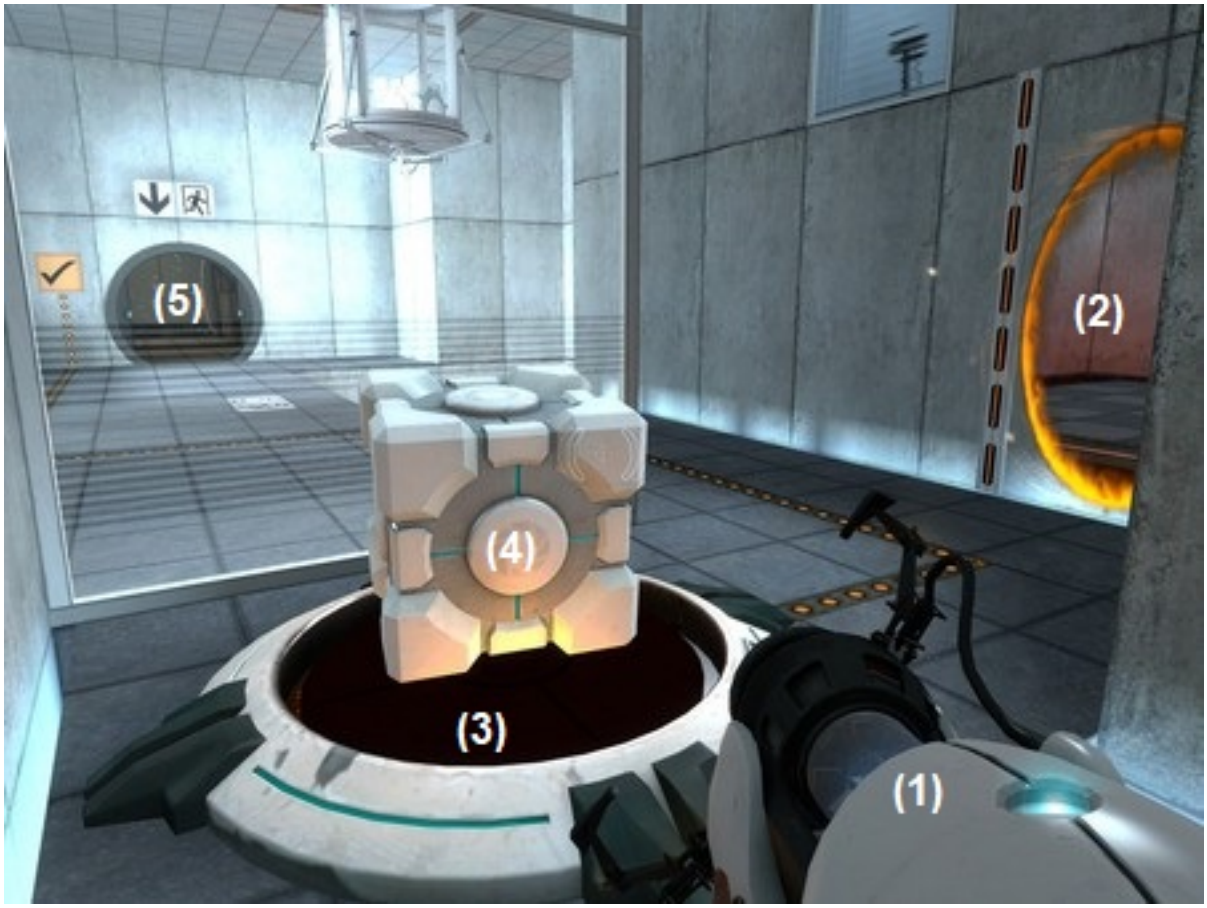

Figure 6 – Level example of Game 4. - extracted from Duarte et al. (2024).

lengthy phases, well-designed large maps, interactions with enemies or NPCs, soundtrack, and meticulously polished graphics. Testing this kind of games poses some challenges, and we anticipated they would yield valuable insights and lessons. They are also mature games, released some years ago, and have received various updates.

We envisaged that positive results for the selected games would motivate further studies about applying ET to games. Conversely, negative results would show major limitations of ET strategies, given the widespread availability and popularity of 2D/3D platform games among players.

3.2.2 Game Testing Procedure

This section outlines the procedure that we adopted to apply ET to the five selected games. Initially, we performed a session to the tester learn about the game, in which we called *Single Session Gameplay*, then we applied Whittaker's strategies (WHITTAKER, 2009) (see Section 2.5) in a specific order.

Single Session Gameplay: Initially, the game under test was played through in a single test session without time constraints or predefined strategies. To accomodate the

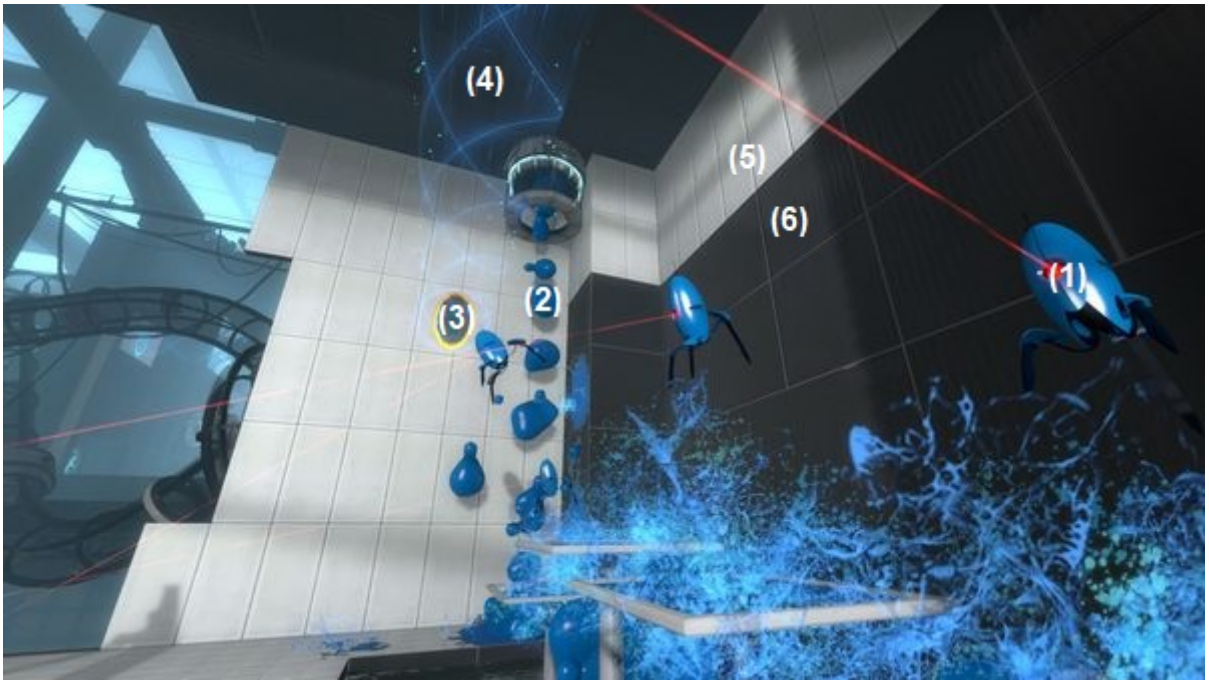

Figure 7 – Level example of Game 5. - extracted from Duarte et al. (2024).

potential for an extended duration, we allowed interruptions so that the game was saved and resumed after a break. This session allowed for the tester to grasp the mechanics of the game, easing the preparation for subsequent test sessions and providing unrestricted access to all game levels. Despite the absence of predetermined timeframes and strategies in this session, the tester actively sought out and documented bugs as they were encountered (see Section 3.3). This test session bears some resemblance to ad hoc tests performed by playtesters. This particular test session was termed *Single Session Gameplay*.

Application of ET strategies: Due to time constraints and potential adaptations required, we decided to employ a subset of ET strategies proposed by Whittaker (2009). In a prior empirical study on ET by Micallef, Porter e Borg (2016), seven of the Whittaker's strategies were selected and analyzed; we found these seven strategies suitable for game testing as well. Consequently, we used these strategies used in our study: Tour Bus Strategy, Exploratory Smoke Testing, Crime Spree Tour, Garbage Collectors Tour, Back Alley Tour, User Interface Exploration, and Bad Neighborhood Tour. We then examined the strategies and how they may be applied to the game, with a focus on assessing the quantity and duration of test sessions for each strategy. Additionally, we undertook adaptations in the strategies for the game's context.

In Tour Bus Strategy, the "main route" refers to completing the game level, while objects, map elements, or alternative options serve as "features". In Back Alley Tour, alternative routes for level completion were categorized as "rarely used elements", along with specific features like console or less displayed options to players. Strategies Ex-

ploratory Smoke Testing, Crime Spree Tour, and Garbage Collectors Tour were employed following their definitions without any adaptations.

For these five strategies, we defined an uninterrupted test session lasting x minutes, focusing on exploring a single game level. The timespan of x minutes was determined based on the experience in the Single Session Gameplay. Subsequently, we randomly selected six levels of the game and applied each strategy to these levels within the specified timespan. Due to the limited number and brevity of levels in Game 1, we considered all its levels as a single one. Table 1 summarizes the settings defined for the five games (i.e., Tour Bus, Exploratory Smoke, Back Alley, Crime Spree and Garbage Collectors.).

Table 1 – Settings for the five ET strategies - extracted from Duarte et al. (2024).

Game	**Session Time**	**Selected Levels**
Game 1	5min	All 6 levels in one session
Game 2	3 min	Level 1, Level 4, Level 8, Level 14, Level 18, Level 19
Game 3	7 min	Level 3, Level 12, Level 18, Level 23, Level 24, Level 27
Game 4	10 min	Chambers 4-7, Chambers 10-12, Chamber 13, Chamber 14, Chamber 17, Chamber 19
Game 5	10 min	Chapter 2, Chapter 3, Chapter 4, Chapter 6, Chapter 7, Chapter 8

For the User Interface Exploration strategy, we chose to focus on Graphical User Interface (GUI) elements akin to those found in conventional software, such as menus and input fields. This strategy, however, was not applied to Games 1 and 2 due to the absence of such GUI elements in these games. In the case of Game 3, we defined a 10-minute test session, whereas 15-minute test sessions were utilized for Games 4 and 5.

The last strategy applied was Bad Neighborhood Tour. Each bug detected so far in the game was examined, and the tester analyzed whether there is a "bad neighborhood" around it to be explored or not. To do so, the tester assessed the surrounding area and deliberated on potential issues related to the bug. For each bug exhibiting a potential "bad neighborhood", the tester conducted a test session, allowing sufficient time for exploration without a time limit.

Operational details: We opted for two authors (DUARTE et al., 2024) to participate as testers so that we could gauge deeper insights about the intrinsic steps of using ET in games. During the test sessions, the testers were instructed to avoid performing actions that could lead to previously-detected bugs; this guideline aimed to save session time to test other features of the game. The test sessions for Games 1-3 were performed in a computer AMD Ryzen 7 5800x 3.8GHz 16GB RAM AMD Radeon RX 6600 8GB with Windows 11, while Games 4 and 5 were executed in a computer AMD Ryzen 5 1600 3.2GHz 16GB RAM Nvidia GTX 1070 6GB with Windows 11.

It is worth noting that, prior to this study, neither of them had played any of the five games under test. Both authors have experience with software engineering and testing, and are well-versed in playing games. All test sessions were captured via screen recording using OBS Studio (OBS Project, 2022), encompassing video and audio of the testers. We used the recordings for further analyses, including test session evidence, checking potential issues, and confirming bugs.

When a bug was discovered during the tests, the tester logged it in a spreadsheet and correlated it with the session recording. The reports were further discussed among the authors to determine whether they should be classified as bugs or not. Each bug is assigned an ID, and in the case of recurring bugs, the ID from the first occurrence was also used in subsequent instances. All bugs were classified based on their severity, determined by the extent to which they bother or disrupt the player's experience. If the bug is minor and would be likely overlooked by the player, we classified it as *low* severity. If the bug pertains to rendering issues or malfunctioning of game elements, we classified it as *medium* severity. In instances where a bug significantly disrupts the player immersion, violating the game's rules, leading them to get stuck or hindering the progression in the game, we classified it as *high* severity.

3.3 Analysis of Results

This section analyzes the results obtained from the 152 test sessions (totaling more than 25 hours), in which the ET strategies were applied to the five games selected. Overall, 111 bugs were reported, being 105 unique bugs (some bugs were found twice). The main findings and the answer to the RQ are highlighted as framed rectangles throughout the section.

Table 2 gives an overview of the main results. For each ET strategy, it shows per game the number of test sessions, the time invested, and the number of bugs reported. Game 1 had 8 test sessions totalling around 50 minutes, and 10 bugs were reported. Game 2 had 33 sessions which took almost two hours, and 14 bugs were identified (12 bugs are unique). Game 3 had 34 sessions which took approximately four hours and a half, and 10 bugs were reported (8 bugs are unique). Game 4 had 40 test sessions totalling almost eight hours, and 42 bugs were reported (41 bugs are unique). Game 5 had 37 test sessions which took around ten hours and a half, and 35 bugs were reported (34 bugs are unique).

Initially, the Single Session Gameplay (row 3 in Table 2) helped to find 12 bugs (2 in Game 1, 2 in Game 2, 1 in Game 3, 2 in Game 4, and 5 in Game 5), over an approximate period of 7 hours and 35 minutes. As seen in Table 2, the duration of each session progressively increases from Game 1 (0:09:17) to Game 5 (4:36:47). Notably, the Game 5 session took more than twice the time invested in Game 4, due to Game 5's greater length and complexity. This part helped to comprehend the game mechanics and unlock access

Table 2 – Overview of the main results. - extracted from Duarte et al. (2024)

	Game 1			Game 2			Game 3			Game 4			Game 5		
Strategy	**Sessions**	**Time**	**Bugs**	**Sessions**	**Time**	**Bugs**	**Sessions**	**Time**	**Bugs**	**Sessions**	**Time**	**Bugs**	**Sessions**	**Time**	**Bugs**
Single Session Gameplay	1	0:09:17	2	1	0:16:42	2	1	0:42:01	1	1	1:50:33	2	1	4:36:47	5
Tour Bus Strategy	1	0:06:37	1	6	0:18:19	6	6	0:42:00	2	6	1:02:08	2	6	1:02:04	5
Exploratory Smoke Testing	1	0:05:10	2	6	0:18:45	2	6	0:42:57	3	6	1:01:24	10	6	1:00:38	4
Crime Spree Tour	1	0:04:57	1	6	0:18:28	0	6	0:42:50	0	6	1:02:09	7	6	1:02:59	2
Garbage Collectors Tour	1	0:04:45	0	6	0:18:08	0	6	0:42:45	1	6	0:59:29	5	6	1:01:07	9
Back Alley Tour	1	0:04:59	2	6	0:17:31	2	6	0:42:00	1	6	0:59:44	4	6	1:03:34	5
User Interface Exploration	-	-	-	-	-	-	1	0:10:06	2	1	0:15:02	8	1	0:14:08	4
Bad Neighborhood Tour	2	0:14:16	2	2	0:07:10	2	2	0:07:45	0	8	0:42:50	4	5	0:27:42	1
Total	8	0:50:01	10	33	1:55:03	14	34	4:32:24	10	40	7:53:19	42	37	10:28:59	35

to all game levels.

Concerning Tour Bus Strategy, Exploratory Smoke Testing, Crime Spree Tour, Garbage Collectors Tour, and Back Alley Tour (shown in rows 4-8 of Table 2), each one of them was applied, in variable time sessions, to six game levels, except for Game 1 in which one level was considered (see Table 1). In Game 1, each of these strategies was applied in one session with around five minutes. For Games 2-5, each strategy was performed in six test sessions, and the total elapsed time was from approximately 18 minutes for Game 2 to around one hour for Games 4 and 5, which can be seen in Table 2. The five strategies were applied directly or with minor adaptations (see subsection 3.2.2) and collectively helped to uncover bugs across all games: 6 bugs in Game 1, 10 bugs in Game 2, 7 bugs in Game 3, 28 bugs in Game 4, and 25 bugs in Game 5.

> *Finding 1:* Five out of the seven ET strategies could be applied to test the games with minor adaptations.

The User Interface Exploration strategy was applied in three games (row 9 of Table 2): a 10-minute session for Game 3 and two 15-minute sessions for Games 4 and 5. Specifically, these test sessions targeted UI elements that are close to traditional software. We found two bugs in Game 3, eight bugs in Game 4, and four bugs in Game 5.

> *Finding 2:* For games that have static menus, strategy User Interface Exploration was set up within only one test session and helped to reveal bugs related to traditional UI elements.

Finally, all bugs found so far (8 bugs in Game 1, 12 bugs in Game 2, 10 bugs in Game 3, 38 bugs in Game 4, and 34 bugs in Game 5) were evaluated for potential "bad neighborhood". For Game 1, two sessions with no time limit were conducted to explore the neighborhood of two bugs. This same configuration was replicated for Games 2 and 3. For Game 4, eight sessions without a time limit were carried out to investigate the surroundings of eight bugs, and for Game 5, five sessions (involving four bugs) were conducted. Applying the Bad Neighborhood Tour strategy (row 10 of Table 2) in Game 1 for 14 minutes supported the discovery of two new bugs; two bugs in 7 minutes for Game 2;

Table 3 – Bug severity by strategy. - extracted from Duarte et al. (2024)

	Game 1			Game 2			Game 3			Game 4			Game 5		
Strategy	**Low**	**Medium**	**High**	**Low**	**Medium**	**High**	**Low**	**Medium**	**High**	**Low**	**Medium**	**High**	**Low**	**Medium**	**High**
Single Session Gameplay	0	1	1	0	2	0	0	1	0	1	1	0	0	4	1
Tour Bus Strategy	1	0	0	0	4	2	1	0	1	0	0	2	0	4	1
Exploratory Smoke Testing	1	0	1	0	1	1	2	0	1	1	9	0	0	2	2
Crime Spree Tour	0	0	1	0	0	0	0	0	0	1	4	2	1	0	1
Garbage Collectors Tour	0	0	0	0	0	0	1	0	0	0	2	3	1	4	4
Back Alley Tour	0	0	2	2	0	0	0	0	1	1	2	1	0	3	2
User Interface Exploration	-	-	-	-	-	-	0	2	0	4	2	2	0	2	2
Bad Neighborhood Tour	1	0	1	0	0	2	0	0	0	0	4	0	0	0	1
Total	3	1	6	2	7	5	4	3	3	8	24	10	2	19	14

for a total of almost 43 minutes helped to reveal four new bugs in Game 4; and in around half an hour, one new bug in Game 5. The strategy did not help to uncover new bugs in Game 3.

> *Finding 3:* Strategy 'Bad Neighborhood Tour' was run at the end using the bugs previously uncovered; it required an assessment step to reason about the existence of a potential bad neighborhood. It helped to reveal new bugs in four out of five games and clarify doubts related to the previously found bugs.

Overall, Exploratory Smoke Testing resulted in the highest number of bugs for Games 1, 3 and 4, while Tour Bus Strategy helped to find the highest number of bugs for Game 2, and Garbage Collectors Tour for Game 5. In Game 1, Single Session Gameplay, Back Alley Tour, and Bad Neighborhood Tour are also among the top strategies. Notice that this analysis does not take into account the severity of bugs, or the time spent in the sessions (pace); we revisit the results from these perspectives next.

During the test sessions, bugs with low, medium and high severity were found. In Game 4, a bug with low severity is, e.g., when the player's death is not properly notified. An example of medium severity bug is the lack of a return-to-main-menu option in the game. An example of high severity bug is when interacting with the required object to complete a task can make it to be stuck, preventing the player from progressing further in the level.

Table 3 presents the number of bugs per severity detected, for each strategy. Proportionally, there are more bugs with medium severity: 1 bug for Game 1, 7 bugs for Game 2, 3 bugs for Game 3, 24 bugs for Game 4, and 19 bugs for Game 5. Next are the bugs with high severity (Game 1: 6, Game 2: 5, Game 3: 3, Game 4: 10, Game 5: 14), and the fewest bugs detected were the ones classified with low severity (Game 1: 3, Game 2: 2, Game 3: 4, Game 4: 8, Game 5: 2).

Concerning Game 1, Back Alley Tour found the highest number of high severity bugs, and Single Session Gameplay helped to uncover the only bug with medium severity. As for Game 2, Bad Neighborhood Tour and Tour Bus Strategy uncovered more high severity bugs, Tour Bus also revealed more medium severity bugs, and Back Alley Tour helped to discover the only two bugs with low severity. In Game 3, the three high severity bugs were

Table 4 – Ratio of the number of detected bugs per minute. - extracted from Duarte et al. (2024)

	Game 1				Game 2				Game 3				Game 4				Game 5			
Strategy	**Low**	**Medium**	**High**	**Total**	**Low**	**Medium**	**High**	**Total**	**Low**	**Medium**	**High**	**Total**	**Low**	**Medium**	**High**	**Total**	**Low**	**Medium**	**High**	**Total**
Single Session Gameplay	0.000	0.108	0.108	0.216	0.000	0.120	0.000	0.120	0.000	0.024	0.000	0.024	0.009	0.009	0.000	0.018	0.000	0.014	0.004	0.018
Tour Bus Strategy	0.151	0.000	0.000	0.151	0.000	0.219	0.109	0.328	0.024	0.000	0.024	0.048	0.000	0.000	0.032	0.032	0.000	0.064	0.016	0.080
Exploratory Smoke Testing	0.193	0.000	0.193	0.386	0.000	0.053	0.053	0.106	0.047	0.000	0.023	0.070	0.016	0.147	0.000	0.163	0.000	0.033	0.033	0.066
Crime Spree Tour	0.000	0.000	0.202	0.202	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.016	0.065	0.032	0.113	0.016	0.000	0.016	0.032
Garbage Collectors Tour	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.023	0.000	0.000	0.023	0.000	0.034	0.050	0.084	0.017	0.065	0.065	0.147
Back Alley Tour	0.000	0.000	0.400	0.400	0.114	0.000	0.000	0.114	0.000	0.000	0.024	0.024	0.017	0.033	0.017	0.067	0.000	0.047	0.032	0.079
User Interface Exploration	-	-	-	-	-	-	-	-	0.000	0.198	0.000	0.198	0.266	0.133	0.133	0.532	0.00	0.141	0.141	0.282
Bad Neighborhood Tour	0.070	0.000	0.070	0.140	0.000	0.000	0.280	0.280	0.000	0.000	0.000	0.000	0.000	0.093	0.000	0.093	0.000	0.000	0.036	0.036
Total				0.200				0.122				0.037				0.089				0.056

uncovered by three different strategies: Tour Bus Strategy, Exploratory Smoke Testing, and Back Alley Tour; two out of three medium severity bugs were identified using User Interface Exploration, and Exploratory Smoke Testing resulted in the highest number of low severity bugs. In Game 4, the strategies that resulted in the highest number of bugs of low, medium, and high severity were, User Interface Exploration, Exploratory Smoke Testing, and Garbage Collectors Tour, respectively. For Game 5, only Crime Spree Tour and Garbage Collectors Tour helped to uncover low severity bugs, while Single Session Gameplay, Tour Bus Strategy, and Garbage Collectors Tour resulted in the highest number of medium severity bugs; Garbage Collectors Tour helped to uncover more bugs with high severity.

Finding 4: All strategies helped to uncover high severity bugs in at least two games, being Tour Bus Strategy, Exploratory Smoke Testing, and Back Alley Tour capable of uncovering bugs in four of the five games. Regarding medium severity, Single Session Gameplay uncovered bugs in all of the games, while User Interface Exploration found bugs in the three games considered. For low severity, Exploratory Smoke Testing helped to identify bugs in three games.

Among the applied ET strategies, Tour Bus Strategy, Garbage Collectors Tour, and Back Alley Tour are the top strategy for more high severity bugs in two games.

From another perspective, we analyzed the performance of each strategy by relating the number of bugs and the time invested. Table 4 presents, per strategy and per game, the ratio of bugs found per minute; it also separates the ratio per bug severity. In general (column Total), Single Session Gameplay underperfomed in Games 3, 4 and 5, detecting around 0.02 bug per minute; Garbage Collector Tour had the worst results in Games 1 and 2. On the other hand, User Interface Exploration had the best performance when applicable, with 0.198 bugs/minute in Game 3, 0.532 in Game 4, and 0.282 in Game 5; Back Alley Tour was the best in Game 1 (0.400 bugs/minute), and Tour Bus Strategy in Game 2 (0.328).

Finding 5: When applicable, User Interface Exploration helped to find more bugs using less session time, even for medium and high severity bugs.

Notice that User Interface Exploration focuses on traditional UI elements and, as a consequence, targets a specific class of bugs. So, we now take a look at the other strategies that explore the gameplay. In Game 1, Exploratory Smoke Testing and Back Alley Tour had the two best results overall (column Total), but only Back Alley had by far the best performance in high severity bugs. In Game 2, Tour Bus Strategy had the best overall result, but Bad Neighborhood Tour had the best performance in high severity bugs. In Game 3, Exploratory Smoke Testing had the best overall best results, and no strategy stood out for high severity bugs. In Game 4, Exploratory Smoke Testing and Crime Spree Tour had the overall best results (column Total), but were outperformed by Garbage Collectors Tour in high severity bugs. In Game 5, Garbage Collectors Tour and Tour Bus Strategy had the overall best results (column Total); Garbage Collectors Tour also had the best performance in high severity bugs.

Finding 6: For strategies that explore the gameplay, Back Alley Tour helped to uncover more bugs per minute in Game 1, Tour Bus in Game 2, Exploratory Smoke Testing in Games 3 and 4, and Garbage Collectors Tour in Game 5. For high severity bugs, Tour Bus Strategy, Garbage Collectors Tour, and Back Alley Tour had the best performance in two games.

Although some ET strategies seem to shine in specific perspectives or games under test, our impressions are that all strategies contributed in some way for a more holistic playtesting. For instance, Single Session Gameplay had the longest test sessions and helped to find the a low number of bugs per minute. Nevertheless, it had a pivotal role to structure the testing procedure and support the acquisition of knowledge required to apply other strategies. Another example is User Interface Exploration that has the best ratio of bugs per minute but targets a specific kind of bugs. So, other strategies are important to explore the gameplay.

Answer to RQ: ET strategies can be applied to games with proper planning, knowledgeable methodology decisions, and minor adaptations. Overall, all strategies helped the tester to uncover several bugs in three indie and open source 2D platform games and two industrial 3D platform games.

3.4 Threats to Validity and Limitations

There exist several 2D and 3D platform games with different mechanics, making it impossible to encompass all these mechanics by testing only a few games. Therefore, we selected five games: three are open source indie 2D games, while two are closed source 3D games. The last two games are well known titles from the same company, and were selected due to their market recognition and being part of the same franchise.

To achieve a better understanding and insights about applying ET to games, two authors performed the role of tester, potentially introducing bias. This experience will help to set up the next steps of the research, which involves customization of ET strategies and other empirical studies.

Our results suggest that ET strategies can be successfully applied, helping testers to uncover several bugs. As our study dealt with a limited context, the results herein presented cannot be generalized.

3.5 Lessons Learned

Frequent motion sickness was noted during the tests of Game 4, necessitating breaks between sessions for the tester's recovery. Upon researching forums and communities, these symptoms seemed to be widespread among players of Games 4 and 5, but more prevalent in Game 4. Suggestions from these forums indicated that modifying certain settings could help alleviate the issue, such as disabling the "Motion Blur" and "Vertical Synchronization" options, and adjusting the "Field of View" setting. This experience underscores that the test sessions may support the identification of other issues related to the game, besides bugs. In a pre-release stage, early feedback on potential motion sickness could alert the developers to take some preventive measures.

While our study focused on already tested, and released games, the testing procedure is anticipated to provide the best return on investment in pre-release stages. On the other hand, if the proposed procedure helped to uncover several bugs in such a scenario, it is expected to deliver promising outcomes for games that have not undergone extensive testing.

Figure 8 shows a diagram that summarizes the steps we performed; its goal is to assist game testers that want to apply the testing procedure herein described. The diagram presents the following steps: (1) the tester performs a Single Session Gameplay, in which he completes the game with the objective of understanding the game mechanics and reason about appropriate ways to organize the test sessions; (2) any bugs encountered during this session should be recorded; (3) test sessions are defined considering the ET strategies' particularities; (4) the test sessions are carried out; (5) any bugs revealed are once again recorded; (6) the bugs found are assessed with respect to the applicability of

the Bad Neighborhood Strategy; (7) new test sessions are defined for applicable bugs; (8) the sessions are conducted; and (9) the bugs uncovered are once again recorded.

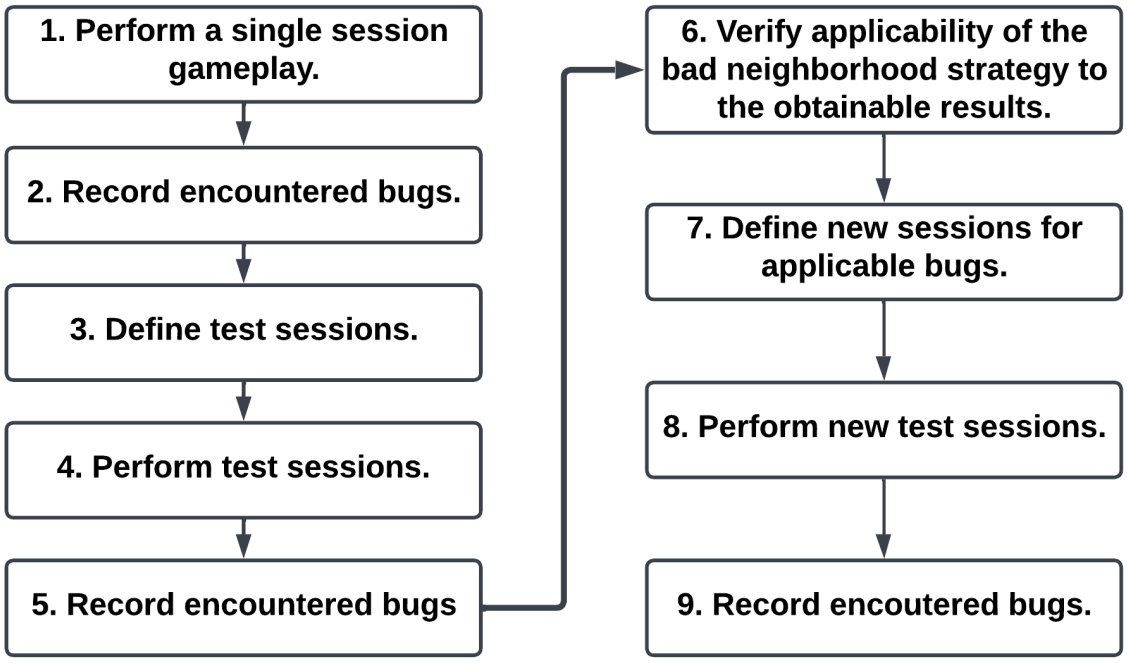

Figure 8 – Steps for conducting the ET testing procedure. - extracted from Duarte et al. (2024)

We suggest recording and narrating the test sessions to document when bugs are revealed and to clarify any cases that are unclear at first sight. For example, a reported bug may be an inconsistent game behavior that was intentionally introduced (an easter egg). Notice that our testing procedure considered the characteristics of the game, as well as the resources available. So, before commencing the tests, it is important to plan properly, make informed decisions, and perform some adaptations.

As for the tester profile, having knowledge of other games and bugs could help the tester in determining what to search for, expect and define during each test session. In this study, the two testers are well-versed in playing games and have software engineering and testing experience; we believe that this profile helped in time spent learning the game and also in test preparation and execution. Some games do not have an easy way to navigate between levels, which can require the tester to (re)play the game every time he changes strategy. In practice, the game developer should provide means for testers to have free access to levels, set up items, and even remove some game rule; e.g., remove or increase the time limit of a level so that the tester can explore it more freely.

The application of each ET strategy yielded valuable insights that we deemed important for game testers when reproducing it. These insights are described in the following

subsections.

Single Session Gameplay. In this part, the tester's primary focus is to obtain a general understanding about the game mechanics, controls, character abilities, scoring system, enemies, and other features. Tutorials and help menus may also aid in the process. To better allocate time and take advantage of other strategies, the exploration cannot divert from the main path as progressing to the game's end. While finding bugs is beneficial, the tester should avoid extensively exploring specific locations, as this task can be undertaken using other strategies. Finally, we recommend the tester takes breaks during extended gaming sessions.

User Interface Exploration. For the tested games, we opted for a single test session applying this strategy as the menus and traditional UIs are mostly static. Nevertheless, it may differ for other games, as the number of menus and other UI elements (like inputs, buttons, comboboxes, etc) varies. It is also important to consider if the game state has some influence on the UIs available to the players. Recommended features for this testing strategy also include graphical settings, keybinds, sounds, consoles, heads-up displays (HUDs), and status bars.

Garbage Collectors Tour. This strategy may draw some inspiration from speedrunning, a gaming practice where the player aims to achieve a specific goal in the game as quickly as possible[2]. The goals extend beyond merely beating a level or completing the entire game. For this strategy, the tester should focus on her objective disregarding other parts of the game, in a rapid way. By reasoning about means of being faster, the exploration may guide the discovery of unintended behavior or bugs.

Tour Bus Strategy. This strategy shares some similarities with completionist[3], a gaming practice where the player intends to complete the entire game while exploring it comprehensively by, for example, obtaining all achievements, or doing all quests. Such a perspective may help to guide the tour to stop not only in the main places of interest, but also the secondary ones.

Back Alley Tour. In this strategy, the tester should focus on less frequently utilized features from the player's standpoint. Some examples are hidden elements, developer options, and consoles. The completionist mindset may also offer some insights, as the pursuit of game completion could reveal concealed behavior to be tested.

Crime Spree Tour. This strategy aims to explore negative scenarios around specific game features. The tester needs to reason about potentially error-prone features and target them to break the normal behavior of the game. For instance, the tester could try to apply mechanics out of their established context or bounds, or push the limits of certain features.

Exploratory Smoke Testing. This strategy should not follow any guideline besides

[2] https://www.speedrun.com/support/learn/what-is-speedrunning
[3] https://ktswblog.net/2022/10/21/being-a-completionist-gamer

randomly explore features of the game. In particular, we used it to answer "what if" questions about game features; those questions came mostly from observations in previous test sessions.

Bad Neighborhood Tour. The neighborhood of a bug may be the home of other bugs, this is an opportunity for the game testers. For instance, if a bug regarding a specific window size of a menu option is found, other menu options could be explored to verify whether this bug could affect the game in any other way. The concept of neighborhood is also extended: elements of a bug can manifest or "teleport" to different levels or contexts within the game. The tester should explore if a faulty behavior can be partially replicated in different states of the game. It is important, however, to evaluate the bugs beforehand and assess if there is a logical neighborhood to explore. Occasionally, this exploration has already been conducted in previous test sessions.

We have some insights about the ordering of ET strategies in games. By maintaining the Single Session Gameplay as the initial step, the tester could consider to apply User Interface Exploration either before or after implementing the other five strategies. Then, Bad Neighborhood Tour is employed after all other sessions have been completed.

Concerning the five strategies, we suggest beginning with the Tour Bus, where the tester briefly explores various features without delving deeply into any one. Subsequently, Crime Spree Tour would be adopted to explore specific features more deeply. Following this, Garbage Collectors Tour is employed as the tester would be well-versed enough in the game to achieve objectives as quickly as possible. At this point, the tester is capable of recognizing the least used features, making it the proper time to apply Back Alley Tour. Lastly, considering that the tester is likely well-acquainted with the game and tried different strategies, testing randomly without following any pattern could yield good results (i.e., Exploratory Smoke Testing).

3.6 Concluding Remarks

Testing in games is crucial to ensure game quality and ET, a well-known approach for traditional software, is a promising approach to improving gameplay testing. However, there is a lack of studies or guides addressing how ET can be applied to games. Therefore, this study contributes to shed some light on this topic by conducting a study that applied seven ET strategies in three indie 2D platform games and two mature and well-known 3D platform games. We reported our approach to setup the test sessions and to adapt the strategies for the games' domain. At the end, several bugs were revealed with different levels of severity. Overall, the results were positive, providing valuable insights for game testers and motivating further investigations.

In our subsequent study, we considered the lessons learned on this study and expanded this adaptation by proposing a framework that revisits of Whittaker's (2009) strategies

into the game context, rather than using only seven of them. This study is presented in the next chapter.

Chapter 4

A framework for exploratory testing in video games

4.1 Overview

This chapter presents xPloiT, a framework developed to conduct ET in games. This framework was developed by merging our lessons learned in our previous study (DUARTE et al., 2024), presented in Chapter 3, and strategies proposed by Whittaker (2009). The focus on this study were indie studios and developers, in order to enhance their test effectiveness.

Section 4.2 introduces the framework and outlines the steps involved. Section 4.3 details the study's setup to assess the proposed framework. Section 4.4 presents an analysis of the results. Section 4.5 discusses the threats to validity and limitations.

This chapter summarizes the main results of the paper "Towards a framework for exploratory testing in video games", Duarte, Y.; Politowski, C.; Endo, A. T., accepted for publication in the 9th International ICSE Workshop on Games and Software Engineering, 2025.

4.2 xPloiT Framework

We developed a framework for ET of games by merging and basing our prior research on traditional software testing and game testing (see Chapter 3). The framework is composed of nine strategies. Each strategy contains subgoals with questions to help guide the tester. All subgoals and questions are listed in Appendix A. The strategies were

designed to be applied individually in a given test session. During the test sessions, the tester registers any relevant gameplay details.

The strategies are performed in sequential order. In summary: (S1) the tester completes the game available segment to understand the mechanics; the tester defines the number of sessions and time constraints for the subsequent strategies (S2) to (S7); (S8) the tester evaluates the bugs found based on the applicability of their neighbouring features and possible consequences; Finally, (S9) unexplored notes or ideas from being executed. The strategies are described in detail in Figure 9 and in the following sections.

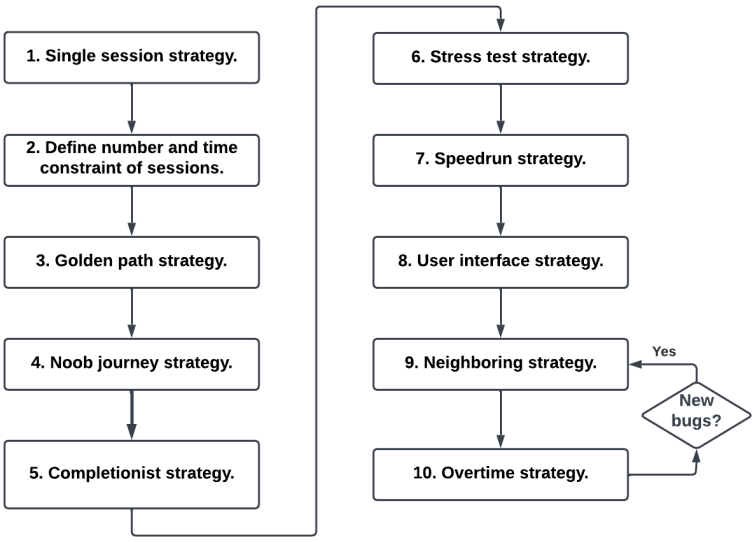

Figure 9 – xPloiT Strategies. - extracted from Duarte, Politowski and Endo (2025)

Strategy #1: Newbie Journey

We start the xPloiT framework with the Newbie Journey Strategy to familiarize testers with the game. In this strategy, the tester plays through the available game segment for the first time to learn about its mechanics, controls, balance, design, and objectives. The tester should avoid spending excessive time exploring during this step. After completing the initial playthrough, the tester will have enough experience to decide on the number of sessions and time constraints for later strategies.

Strategy #2: Golden Path

We established the Golden Path Strategy as the second step of xPloiT to help testers play the game as intended by the designers. In this strategy, testers play the game as intended by the designers and keep their actions straightforward. They follow the main or default path in the game test segment, avoiding deviations. Testers may briefly stop to investigate features of particular interest.

Development of the Strategy

We developed this strategy by merging the Tour Bus strategy (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016), the Coach Potato tour (WHITTAKER, 2009), and the game design concept known as the Golden Path (DEVELOPER, 2019), where players make decisions that align with the designers' intended experience.

Example

In Figure 10, the main character (1) needs to reach the objective (4). Using the *Golden Path Strategy*, the tester focuses on proceeding directly to the objective without spending much time on items (3) or enemies (2) along the way (blue arrow). However, the tester may briefly stop to investigate features of particular interest (cyan arrow).

Figure 10 – Level example of *Little Spy*. - extracted from Duarte, Politowski and Endo (2025)

Strategy #3: Noob Journey

We established the *Noob Journey*[1] *Strategy* as the third step of xPloiT to help testers explore unused or less common paths and features. In this strategy, testers ignore the game's tips and requirements and explore alternative paths to reach the objectives.

[1] Among the gaming community, "noob" define a player that purposely does not want to learn a game, while "newbie" is a beginner or inexperient one(CODERRACH, 2010; TRISP, 2013; DARKSCARLETX, 2015).

Development of the Strategy

We developed this strategy by merging the Antisocial tour and its subtours (WHITTAKER, 2009), the Back Alley tour (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016), and the Blockaded Taxicab tour (WHITTAKER, 2009).

Example

In Figure 11, the tester focuses on reaching the objective by avoiding the expected path (blue arrow) and attempts to reach it through alternative methods without meeting the usual requirements (cyan arrow). Game tips and guides are disregarded.

Figure 11 – Level example of *Little Spy.* - extracted from Duarte, Politowski and Endo (2025)

Strategy #4: Completionist

We established the *Completionist Strategy* as the fourth step of xPloiT to help testers focus on collecting game outputs such as achievements, items, rewards, scores, and upgrades. In this strategy, the tester tries to collect every possible output from the game, following the longest path to do so. Time constraints should account for the extra time needed compared to other strategies; therefore, additional time may be allocated.

Development of the Strategy

We developed this strategy by merging the Collectors Tour (WHITTAKER, 2009), the Lonely Businessman Tour (WHITTAKER, 2009), and the Completionist game concept (CRUZ; HANUS; FOX, 2017), where players complete every available task and collect all badges the game offers.

Example

In Figure 12, the tester explores the longest path (blue arrow) to reach the objective, defeating every enemy (red circles) and collecting all items along the way (pink circles). Additionally, the tester considers exploring the map (cyan arrow) to search for possible secrets[2].

Figure 12 – Level example of *Little Spy*. - extracted from Duarte, Politowski and Endo (2025)

Strategy #5: Stress Test

We established the *Stress Test Strategy* as the fifth step of xPloiT to help testers explore unexpected inputs and actions in the game test segment. In this strategy, testers explore unexpected, invalid, or illegal inputs and actions, repeating them to identify potential issues.

[2] A secret is a piece of content within the game that can be missed by the player (BURGUN, 2023).

Development of the Strategy

We developed this strategy by merging the Antisocial Tour and its subtours (WHITTAKER, 2009) with the OCD Tour (WHITTAKER, 2009).

Example

As shown in Figure 13, the game specifies objectives and controls to the tester. The tester should attempt to use other buttons, even if not obvious, to see if they trigger any interactions. If game progression is achieved by reaching the helicopter, the tester should also consider moving in the opposite direction to explore alternative outcomes.

Figure 13 – Start menu of *Little Spy*. - extracted from Duarte, Politowski and Endo (2025)

Strategy #6: Speedrun

We established the *Speedrun Strategy* as the sixth step of xPloiT to help testers try different paths and discover possible exploits. In this strategy, testers learn about the paths and objectives of the game test segment, set new objectives, and attempt to reach them using the shortest available path. They aim to achieve goals quickly by ignoring requirements or attempting alternate paths. Time constraints should reflect that testers will spend less time on path discovery compared to other strategies, allowing for potentially shorter sessions.

Development of the Strategy

We developed this strategy by merging the Speedrun concept (SPEEDRUN.COM, 2014), where players try to complete the game or part of it as fast as possible, with the Garbage Collectors Tour (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016), Taxicab Tour (WHITTAKER, 2009), Landmark Tour (WHITTAKER, 2009), Coach Potato Tour (WHITTAKER, 2009), and Blockaded Taxicab Tour (WHITTAKER, 2009).

Example

In Figure 14, the tester considers whether there is a faster path to reach the objective (blue arrow). They might explore climbing platforms, bypassing via a shortcut (cyan arrow), or exploiting potential bugs to quickly reach the objective.

Figure 14 – Level example of *Little Spy*. - extracted from Duarte, Politowski and Endo (2025)

Additionally, Figure 15 shows a faster path to the objective (blue arrow), where the tester could benefit from the character's spawn to avoid reaching the lower platform. The cyan arrow represents a possible even faster path, eliminating the need for an additional jump. This could led to the discovery of new bugs.

Strategy #7: User Interface

The *User Interface Strategy* aims to help testers cover the menus and interface of the game test segment. In this strategy, testers focus on the user interface and localization

Figure 15 – Level example of *Little Spy*. - extracted from Duarte, Politowski and Endo (2025)

aspects of the game. Time constraints and the number of sessions should consider the complexity and quantity of menus and interfaces within the game test segment.

Development of the Strategy

We developed this strategy by merging User Interface Exploration (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016), Supermodel Tour (WHITTAKER, 2009), and Multicultural Tour (WHITTAKER, 2009).

Example

In our previous work (DUARTE et al., 2024), a bug was discovered during a user interface session. The tester found that when playing the game in windowed mode, enlarging the "Achievements" pop-up could block interactions with the menu.

Strategy #8: Neighboring

The *Neighboring Strategy* aims to help testers explore found bugs. In this strategy, testers explore areas surrounding bugs, investigate potential consequences, and examine nearby features of commonly used elements with potential vulnerabilities. Time constraints and the number of sessions depend on the presence of applicable bugs and the time required to explore their surroundings.

Development of the Strategy

We developed this strategy by merging the Bad Neighborhood Tour (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016) with the Supporting Actor Tour (WHITTAKER, 2009).

Example

In one of the games tested in our previous work (DUARTE et al., 2024), we discovered a bug. Saving the game during a developer's commentary caused the commentary to restart and only continue for the remaining time from the point of saving. This led us to ask, "Does the same occur with NPC dialogues?" Further investigation revealed two additional bugs: "Subtitles disappear when the game is saved and reloaded during a character's dialogue," and "Voice sometimes echoes when the game is saved and reloaded during a character's dialogue."

Strategy #9: Overtime

We established the *Overtime Strategy* to help testers address their questions and notes from previous steps. This is an iterative strategy with the last one (Strategy #8) because newly found bugs may be applicable to the *Neighboring Strategy*. In this strategy, testers address all the questions or possible test cases they have noted and perform random tests whenever they think of something new to explore. Time constraints and the number of sessions depend on the tester's questions, notes, and ideas gathered in previous steps.

Development of the Strategy

We developed this strategy by merging Exploratory Smoke Testing (WHITTAKER, 2009; MICALLEF; PORTER; BORG, 2016) with the Intellectual Tour (WHITTAKER, 2009).

Example

In the game presented in Figures 10 to 15, the number displayed in the top-right corner represents a timer. When the timer reaches "000", the player loses. A potential question for the tester is: "What happens if I enter the helicopter just as the timer reaches zero?" The answers are: case 1, the tester reaches the helicopter before the timer reaches "000" and proceeds to the next stage; and case 2, the timer reaches "000" before the tester reaches the helicopter, resulting in game over. Such questions and other ideas should be tested in this strategy.

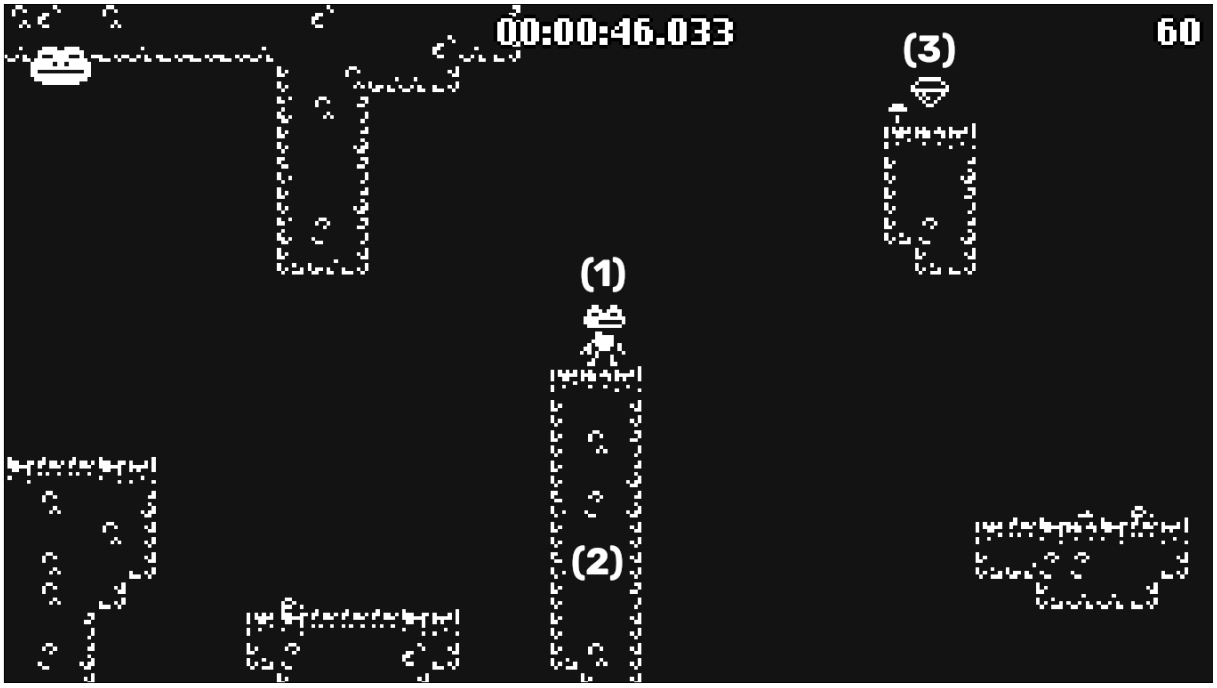

Figure 16 – A sample level from Leap Hero. - extracted from Duarte, Politowski and Endo (2025)

4.3 Study Setup

To provide the first assessment of the proposed framework, we conducted an evaluation using a platform game developed by an indie studio. The main author served as the tester; this was the tester's first time playing the game and he had no prior knowledge about bugs. Finally, the bug report was validated with the game developers.

4.3.1 Game Under Test

For this study, we selected the game *Leap Hero* (JAPASSOU; MACHADO, 2023), with a demo available in the *itch.io*[3] platform, developed using GameMaker engine (LTD., 2013). In this game, players control a knight tasked by the king to rescue the princess; the knight is also a frog who has the skill of jumping between walls and grabbing them with his tongue. Figure 16 shows a level from the game where the player (1) navigates through platforms (2), collects diamonds (3), avoids enemies and proceeds to the next map. The game had already undergone informal testing through its demo available on *itch.io* and during game events the developers attended.

[3] <https://itch.io>

4.3.2 Testing Procedure

After completing the game once in the *Newbie Journey Strategy*, the tester estimated the time needed for an average strategy session. The game consists of two parts: the first where the player controls a knight; and the second where the player controls a frog. We decided to have test sessions for each part, with different periods set due to their varying complexity and time spent. We defined the number of sessions and time constraints for six strategies: *Golden Path, Noob Journey, Completionist, Stress Test, Speedrun. User Interface* did not have the same setup as other strategies since it depends on menus, interfaces and their complexities; the *Neighboring* strategy depends on the number of applicable bugs discovered; and the *Overtime* strategy considers the notes and questions the tester gathers by the end of the tests.

After completing the previously defined test sessions, the tester applied the *User Interface* strategy, then evaluated which bugs were applicable under the *Neighboring* strategy and conducted a session for each applicable bug. Finally, the tester implemented *Overtime* strategy, where he addressed questions and notes taken in earlier steps. The tester did not consider any bug found during *Overtime* strategy applicable for further investigation under the *Neighboring* strategy.

Operational Details

We conducted test sessions on a laptop with an Intel I7-11800H 4.60GHz CPU, 16GB RAM, and Nvidia RTX 3060 Mobile GPU, running Windows 11. We recorded all test sessions using OBS Studio (OBS Project, 2022), capturing video and audio of the tester, and used the recordings for further analysis, including reviewing evidence of test sessions and confirming bugs with the developers.

The tester avoided triggering previously-detected bugs, to use session time more efficiently. The tester reported in a spreadsheet whenever found a bug, assigning an ID, description, timestamp and current strategy being used. He logged recurring bugs under the ID from their first occurrence.

After completing all the tests, the tester discussed all bug reports with the developer in a synchronous meeting. The developer classified each report as a false positive (e.g., it is an expected feature), an improvement, or a bug. For bugs, we also registered whether it was known or not by the developer. Afterwards, the researchers reviewed the classification, and any inconsistencies were resolved collaboratively.

XPloiT guidelines, including subgoals and questions to apply the strategies, raw data collected, recordings and bug reports are available in the following page: <https://zenodo.org/records/14743857>

4.4 Analysis of Results

Table 5 presents, per strategy, the number of test sessions, the time invested, the number of reports, false positives (FPs), improvements, bugs known by the developers, unknown bugs, total of bugs (known and unknown), and the ratio of the bugs found (i.e., how many bugs per hour each strategy detected). The results obtained from the 28 test sessions, totalling almost 5 hours using nine strategies, consist of: 87 bug reports, 80 of which were confirmed as bugs, with 40 known by the developers and 40 new.

We noticed that some of the 80 confirmed bugs had the same root cause or repeated manifestation. By removing this redundancy, we obtained 60 bugs uniquely revealed. A minor portion of reports (7 reports - 8%) was not classified as bugs, but six were FPs and one improvement. Particularly, the FPs occurred mostly in the first session. This is somewhat expected due to the lack of familiarity with the game, and **confusing bugs and features**. Reviewing the initial bugs (found in *Single Session*) with the developers could help avoid such misunderstandings.

Table 5 – Overview of main results per strategy. - extracted from Duarte, Politowski e Endo (2025)

Strategies	# Sessions	Time	# Reports	# False Positives	# Improvements	Known bugs	Unknown bugs	Total bugs	Ratio
Single Session	1	00:57:44	19	4	1	9	5	14	14.55
Golden Path	2	00:37:27	11	1	-	8	2	10	16.02
Noob Journey	2	00:22:57	13	-	-	8	5	13	33.99
Completionist	2	00:46:33	9	1	-	5	3	8	10.31
Stress Test	2	00:29:07	12	-	-	3	9	12	24.73
Speedrun	2	00:26:47	3	-	-	1	2	3	6.72
User Interface	1	00:10:56	7	-	-	3	4	7	38.41
Neighboring	14	00:54:54	11	-	-	3	8	11	12.02
Overtime	2	00:07:56	2	-	-	0	2	2	15.13
Total	28	04:54:21	87	6	1	40	40	80	16.31

Among the strategies, *Newbie Journey* had the longest time (almost 1 hour) and yielded the highest number of bugs (total: 14). Still, most of the bugs (9 out of 14) were known by the developers. This scenario is repeated in the following strategies (*Golden Path*, *Noob Journey*, and *Completionist*). These changes start in *Stress Test* to *Overtime* where most bugs uncovered are mainly unknown by the developers. For instance, the *Stress Test* strategy uncovered the most unknown bugs (9 out of 12), followed by *Neighboring Strategy* (8 out of 11).

We identified two potential explanations for the number of bugs uncovered among these strategies. The first is based on the experience and invested time factors: the bugs easier to detect (i.e., the bugs known by the devs and revealed by simple ad-hoc testing) are uncovered first and, as more time is invested and more experienced is the tester, more unknown bugs are found. The second explanation relies on how the strategies are designed: while the first four strategies have their own goals and testing perspectives, they in general aim to explore the primary features of the game under test. This makes them more subjected to discovering known bugs. On the other hand, the last five intend to test the game by adopting unconventional perspectives, helping to detect more unknown

bugs. While we cannot rule out the first hypothesis without further experiments, the observation that all strategies detected at least 2 unknown bugs gives some evidence that all strategies contribute to the exploratory tests and complement each other within the framework.

Speedrun and *Overtime* detected fewest bugs, with 3 and 2 respectively. As for *Speedrun*, we believe this result comes from the strategy targeting bugs that are likely to be restricted in scope and quantity. Concerning *Overtime*, as the strategy to be performed at the end to explore untried ideas, some saturation is expected with only a few bugs to be detected.

We now look at the number of bugs detected concerning the time invested (column Ratio in Table 5); in general, a higher ratio may indicate a better cost-effectiveness relation. *Speedrun* had the lowest ratio (6.72 bugs/hour), which is somewhat explained by its low number of bugs detected. The best ratios are for *User Interface* (38.41 bugs/hour), *Noob Journey* (33.99 bugs/hour), and *Stress Test* (24.73 bugs/hour).

A possible explanation for the performance of *Noob Journey* and *Stress Test* is that they oppose *Golden Path*, where the tester follows the path expected by the developer. By taking unexpected paths, exploring random inputs and covering features that may have received less attention from the developers, the tester is more likely to uncover additional bugs; as for *User Interface*, we believe that by focusing on traditional UI elements, it targets a specific class of bugs that are not addresed by other strategies. The other strategies are closer to the average for the full framework application (16.31 bugs/hour).

> *Finding 1:* The XPloiT application helped to uncover 80 bugs in an indie game under development. Half of these bugs were unknown by the developers. All strategies revealed bugs, ranging from 2 to 14 bugs detected per strategy. Collectively, the exploratory tests detected around 16 bugs per hour.

We herein describe bugs that seem to be more connected with the perspective approached by the strategies.

In *Newbie Journey*, the tester encountered bug G1-B-05[4], where coins collected remain in the inventory even after the character's death. Since this bug involves a common feature available from the start of gameplay, it is expected to be found in a general, non-focused strategy. In *Golden Path*, as the game includes a mechanic where players must jump on specific enemies to defeat them, the tester discovered bug G1-B-20. Here, the character takes damage if the player attempts to jump on two enemies consecutively during an encounter, contrary to the tester's expectation that consecutive jumps would be what the designer wants the players to do. In *Noob Journey*, while ignoring tips and exploring as

[4] We use the bug ID here to track it to the supplementary material.

he wanted, the tester uncovered bug G1-B-28, where climbing a specific wall allowed the character to enter an out-of-bounds area of the map.

In *Completionist*, while trying to explore all areas, the tester encountered bug G1-B-34, where a mechanic involving the character's tongue and a jump failed to function stably. In *Stress Test*, as the tester tried random inputs, he uncovered bug G1-B-38, where pressing "Esc" to exit the menu unpauses the game before the menu disappears from the screen. In *Speedrun*, the tester observed bug G1-B-49, an animation glitch that appeared during a character transformation, discovered while attempting to progress quickly through the game. In *User Interface*, the tester found bug G1-B-52, where changing the game's resolution setting did not function correctly as expected during menu exploration.

For *Neighboring*, the tester discovered bug G1-B-55, where restarting the level while in a dialogue rendered the character unresponsive. The tester discovered this while exploring the surroundings of bug G1-B-06, where jumping in a specific position causes the character to keep falling. Finally, in *Overtime*, bug G1-B-65 emerged, where high-frequency monitors caused the game's responsiveness issues. During test sessions of *Speedrun*, the tester took note of this issue after observing instability in the character's tongue and jump mechanics.

Finding 2: Our results gave initial evidence that the testing perspective fostered by each strategy may help the tester to target specific bugs. As such, all testing strategies can be viewed as complementary, providing better overall bug detection capabilities within the XPloiT framework.

4.5 Threats to Validity and Limitations

The study herein presented has some limitations. For instance, a limitation is the number and background of the tester; as one of the authors acted as the tester, this could introduce bias in the evaluation of the framework's effectiveness. The evaluation was conducted with a platform game, so the results do not generalize for other genres, such as fighting, puzzle, and shooter.

It is not possible to precisely define the developers' efforts on tests conducted before this study. Additionally, individual characteristics of the testers could influence the results obtained.

4.6 Concluding Remarks

This chapter contributes to Game Software Engineering by presenting XPloiT, a framework designed for game testers to perform exploratory testing strategies in games. The

framework contains nine strategies that support the structuring of test sessions, in which the tester approaches game under test from different perspectives. To assess it, we conducted a evaluation of applicability with a 2D platform game developed by an indie studio. Overall, we found the results promising, once we revealed 80 total bugs, being half unknown to the developers and 60 of them unique. We also found an initial evidence that the strategies may guide the detection of specific types of bugs, and the studio gave positive feedback.

Chapter 5

Conclusion

Testing is essential to guarantee the quality of a game. This thesis has contributed with two studies on adoption and adaptation of software practices, more specifically Exploratory Testing strategies, in game testing context. The results of these were positive, achieving the main objective of this project, proposed in Chapter 1, i.e., to investigate ET for games in order to enhance bug detection and improve product quality.

The contributions that support this achievement and results are revisited in Section 5.1. Section 5.2 explores the possible future directions of this thesis.

5.1 Revisiting Contributions

A study on exploratory testing for video games: We conducted a study testing five platform games (three 2D and two 3D) using seven strategies adapted for the gaming context. These strategies, proposed by Whittaker (2009) and Micallef, Porter and Borg (2016), include User Interface Exploration, Garbage Collector's Tour, Tour Bus Strategy, Back Alley Tour, Crime Spree Tour, Exploratory Smoke Testing, and Bad Neighborhood Tour. Our results revealed 111 bugs, identified across 152 test sessions. The insights obtained from these findings, along with reviewers feedback, contributed to the development of our framework.

Framework xPloiT: We conducted a development of a framework specifically designed for conducting ET in games. Then, we performed an evaluation of applicability of the framework in which a 2D platform game, still in development, was tested. Our results revealed 80 bugs, 60 of which were unique, identified across 28 test sessions.

5.2 Future Directions

We believe that by adjusting the session duration and quantity, it is possible to adapt the framework to different game genres, considering their specificities (e.g., procedural games and how their levels should be considered during sessions). The strategies could be explored for their effectiveness in identifying specific bug types. Further studies may provide more evidence on ET in games. It is of interest to evaluate how these ET strategies benefit testers with different backgrounds and levels of experience.

Future studies should involve a broader pool of testers to provide additional evidence, refine the framework, and identify key challenges in its application. Additionally, tool support could further aid the implementation of xPloiT by helping testers define the testing process, select strategies, and manage test sessions.

Building on these findings, we plan to explore playtesting within indie game studios. Since these studios often have limited resources for testing, they rely heavily on manual playtesting and may need to recruit inexperienced testers or volunteers. Our research could significantly benefit professionals in this context. Offering training and guidelines on testing could improve the effectiveness of manual game testing and contribute to better outcomes. Also, another direction to follow is to use the knowledge obtained to aid the implementation of automated testing using autonomous agents.

References

AL-AZAWI, R.; AYESH, A.; OBAIDY, M. A. Generic evaluation framework for games development methodology. In: **2013 Third International Conference on Communications and Information Technology (ICCIT)**. Beirut, Lebanon: IEEE, 2013. p. 55–60. ISBN 978-1-4673-5307-6 978-1-4673-5306-9. Disponível em: <http://ieeexplore.ieee.org/document/6579522/>.

ALEEM, S.; CAPRETZ, L. F.; AHMED, F. Critical Success Factors to Improve the Game Development Process from a Developer's Perspective. **Journal of Computer Science and Technology**, v. 31, n. 5, p. 925–950, set. 2016. ISSN 1000-9000, 1860-4749. Disponível em: <http://link.springer.com/10.1007/s11390-016-1673-z>.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing:**. 2. ed. Cambridge University Press, 2016. ISBN 978-1-107-17201-2 978-1-316-77127-3. Disponível em: <https://www.cambridge.org/highereducation/books/introduction-to-software-testing/95E57CCADEA697EC8594F03729F47311#contents>.

AMPATZOGLOU, A.; STAMELOS, I. Software engineering research for computer games: A systematic review. **Information and Software Technology**, v. 52, n. 9, p. 888–901, set. 2010. ISSN 09505849. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0950584910000820>.

ARIYUREK, S.; SURER, E.; BETIN-CAN, A. Playtesting: What is Beyond Personas. **IEEE Transactions on Games**, v. 15, n. 3, p. 348–359, set. 2023. ISSN 2475-1502, 2475-1510. Disponível em: <https://ieeexplore.ieee.org/document/9754697/>.

AZIZI, E.; ZAMAN, L. AstroBug: Automatic Game Bug Detection Using Deep Learning. **IEEE Transactions on Games**, p. 1–14, 2024. ISSN 2475-1502, 2475-1510. Disponível em: <https://ieeexplore.ieee.org/document/10533870/>.

BACH, J. Exploratory Testing Explained. 2003.

BANKHURST, A. **Cyberpunk 2077 teve o maior lançamento digital da história, mostra levantamento**. 2021. Disponível em: <https://br.ign.com/cyberpunk-2077/86692/news/cyberpunk-2077-teve-o-maior-lancamento-digital-da-historia>. Acesso em: 06 de outubro de 2021.

BLIZZARD ENTERTAINMENT, I. **Battle.net**. 2024. Disponível em: <https://us.shop.battle.net/en-us>.

BORG, M. et al. Video Game Development in a Rush: A Survey of the Global Game Jam Participants. **IEEE Transactions on Games**, v. 12, n. 3, p. 246–259, set. 2020. ISSN 2475-1502, 2475-1510. Disponível em: <https://ieeexplore.ieee.org/document/8686138/>.

BURGUN, K. **Secrets in Videogames**. 2023. <https://keithburgun.net/secrets-in-videogames/>. Access on October 2nd 2024.

CASAMAYOR, R. et al. Bug localization in game software engineering: evolving simulations to locate bugs in software models of video games. In: **Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems**. Montreal Quebec Canada: ACM, 2022. p. 356–366. ISBN 978-1-4503-9466-6. Disponível em: <https://dl.acm.org/doi/10.1145/3550355.3552440>.

CHOI, J. O. et al. Playtesting with a Purpose. In: **Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play**. Austin Texas USA: ACM, 2016. p. 254–265. ISBN 978-1-4503-4456-2. Disponível em: <https://dl.acm.org/doi/10.1145/2967934.2968103>.

CHUECA, J. et al. The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games. **Information and Software Technology**, v. 165, p. 107330, jan. 2024. ISSN 09505849. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0950584923001854>.

CLEMENT, J. **Digital video game market revenue worldwide from 2017 to 2027**. 2023. Disponível em: <https://www.statista.com/statistics/1344686/global-digital-gaming-revenue/>. Acesso em: 19 de novembro de 2023.

_____. **Number of monthly active users on gaming platform Steam worldwide from 2017 to 2021**. 2023. Disponível em: <https://www.statista.com/statistics/733277/number-stream-dau-mau/>. Acesso em: 03 de fevereiro de 2025.

CODERRACH. **Is there a difference between Newb and Noob?** 2010. <https://www.reddit.com/r/AskReddit/comments/cxubi/is_there_a_difference_between_newb_and_noob/>. Access on November 18th 2024.

COPCHE, R. et al. Can a chatbot support exploratory software testing? preliminary results. In: **Conference: 26th International Conference on Enterprise Information Systems**. [S.l.: s.n.], 2024. p. 159–166.

_____. Exploratory testing of apps with opportunity maps. In: **XX Brazilian Symposium on Software Quality**. Virtual Event Brazil: ACM, 2021. p. 1–10. ISBN 978-1-4503-9553-3. Disponível em: <https://dl.acm.org/doi/10.1145/3493244.3493248>.

CRUZ, C.; HANUS, M. D.; FOX, J. The need to achieve: Players' perceptions and uses of extrinsic meta-game reward systems for video game consoles. **Computers in Human Behavior**, v. 71, p. 516–524, jun. 2017. ISSN 07475632. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0747563215300960>.

DARKSCARLETX. **The difference between a Noob and a Newbie**. 2015. <https://forums.daybreakgames.com/dcuo/index.php?threads/learn-of-the-difference-between-newbie-and-noob.259952/>. Access on November 18th 2024.

DEMARTINI, F. **Reembolsos de Cyberpunk 2077 representaram 9% do faturamento da CD Projekt Red**. 2021. Disponível em: <https://esportes.yahoo.com/noticias/reembolsos-cyberpunk-2077-representaram-9-135800899.html>. Acesso em: 15 de outubro de 2021.

DESURVIRE, H.; CAPLAN, M.; TOTH, J. A. Using heuristics to evaluate the playability of games. In: **CHI '04 Extended Abstracts on Human Factors in Computing Systems**. Vienna Austria: ACM, 2004. p. 1509–1512. ISBN 978-1-58113-703-3. Disponível em: <https://dl.acm.org/doi/10.1145/985921.986102>.

DEVELOPER, S. G. **Troll Games - Design a golden path**. 2019. <https://www.gamedeveloper.com/design/troll-games---design-a-golden-path>. Access on October 1st 2024.

DI MARTINO, S. et al. GUI testing of Android applications: Investigating the impact of the number of testers on different exploratory testing strategies. **Journal of Software: Evolution and Process**, v. 36, n. 7, p. e2640, jul. 2024. ISSN 2047-7473, 2047-7481. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/smr.2640>.

DRACHEN, A.; NACKE, L. E.; MIRZA-BABAEI, P. (Ed.). **Games user research**. First edition. Oxford, United Kingdom: Oxford University Press, 2018. ISBN 978-0-19-879484-4 978-0-19-183633-6 978-0-19-251391-5.

DUARTE, Y. et al. Exploratory testing strategies for video games:an experience report. In: **Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment**. Rio Grande (RS) Brazil: ACM, 2023. p. 46–55. ISBN 979-8-4007-1627-0. Disponível em: <https://dl.acm.org/doi/10.1145/3631085.3631227>.

_____. Exploratory testing for platform video games: strategies and lessons learned. **Journal on Interactive Systems**, v. 15, n. 1, p. 657–669, jul. 2024. ISSN 2763-7719. Disponível em: <https://journals-sol.sbc.org.br/index.php/jis/article/view/4156>.

DUARTE, Y.; POLITOWSKI, C.; ENDO, A. T. Towards a framework for exploratory testing in video games. In: **9th International Workshop on Games and Software Engineering (GAS 2025) in conjuction with the 47th IEEE/ACM International Conference on Software Engineering (ICSE 2025)**. [S.l.: s.n.], 2025. p. 1–8.

ELECTRONIC ARTS, I. **Download the EA app**. 2024. Disponível em: <https://www.ea.com/pt-br/ea-app>.

ENGSTRöM, H. et al. Game development from a software and creative product perspective: A quantitative literature review approach. **Entertainment Computing**, v. 27, p. 10–22, ago. 2018. ISSN 18759521. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1875952117300927>.

EPIC GAMES, I. **Epic Games Store**. 2024. Disponível em: <https://store.epicgames.com/>.

Escada Games. **Diver Down by Escada Games**. 2024. <https://escada-games.itch.io/diver-down>. Access on 02 July 2024.

GHAZI, A. N. et al. Exploratory testing: One size doesn't fit all. **CoRR**, abs/1704.00537, 2017. Disponível em: <http://arxiv.org/abs/1704.00537>.

Godot. **Godot Engine - Free and open source 2D and 3D game engine**. 2024. <https://godotengine.org/>. Access on 02 July 2024.

Harmony Honey. **Tiny Crate by Harmony Honey**. 2024. <https://harmonyhoney.itch.io/tinycrate>. Access on 02 July 2024.

HENDRICKSON, E. **Explore it!** [S.l.]: The Pragmatic Bookshelf, 2013.

IFTIKHAR, S. et al. An automated model based testing approach for platform games. In: **2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)**. Ottawa, ON, Canada: IEEE, 2015. p. 426–435. ISBN 978-1-4673-6908-4. Disponível em: <http://ieeexplore.ieee.org/document/7338274/>.

ITCH, c. **About itch.io**. 2024. Disponível em: <https://itch.io/docs/general/about>.

ITKONEN, J. **Empirical studies on exploratory software testing**. Tese (Doutorado) — School of Science, Aalto Universit, 2011. Disponível em: <http://urn.fi/URN:ISBN:978-952-60-4339-5>.

JAPASSOU; MACHADO, R. **Leap Hero(Gamescom Demo) by JapaSsou, Renato Machado**. 2023. <https://japassou.itch.io/leap-hero>. Access on October 15th 2024.

JEGERS, K. Pervasive game flow: understanding player enjoyment in pervasive gaming. **Computers in Entertainment**, v. 5, n. 1, p. 9, jan. 2007. ISSN 1544-3574. Disponível em: <https://dl.acm.org/doi/10.1145/1236224.1236238>.

JUSTESEN, N. et al. **Deep Learning for Video Game Playing**. arXiv, 2019. 1–20 p. ArXiv:1708.07902 [cs]. Disponível em: <http://arxiv.org/abs/1708.07902>.

JUUL, J. The game, the player, the world looking for a heart of gameness. p. 1–13, 2003.

KANER, C.; FALK, J.; NGUYEN, H. Q. **Testing Computer Software**. [S.l.: s.n.], 1999.

KANODE, C. M.; HADDAD, H. M. Software Engineering Challenges in Game Development. In: **2009 Sixth International Conference on Information Technology: New Generations**. Las Vegas, NV, USA: IEEE, 2009. p. 260–265. ISBN 978-1-4244-3770-2. Disponível em: <http://ieeexplore.ieee.org/document/5070627/>.

KASURINEN, J.; SMOLANDER, K. What do game developers test in their products? In: **Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. Torino Italy: ACM, 2014. p. 1–10. ISBN 978-1-4503-2774-9. Disponível em: <https://dl.acm.org/doi/10.1145/2652524.2652525>.

LEITNER, A. et al. Reconciling Manual and Automated Testing: The AutoTest Experience. In: **2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)**. Waikoloa, HI: IEEE, 2007. p. 261a–261a. ISBN 978-0-7695-2755-0. Disponível em: <https://ieeexplore.ieee.org/document/4076909/>.

LEVEAU, J. et al. Fostering the diversity of exploratory testing in web applications. **Softw. Test. Verification Reliab.**, v. 32, n. 5, 2022. Disponível em: <https://doi.org/10.1002/stvr.1827>.

LEWIS, C.; WHITEHEAD, J. The whats and the whys of games and software engineering. In: **Proceedings of the 1st International Workshop on Games and Software Engineering**. Waikiki, Honolulu HI USA: ACM, 2011. p. 1–4. ISBN 978-1-4503-0578-5. Disponível em: <https://dl.acm.org/doi/10.1145/1984674.1984676>.

LI, Z. et al. GBGallery : A benchmark and framework for game testing. **Empirical Software Engineering**, v. 27, n. 6, p. 140, nov. 2022. ISSN 1382-3256, 1573-7616. Disponível em: <https://link.springer.com/10.1007/s10664-022-10158-x>.

LIN, D.; BEZEMER, C.-P.; HASSAN, A. E. Studying the urgent updates of popular games on the Steam platform. **Empirical Software Engineering**, v. 22, n. 4, p. 2095–2126, ago. 2017. ISSN 1382-3256, 1573-7616. Disponível em: <http://link.springer.com/10.1007/s10664-016-9480-2>.

LIU, J.-W. et al. The role of Sprint planning and feedback in game development projects: Implications for game quality. **Journal of Systems and Software**, v. 154, p. 79–91, ago. 2019. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0164121219300974>.

LOVRETO, G. et al. Automated Tests for Mobile Games: An Experience Report. In: **2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. Foz do Iguaçu, Brazil: IEEE, 2018. p. 48–488. ISBN 978-1-5386-9605-7. Disponível em: <https://ieeexplore.ieee.org/document/8636923/>.

LTD., Y. G. **Home of GameMaker Game Engine**. 2013. <https://gamemaker.io/en/about>. Access on November 4th 2024.

LYNDSAY, J. Adventures in Session-Based Testing. p. 1–17, 2003.

MICALLEF, M.; PORTER, C.; BORG, A. Do Exploratory Testers Need Formal Training? An Investigation Using HCI Techniques. In: **2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Chicago, IL, USA: IEEE, 2016. p. 305–314. ISBN 978-1-5090-3674-5. Disponível em: <http://ieeexplore.ieee.org/document/7528977/>.

MIRZA-BABAEI, P.; MOOSAJEE, N.; DRENIKOW, B. Playtesting for indie studios. In: **Proceedings of the 20th International Academic Mindtrek Conference**. Tampere Finland: ACM, 2016. p. 366–374. ISBN 978-1-4503-4367-1. Disponível em: <https://dl.acm.org/doi/10.1145/2994310.2994364>.

MURPHY-HILL, E.; ZIMMERMANN, T.; NAGAPPAN, N. Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development? In: **Proceedings of the 36th International Conference on Software Engineering**. Hyderabad India: ACM, 2014. p. 1–11. ISBN 978-1-4503-2756-5. Disponível em: <https://dl.acm.org/doi/10.1145/2568225.2568226>.

MYERS, G. J. **The Art of Software Testing**. 3rd. ed. [S.l.: s.n.], 2012.

MåRTENSSON, T. et al. Efficient and effective exploratory testing of large-scale software systems. **Journal of Systems and Software**, v. 174, p. 110890, abr. 2021. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0164121220302806>.

NOVAK, J. **Desenvolvimento de games**. [S.l.]: Cengage Learning, 2017. ISBN 978-85-221-2725-2.

OBS Project. **Wiki OBS Studio**. 2022. <https://obsproject.com/wiki/>. Access on 02 July 2024. Acesso em: 27 de maio de 2022.

PARRISH, A. **Former Ubisoft executives arrested after sexual harassment investigation**. 2023. Disponível em: <https://www.theverge.com/2023/10/4/23901575/ubisoft-executives-arrest-sexual-harassment-investigation>.

PEREIRA, N. S. et al. Towards Automated Playtesting in Game Development. In: **Anais Estendidos do XX Simpósio Brasileiro de Games e Entretenimento Digital (SBGames Estendido 2021)**. Brasil: Sociedade Brasileira de Computação, 2021. p. 349–353. Disponível em: <https://sol.sbc.org.br/index.php/sbgames_estendido/article/view/19666>.

POLITOWSKI, C.; GUéHéNEUC, Y.-G.; PETRILLO, F. **Towards Automated Video Game Testing: Still a Long Way to Go**. arXiv, 2022. 1–7 p. ArXiv:2202.12777 [cs]. Disponível em: <http://arxiv.org/abs/2202.12777>.

POLITOWSKI, C. et al. **Assessing Video Game Balance using Autonomous Agents**. arXiv, 2023. 1–8 p. ArXiv:2304.08699 [cs]. Disponível em: <http://arxiv.org/abs/2304.08699>.

POLITOWSKI, C.; PETRILLO, F.; GUEHENEUC, Y.-G. A Survey of Video Game Testing. In: **2021 IEEE/ACM International Conference on Automation of Software Test (AST)**. Madrid, Spain: IEEE, 2021. p. 1–10. ISBN 978-1-6654-3567-3. Disponível em: <https://ieeexplore.ieee.org/document/9463010/>.

POLITOWSKI, C. et al. **Game Industry Problems: an Extensive Analysis of the Gray Literature**. arXiv, 2021. 1–18 p. ArXiv:2009.02440 [cs]. Disponível em: <http://arxiv.org/abs/2009.02440>.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. 7th ed. ed. Dubuque, IA: McGraw-Hill, 2010. ISBN 978-0-07-337597-7.

RIOT GAMES, I. **How to Play - League of Legends**. 2024. Disponível em: <https://www.leagueoflegends.com/en-us/how-to-play/>.

SANTOS, R. E. S. et al. Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners. In: **Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. Oulu Finland: ACM, 2018. p. 1–10. ISBN 978-1-4503-5823-1. Disponível em: <https://dl.acm.org/doi/10.1145/3239235.3268923>.

SATTAR, M. **All League of Legends champions ny release date**. 2024. Disponível em: <https://www.rivalry.com/pt/news/all-league-of-legends-champions-by-release-date>.

SCHULTZ, C. P.; BRYANT, R. D. **Game Testing: All in One**. Thomson Course Technology PTR, 2005. ISBN 978-1-68392-285-8. Disponível em: <https://www.degruyter.com/document/doi/10.1515/9781683922858/html>.

SESTINI, A. et al. **Towards Informed Design and Validation Assistance in Computer Games Using Imitation Learning**. arXiv, 2022. 1–10 p. ArXiv:2208.07811 [cs]. Disponível em: <http://arxiv.org/abs/2208.07811>.

SHAWAR, B.; ATWELL, E. Chatbots: Are they really useful? **LDV Forum**, v. 22, p. 29–49, 07 2007.

SMITH, G.; CHA, M.; WHITEHEAD, J. A framework for analysis of 2D platformer levels. In: **Proceedings of the 2008 ACM SIGGRAPH symposium on Video games**. Los Angeles California: ACM, 2008. p. 1–6. ISBN 978-1-60558-173-6. Disponível em: <https://dl.acm.org/doi/10.1145/1401843.1401858>.

SOUZA, M. et al. On the Exploratory Testing of Mobile Apps. In: **Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing**. Salvador Brazil: ACM, 2019. p. 42–51. ISBN 978-1-4503-7648-8. Disponível em: <https://dl.acm.org/doi/10.1145/3356317.3356322>.

SPEEDRUN.COM. **What is speedrunning?** 2014. <https://www.speedrun.com/support/learn/what-is-speedrunning>. Access on October 2nd 2024.

SRIRAM, V. **Automated playtesting of platformer games using reinforcement learning**. Tese (Doutorado) — Northeastern University, 2019. Disponível em: <http://hdl.handle.net/2047/D20335186>.

TRISP. **The Difference Between Noob and Newb**. 2013. <https://www.growtopiagame.com/forums/forum/other/everything-else/70716-the-difference-between-noob-and-newb>. Access on November 18th 2024.

VALVE, C. **Steam, The Ultimate Online Game Platform**. 2024. Disponível em: <https://store.steampowered.com/about>.

Valve Corporation. **Portal 2 on Steam**. 2023. <https://store.steampowered.com/app/620/Portal/>. Access on 02 July 2024. Acesso em: 27 de maio de 2022.

_____. **Portal on Steam**. 2023. <https://store.steampowered.com/app/400/Portal/>. Access on 02 July 2024.

WHITTAKER, J. A. **Exploratory Software Testing Tips, Tricks, Tours, and Techniques to Guide Test Design**. 1st. ed. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0-321-63641-4.

WINPRESS, M. **Little Spy by Martin Winpress**. 2024. <https://wimpress.itch.io/little-spy>. Access on 02 July 2024.

WOOD, A. **Riot Games Will Compensate 1,548 Women For Gender Discrimination**. 2023. Disponível em: <www.ign.com/articles/riot-games-will-compensate-1548-women-for-gender-discrimination>.

XPAW. **Scanning all possible Steam IDs, and what we have found**. 2020. Disponível em: <https://steamdb.info/blog/scanning-all-steam-ids/>.

ZHENG, Y. et al. Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning. In: **2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. San Diego, CA, USA: IEEE, 2019. p. 772–784. ISBN 978-1-7281-2508-4. Disponível em: <https://ieeexplore.ieee.org/document/8952543/>.

Appendix

APPENDIX A

xPloiT Subgoals and Questions

Subgoal	Identifier	Question	Element
Single Session Strategy			
#1 Learn about the game.	<SSS-LEARN-BASICs>	Does the game have a tutorial or prologue to teach you the mechanics, controls and features? Complete it.	Any
#2 Beat the game.	<SSS-BEAT-PLAYABLE>	Have you finished the available `game segment`? Complete it.	Any
Golden Path Strategy			
#4 Collect, explore and interact with the essential.	<GPS-OUT-ESSENTIAL>	Which outputs are essential to progression? Collect only those.	Output
#4 Collect, explore and interact with the essential.	<GPS-PATH-ESSENTIAL>	Which path is essential to progression? Cover that path.	Path
#4 Collect, explore and interact with the essential.	<GPS-NPC-ESSENTIAL>	Which interactions with NPCs (both friendly and hostile) are essential to progression? Interact with those.	NPC
#5 Collect, explore and interact with the effortless.	<GPS-OUT-EFFORTLESS>	Which outputs are less demanding to collect in terms of effort? Collect only those.	Output
#5 Collect, explore and interact with the effortless.	<GPS-PATH-EFFORTLESS>	Which path is shorter or less demanding in terms of effort? Cover that path.	Path
#5 Collect, explore and interact with the effortless.	<GPS-NPC-EFFORTLESS>	Which interactions with NPCs (both friendly and enemies) are less demanding in terms of effort? Interact with those.	NPC
#6 Collect, explore and interact as the designers supposedly want.	<GPS-OUT-DESIGNER>	Which outputs, in your opinion, were intended by the designer to be collected? Collect only those.	Output
#6 Collect, explore and interact as the designers supposedly want.	<GPS-PATH-DESIGNER>	Which path, in your opinion, was intended by the designer to be followed? Cover that path.	Path
#6 Collect, explore and interact as the designers supposedly want.	<GPS-NPC-DESIGNER>	Which NPCs (both friendly and enemies), in your opinion, were intended by the designer to be interacted with? Interact with those.	NPC
Noob Journey Strategy			
#7 Collect, explore and interact with the remaining.	<NJS-OUT-COLLECTED>	Which outputs were not yet collected? Collect those.	Output
#7 Collect, explore and interact with the remaining.	<NJS-PATH-EXPLORED>	Which paths were not yet explored? Cover those.	Path
#7 Collect, explore and interact with the remaining.	<NJS-NPC-INTERACTED>	Which NPCs were not yet interacted with? Interact with those.	NPC
#8 Collect, explore and interact ignoring the tips.	<NJS-OUT-TIPS>	Does the game offer any tips or guidance for collecting outputs? Ignore them.	Output
#8 Collect, explore and interact ignoring the tips.	<NJS-PATH-TIPS>	Does the game offer any tips or guidance for paths? Ignore them.	Path
#8 Collect, explore and interact ignoring the tips.	<NJS-NPC-TIPS>	Does the game offer any tips or guidance to NPCs? Ignore them.	NPC
Completionist Strategy			
#9 Find and collect every output from the game.	<CS-OUT-COLLECTIBLES>	Does the game have achievements, items, rewards, badges, or a score system? Collect them all.	Output
#9 Find and collect every output from the game.	<CS-OUT-MISSABLE>	Are there missable items or rewards? Collect them all.	Output
#9 Find and collect every output from the game.	<CS-OUT-HIDDEN>	Are there hidden items, areas, rewards or achievements? Collect them all.	Output
#9 Find and collect every output from the game.	<CS-OUT-SECRETS>	Have you searched for any secrets (e.g., objectives, interactions, areas)? Find them.	Output
#9 Find and collect every output from the game.	<CS-OUT-EASTER>	Have you searched for any Easter eggs in the game? Find them.	Output
#9 Find and collect every output from the game.	<CS-OUT-FULL>	Are there badges, titles, or additional rewards for fully completing the game or specific sections? Collect them all.	Output
#10 Explore the paths.	<CS-PATH-FURTHER>	Among the available paths, which one is furthest from the starting point? Cover that path.	Path
#10 Explore the paths.	<CS-PATH-TIME>	Which path is the longest or most demanding in terms of time? Cover that path.	Path
#10 Explore the paths.	<CS-PATH-EFFORT>	Which path is the longest or most demanding in terms of effort? Cover that path.	Path
#10 Explore the paths.	<CS-PATH-COLLECTION>	Have you completed the game by following the longest possible route to collect all outputs? Do it.	Output, Path
#10 Explore the paths.	<CS-PATH-BONUS>	Are there any tasks, quests, or side missions that you have not completed yet? Complete them.	Path
#11 Interact with every NPC.	<CS-NPC-ENEMY>	Does the game have enemies? Defeat them all.	NPC
#11 Interact with every NPC.	<CS-NPC-INTERACT>	Does the game have interactions with characters (NPCs)? Exhaust them all.	NPC
#11 Interact with every NPC.	<CS-NPC-PRIZE>	Do defeating enemies or interacting with NPCs unlock additional rewards or paths? Discover it.	NPC, Output, Path
Stress Test Strategy			
#12 Do the opposite of the tips.	<STS-OUT-TIPS>	Does the game offer any tips or guidance to outputs? Do the opposite.	Output
#12 Do the opposite of the tips.	<STS-PATH-TIPS>	Does the game offer any tips or guidance to paths? Do the opposite.	Path
#12 Do the opposite of the tips.	<STS-NPC-TIPS>	Does the game offer any tips or guidance to NPCs? Do the opposite.	NPC
#13 Check and cover other outputs, paths and interactions.	<STS-OUT-CONTROLS>	Does the game specify the controls? Check for trigger interactions with other buttons.	Output
#13 Check and cover other outputs, paths and interactions.	<STS-PATH-PROGRESSION>	Is the game progression path set? Cover the opposite or other directions.	Path
#13 Check and cover other outputs, paths and interactions.	<STS-NPC-PROGRESSION>	Is the game progression with NPCs set? Cover the opposite or other interactions.	NPC
Speedrun Strategy			
#14 Use the game objective.	<SRS-OUT-COLLECTIBLES>	Are any of the collectibles necessary to beat the game goal or objective? Bypass them.	Output
#14 Use the game objective.	<SRS-OUT-OBTAINABLE>	Are there any obtainable requirements to beat the game goal or objective? Bypass them.	Output
#14 Use the game objective.	<SRS-PATH-FASTER>	Among the available paths, which is the fastest to reach the game goal? Cover that path.	Path
#14 Use the game objective.	<SRS-PATH-SHORTER>	Which path is shorter to reach the game goal? Cover that path.	Path
#14 Use the game objective.	<SRS-PATH-MISSIONS>	Are there any tasks or quests required to unlock paths? Bypass them.	Path
#14 Use the game objective.	<SRS-PATH-LOCKED>	Are there any paths necessary to beat the game that are locked? Bypass them.	Path
#14 Use the game objective.	<SRS-NPC-ENEMY>	Is it possible to beat the game without defeating enemies (or defeating the smallest possible number)? Discover it.	NPC
#14 Use the game objective.	<SRS-NPC-INTERACT>	Does the game require obligatory interactions with NPCs? Bypass them.	NPC
#14 Use the game objective.	<SRS-NPC-PRIZE>	Do defeating enemies or interacting with NPCs unlock additional rewards or paths? Try to unlock them without fulfilling the requirements, or even check if the time needed to unlock them compensates and enhance the time needed to <SRS-PATH-FASTER> or <SRS-PATH-SHORTER>.	NPC
#15 Define a speedrun objective X that is not the current game objective.	<SRS-OUT-GOALCOL>	Are any of the collectibles necessary to beat your goal or objective? Bypass them.	Output

#15 Define a speedrun objective X that is not the current game objective.	<SRS-OUT-GOALOBT>	Are there any obtainable requirements to beat your goal or objective? Bypass them.	Output
#15 Define a speedrun objective X that is not the current game objective.	<SRS-PATH-GOALFAST>	Among the available paths, which is the fastest to reach your goal? Cover that path.	Path
#15 Define a speedrun objective X that is not the current game objective.	<SRS-PATH-GOALSHORT>	Which path is shorter to reach your goal? Cover that path.	Path
#15 Define a speedrun objective X that is not the current game objective.	<SRS-PATH-GOALMISS>	Are there any tasks or quests required to unlock paths to your goal? Bypass them.	Path
#15 Define a speedrun objective X that is not the current game objective.	<SRS-PATH-GOALLOCK>	Are there any paths necessary to beat your that are locked? Bypass them.	Path
#15 Define a speedrun objective X that is not the current game objective.	<SRS-NPC-GOALENEM>	Is it possible to beat your goal without defeating enemies (or defeating the smallest possible number)? Discover it.	NPC
#15 Define a speedrun objective X that is not the current game objective.	<SRS-NPC-GOALINT>	Does your goal require obligatory interactions with NPCs? Bypass them.	NPC
User Interface Strategy			
#16 Investigate the UI.	<UIS-UI-LOCALIZATION>	Is the game localization well done? Check it.	UI
#16 Investigate the UI.	<UIS-UI-HUDs>	Are the HUDs well positioned? Check it.	UI
#16 Investigate the UI.	<UIS-UI-SHORTCUTS>	Are every keybind/shortcut presented linked to a key? Check it.	UI
#16 Investigate the UI.	<UIS-UI-UNASSIGNED>	Is there any key with an unassigned function? Check it.	UI
Neighboring Strategy			
#17 Investigate the neighboring features.	<NS-OUT-NEIGHBORING>	How the bug affects neighboring outputs? Discover it.	Output
#17 Investigate the neighboring features.	<NS-PATH-NEIGHBORING>	How the bug affects neighboring paths? Discover it.	Path
#17 Investigate the neighboring features.	<NS-NPC-NEIGHBORING>	How the bug affects neighboring NPCs? Discover it.	NPC
#17 Investigate the neighboring features.	<NS-UI-NEIGHBORING>	How the bug affects neighboring UI elements? Discover it.	UI
#18 Investigate the consequences of the bug.	<NS-ANY-CONSEQUENCES>	What are possible consequences of the bug? Discover it.	Any
#19 Investigate features with potential to break.	<NS-ANY-FEATURES>	Are there any features that have the potential to break? Explore them.	Any
Overtime Strategy			
#20 Answer your questions.	<OTS-OUT-QUESTION>	Do you have any questions that haven't been answered related to outputs? Answer them.	Output
#20 Answer your questions.	<OTS-PATH-QUESTION>	Do you have any questions that haven't been answered related to paths? Answer them.	Path
#20 Answer your questions.	<OTS-NPC-QUESTION>	Do you have any questions that haven't been answered related to NPCs? Answer them.	NPC
#20 Answer your questions.	<OTS-UI-QUESTION>	Do you have any questions that haven't been answered related to UI? Answer them.	UI
#21 Extra thoughts.	<OTS-ANY-NOTES>	Do you have any ideas or notes that you haven't tested yet? Test them.	Any
#21 Extra thoughts.	<OTS-ANY-IDEAS>	Have you had any random idea that you want to test? Test it.	Any
#21 Extra thoughts.	<OTS-ANY-EXTRAS>	Do you think you could test more in any strategy but did not have the time? Test it.	Any