POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Test automation in video game development: Literature review and Sound testing implementation

Supervisors:

Prof. Riccardo Coppola

Prof. Francesco Strada

Candidate:

Mattia Riola

Academic Year 2022/2023
Torino

Abstract

The video game industry has been rapidly growing in both financial value and scale. The variety of video game genres, the prevalence of unconventional coding practices, constant alterations in game design, and the unpredictable behaviour of most games contribute to the complexity and lack of standardization in testing, often resulting in its underestimation.

This situation has motivated researchers to develop new techniques and tools that assist in various stages of the tougher phases of the game development lifecycle.

The presented thesis is structured in two different parts.

In the first part, a simplified version of the multi-vocal literature review is performed to find a helpful taxonomy to classify the testing levels, tools, goals and metrics used and developed in the literature. 765 papers were found from different repositories using specific search strings, after the first filtering phase, the remaining 118 papers were analysed defining 4 testing levels, 24 testing goals, 43 metrics and 129 tools/approaches. This review shows a lack of user experience testing especially in audio testing. In fact, there is only one free and open-source tool that aims to test the audio in a game directly. In the second part of this thesis, sound testing has been integrated with one of the most promising testing frameworks found thanks to the literature review. Sound testing is used to check the presence in specific time windows of 6 different sound assets in 3 different levels of LabRecruits played by iv4xr's agents checking their presence and the timing in which they are played.

Acknowledgement

I would like to extend my sincerest gratitude to the following important figures to whom I owe my deepest gratitude:

First and foremost, to my girlfriend, Chiara, whose constant love, support, patience, and understanding not only made this journey possible, but added immeasurable value to my experience. Your unwavering presence in this endeavor has meant more to me than words can adequately express.

To my family, who believed in my academic ambitions and supported my decision to study in a different city at the prestigious Polytechnic of Turin.

To my in-law's family, who instantly welcomed me as one of their own, I extend my heartfelt thanks.

To Marco and Gabriele, my teammates and friends, the pleasure of working with you on this journey far exceeded the challenges posed by the rigorous coursework. Our shared passion for this field turned our work into an exciting adventure.

To Alessio, Daniele, Paolo and Mattia, academic colleagues and friends, your insights and inspirations have been invaluable.

To my work for providing an environment that allowed me to implement the theoretical knowledge acquired during my academic courses. Your contribution to my professional growth has been significant.

Finally, a special mention goes out to my delightful pets: Burzi, Biscuit, Bruschetta, Red, Bugi and Biba. Your companionship brought joy and laughter into my everyday life, lightening even the most difficult moments.

Thank you all, from the bottom of my heart. You have made this journey an unforgettable chapter of my life.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

AIG
Automated test Input Generation

BDD
Behaviour-Driven Development

BDI
Belief-Desire-Intent

BT
Behaviour Tree

DFT
Discrete Fourier Transform

DSL
Domain Specific Language

EC
Exclusion Criteria

FFT
Fast Fourirer Transform

FPR
False Positive Rate

GUI

Graphical User Interface

GL

Grey Literature

HUD

Heads-Up Display

IC

Inclusion criteria

LCS

Longest Common Substring

MLR

Multivocal Systematic Literature Review

QA

Quality Assurance

RandR

Record and Replay

ROC

receiver operating characteristic

SLR

Systematic Literature Review

SUT

System Under Test

TDD

Test Driver Development

TPR

True Positive Rate

UX

User experience

VR

Virtual Reality

WL

White Literature

Chapter 1

Summary

This thesis is based on two primary objectives: advancing the understanding of game testing automation through a comprehensive literature review and a practical contribution to address identified gaps in the field. The findings of the literature review directed the focus towards a neglected area of game testing: audio testing. The second objective was to put the basis to fill the discovered gap in audio testing adding sound testing in one of the most promising testing frameworks.

1.1 Multi-Vocal Literature review

To understand the state of the art of the complex automated video game testing field, a multi-vocal literature review was conducted. This review classified levels, tools, goals, and metrics pertinent to game testing automation, thereby establishing a common taxonomy. 765 academic papers were identified from various repositories using specific search strings. Following an initial filtering phase, 118 papers were scrutinized, resulting in the definition of four testing levels, 24 testing goals, 43 metrics, and 129 tools or approaches.

The review conducted revealed several pertinent trends in game testing automation. Firstly, there has been a noticeable increase in the number of papers addressing game test automation, underlining the growing interest and importance in this field. Notably, a majority of these studies focus on system-level testing, indicating a preference towards a more comprehensive, system-wide testing approach in the gaming industry.

Additionally, the review showed that the most common testing goal in the literature is functionality testing, followed by regression testing and balanced testing. This suggests a strong emphasis on ensuring that games function as intended, are free from regressive bugs, and provide a well-balanced experience for players.

In terms of metrics used, the majority are oriented towards testing quality or game correctness. This trend is in line with the gaming industry's primary focus on delivering high-quality and correct gaming experiences.

Regarding testing tools, out of 129 identified, only 50 were found to be free and specific to video games. These tools primarily address logical bugs, UI bugs, stuck bugs, and graphical bugs. Although these tools cover a range of bug types, the review exposed a significant deficit in the area of audio testing, laying the groundwork for the second part of the thesis, which sought to address this gap.

1.2 Sound testing

Noticing this gap in audio testing served as a catalyst for the second part of the thesis, which involved integrating a sound recognition system into iv4xr, a promising game-testing framework. Six different sound assets across three different levels of a game called LabRecruits 1.1 were selected for the sound testing process. As iv4xr agents played these levels, the system monitored the presence and timing of the expected sound assets.

The sound recognition system that was introduced employed spectrogram analysis of the audio signals, enabling it to check the presence and absence of specific sounds within given time windows of the recorded gameplay. This enhancement was not initially part of the iv4xr framework but was introduced to broaden its testing capabilities. The new system was then utilized to verify if specific game events triggered their associated sounds, contributing to a more comprehensive testing scenario.

For the implementation, the sound recognition system needed to surmount initial challenges such as audio compatibility, for which a common audio format, the Waveform Audio File Format (WAV), was chosen. Additionally, an online converter was utilized to transform all sound assets in the game to ensure compatibility using, not only the same file extension but also the same sample rate and sample size.

Implementation

A Python script was developed to record and analyze the audio stream from the game while it was running. The recorded gameplay was then analyzed to match the occurrence of observed events with the presence of the corresponding sounds.

Different configurations of the sound recognition system were tested to optimize true positive and false positive ratios. This was conducted by plotting a receiver operating characteristic (ROC) curve for each specific configuration 1.2, adjusting parameters such as the 'fuz factor', chunk size and match threshold, and ultimately identifying the most effective configurations.

Despite the system's effective recognition capabilities, some challenges remain, such as the presence of background music or sounds, and the possible impact of sound effects applied based on the situation in the game. These aspects form part of the future avenues of exploration to improve the comprehensiveness and accuracy of audio testing in game testing automation.

Figure 1.1: LabRecruit screenshot

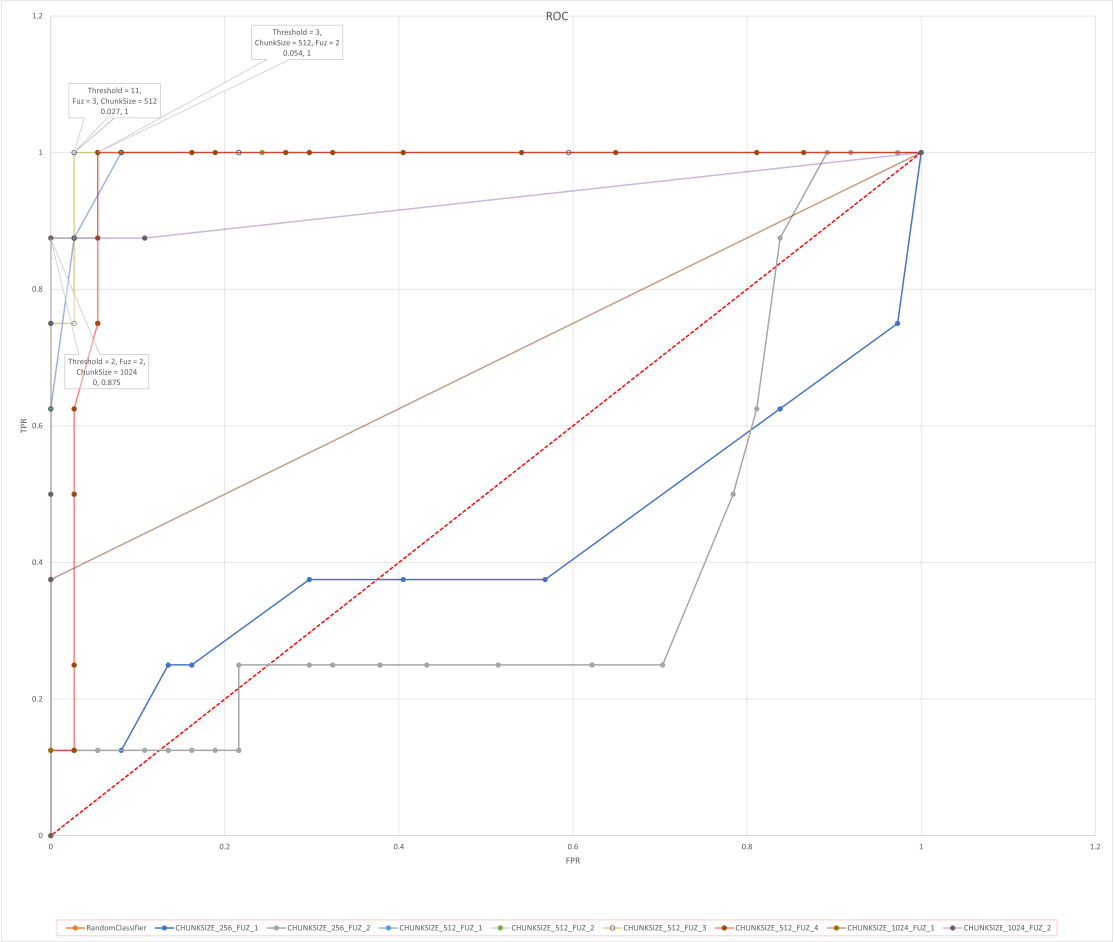

Figure 1.2: ROC curve varying the configurations of the sound recognition system

Chapter 2

Introduction

2.1 Importance of tests in video games

2.1.1 Game design growth

With the maturation of the video game industry, a substantial and discerning audience has emerged, expecting high-quality and immersive user experiences from their games. As per a report by gamesindustry.biz [1], the combined revenue generated from games across various platforms such as PCs, consoles, mobile devices, and the web stood at an astounding 134.9 billion dollars in 2018. Given the magnitude of the industry, the emphasis on quality assurance and verification procedures within game development has become essential [2].

2.1.2 Importance of testing

In the gaming sector, the first impression is a key factor for the success of a title; thus, only games of superior quality can anticipate success. Realising a highly anticipated video game in a "buggy state" can result in a loss in money, reputation and goodwill of the development company involved [3].

This is one of the most important reasons that show how software testing and error identification are paramount. The act of testing software involves assessing

and confirming that software products align with specified requirements and design, guaranteeing that the software functions as intended. Given the escalating requirements for more advanced software solutions, testing has become not only indispensable but also increasingly complex [2]

Also Taxonomies, along with the application of universal metrics and standards, play a fundamental role in organizing, consolidating, and structuring knowledge. The primary purpose of a taxonomy is to foster a uniform perspective on the interpretation and application of the existing knowledge in a specific field. As a result, taxonomies are invaluable not only in research but also significantly contribute to industrial practices and educational endeavours [4].

2.1.3 Literature gap

Existing literature reviews provide a solid foundation for understanding the current state of game testing. In the first section of this thesis, an extensive and updated literature review is conducted, which also incorporates sources from the grey literature. This review aims to identify definitions and trends in this crucial field.

The literature review conducted highlighted gaps in the current methods and tools available for detecting audio-related bugs in video games. Therefore, in the second part of the thesis, a sound recognizer that analyzes the spectrogram and checks for the presence of a sound in a specific time window has been implemented. Finally, this sound recognizer has been integrated with a promising testing framework found in the literature called iv4xr [5].

Chapter 3

Background

In this section, a description of software testing and video game testing is provided to contextualize the works in this thesis. Next, an overview of RiverGame [6], which is the only available free testing tool capable of finding in-game audio bugs found in the review, is provided.

3.1 Software testing

The software industry is growing as well as the game industry and the importance of software testing is more pronounced in every kind of software application. Testing is an integral part of the quality verification process for a System Under Test (SUT), the cost to guarantee a certain level of quality amounts to more than 50% of the total software development cost and it is often undervalued [7]. The main aim of the tests is to check if the SUT works as expected; if the SUT contains errors that make it deviates from the intended specifications, the tests have to give information about the bugs involved with the feature tested. The number of testing levels is a topic of debate, with some sources citing three levels [7] and others citing four [8]. These levels encompass Unit Testing (testing the smallest parts of the system), Integration Testing (focused on the complex integration of classes or procedures), System Testing (focused on the main or most risky flows of the application), and, according to some sources, Acceptance Testing (used by clients to evaluate the final product). Tests can be conducted by the development team, who has access to the SUT (white-box testing), or by an external party focusing on the SUT's functionalities (black-box testing). Black-box testing relies

solely on the SUT specification to generate and verify test cases, not the internal structure of the SUT.

3.1.1 Manual testing

Manual testing is a fundamental step in the software testing process, it requires an examination of test cases executed manually by real people. In this process, testers evaluate the software from the viewpoint of the end-user, cross-checking that it behaves according to the specifics set out in the requirements document. This fundamental testing method is particularly effective at detecting both overt and covert defects within the software. Any deviation from the expected output, when compared to the output produced by the software, is classified as a defect. These identified issues are then rectified by developers, and the amended software is sent back to the tester for ongoing validation. Even with the advent of automated testing, manual testing is an essential step that should be performed on all newly created software. Though this method may require considerable time and effort, it represents an investment to ensure the delivered software is devoid of bugs. Expertise in manual testing techniques is a must for this process, while familiarity with automated testing tools is not a prerequisite [8].

3.1.2 Automated testing

Automated testing is a multidimensional discipline that incorporates a variety of methodologies to effectively evaluate software behaviour and performance. Its application is not limited to the execution of pre-written scripts but explores a multitude of approaches, each aiming at different aspects of software testing. Core components include automation APIs and frameworks that serve as interfaces for obtaining GUI-related information and simulating user interactions. Record and Replay (R&R) techniques simplify the process of creating test scripts by recording and replaying user actions, which is beneficial for scenarios demanding precise event timing. Automated Test Input Generation (AIG) techniques are designed to automate the complex and time-consuming task of input generation with specific goals like achieving high code coverage or uncovering maximum bugs. Bug and error reporting/monitoring tools provide real-time insights into software performance and user interactions, aiding in crash diagnostics and problem reproduction. Device Streaming Tools facilitate the testing process by allowing developers to mirror or access devices remotely. As an evolving field, automated testing continues to adapt to new challenges presented by ongoing technological advancements and the

complexities of software development [9].

3.1.3 Automated vs Manual testing

So, comparing manual testing with automated testing, automation is much faster, is resilient to errors due to repetitive tasks executed by humans, is easier to build, is easier to organize and can give precise metrics about software coverage and performance statistics. These advantages suggest a high return on investment in test automation; however, automated testing cannot provide user-friendliness. [8].

3.2 Video game testing

According to the academic literature, the main issues in video game testing are:

- Coupling: Merging of game mechanics and user interface (UI) code;
- Scope: Games are extensive, making comprehensive coverage challenging;
- Randomness: Identical inputs may yield different outputs;
- Changes: Core game design is continuously evolving;
- Cost: The expense of an engineer surpasses that of a game tester;
- Time: Developers are primarily focused on generating content to meet deadlines;
- Fun-factor: How can fun be evaluated automatically?

For this reason, in the field of video games, manual testing prevails over automatic testing. Despite this, as demonstrated in the first part of this thesis, researchers are striving to facilitate these challenging testing activities by developing new tools and frameworks for the gaming industry [3].

3.2.1 Sound testing

In the literature review, only three papers analyse the audio to check if the game is working:

- Eric Nelson [10] proposes a novel testing methodology that leverages the recent surge in the popularity of live streaming. By utilizing a "think-aloud" testing approach during a live stream, the researcher posits that game developers can gain invaluable insights into player behaviours, contextual issues, and the overall quality of the game design. This method not only uncovers problems that are typically identified in traditional user testing scenarios but also reveals contextual behaviours that could otherwise go unnoticed in a lab setting. However, this approach primarily focuses on the level of acceptance testing and does not directly test the game using specific test cases. Furthermore, it doesn't provide the opportunity for regression testing, which is fundamental for maintaining the game's stability in its development and maintenance.

- Sogeti [11] proposes an approach that uses AI to scrutinize the game's audio assets. This is generally achieved by looking for similarities between a benchmark sound and the one observed during testing, rather than focusing on their differences. The model accounts for variations in time and amplitude, and calculates the cross-correlation between the two signals, while one remains fixed and the other is examined for overlap. The only problem is that Sogeti does not provide free open software.

- RiverGame [12] is a testing tool, compatible with multiple game engines, that provides sound, animation, performance and visual testing using behaviour-driven development (BDD) methodology for test specifications. Regarding sound recognition, it can identify if a specific sound played was played in the last N frames and if the sound feedback is correct. To do so, it uses two different methodologies to check the sounds in the game:

  - Speech-to-text

    * It uses Facebook's wav2vec [13] model which utilizes a Deep Learning and Transformers method to convert voice to text. Then, it compares the string obtained with the string expected using a similarity algorithm that considers the longest common substring (LCS).

  - Spectrogram comparison

    * It is used to check background music and sound effects. It uses Librosa library [14] to read sound data and convert it from the time domain to the frequency domain using the Fast Fourier Transform (FFT). The default sampling rate of the recorded audio is 44.1 kHz and each bin used to build the spectrogram is 1Hz large. The two spectrograms are then printed and their difference is analysed. However, the instance when the audio is reproduced must be specified in the test.

(further information about other testing features provided by RiverGame can be found in the results of the literature review)

3.2.2 iv4XR

The iv4XR framework is an innovative game-testing platform designed to automate tests in modern, dynamic gaming environments. iv4XR, leveraging the Belief-Desire-Intent (BDI) concept, employs specialized agents to execute complex testing tasks, offering an unprecedented level of testing fluency. Its flexible and extensible architecture enhances its testing capabilities by facilitating the easy integration of additional tools. As a testament to its potential, the framework has been successfully employed in a demo application, LabRecruits, to execute a variety of test cases.

(further information about iv4xr can be found in the results of the literature review)

Chapter 4

Literature review

The objectives of this study are:

- Identification of the different testing methods and goals in video games in order to establish a common (hierarchical) taxonomy
- Collection of a set of video game testing tools and frameworks for each testing method
- Understand the limitations of the existing automated testing tool

The need for these goals comes from the high complexity of testing video games. As highlighted in preceding chapters, the intricate nature of video games is shaped by elements like unpredictability, continual design changes, and more. This makes the task of testing a game more extensive than with standard software, due to a much larger field input space for test practices. Therefore, literature reviews are highly beneficial for keeping developers updated on the state of the art and for spotting new challenges for researchers in the specific field.

4.1 Multivocal Systematic Literature Review

A simplified version of Multivocal Systematic Literature Review (MLR) [15] applied to the field of test automation in the video-game industry will be defined, conducted and documented. A MLR differs from a Systematic Literature Review (SLR) [16]

in that it includes the Grey Literature (GL) in addition to the White Literature [17]. Grey Literature is defined as *what is produced on all levels of government, academics, business and industry in print and electronic formats, but which is not controlled by commercial publishers, i.e., where publishing is not the primary activity of the producing body* [18]. Adams et al. classify Grey Literature into three categories: 1st tier (or high credibility), which includes books, magazines, government reports, and white papers; 2nd tier (or moderate credibility), including annual reports, news articles, presentations, videos, question and answers websites; 3rd tier (or low credibility), including blogs, evidence from e-mails, posts on social networks [19]. The MLR methodology for SE has been provided only recently by Garousi et al. [15]. The authors base their guidelines on well-established methodological guidelines to conduct traditional SLR while stressing the benefits provided by having an overview of both the state of practice and academic state of the art. The combination of the two points of view, in fact, permits to analyse of emerging trends coming from dual perspectives, therefore reducing the risks of neglecting aspects of the topic. The choice of the MLR, compared to the SLR, is also given by the fact that the tools used by game developers are spread more easily from the GL like websites and blogs dedicated to game development. This way the trends of game testing can be more accurate and not only restricted to the academic field, using only white literature the results may have a bias since game testing (especially the game exploration using agents) is often used to test the performances of machine learning (ML) related algorithms.

4.2 Secondary studies

There are four works available in the literature discussing test automation in the gaming industry. Before proceeding with the literature review, secondary studies have been analysed and summarized. The report of these secondary studies can be found in 4.1.

4.3 Planning phase

Before starting the literature review a planning phase was conducted. In this phase, the most appropriate methodology has been chosen and structured taking into account other secondary studies and the objectives of the review. The mentioned secondary studies offer a preliminary glimpse of taxonomy in video game testing using WL and surveys, it can be enhanced and broadened to incorporate additional

Ref.	Year	Title	Research method	Description
[2]	2020	Video Game Automated Testing Approaches: An Assessment Framework	SLR	An attribute-based framework is presented to classify and compare different testing techniques and provide such aid to the game developers It analyses the benefits and limitations, the goals, the targeted game, the general applicability and the availability of the proposed approaches
[7]	2022	Towards Automated Video Game Testing: Still a Long Way to Go	LR and Survey	This study highlights the gap between the solutions for automated video game testing and the game developers' needs in practice. The problem with the solutions proposed in the literature the focus seems to be related to the performance of machine learning (ML) models instead of testing the game. In fact, in the survey, developers are sceptical about using automated agents to test their games.
[3]	2021	A Survey of Video Game Testing	Survey	This survey has found that Game developers rely mainly upon manual play-testing. The lack of test automation can be the next step in improving the quality of games while maintaining costs. However, the current game-testing techniques can not be applied to every type of game.
[20]	2019	Analysis of artificial intelligence applications for automated testing of video games		First, it is discussed the randomness in video games in relation to software testing. Then the paper provides an overview of existing automated video game testing approaches. It can be used as the first step in the research of automated procedurally generated game-level testing using AI.
[21]	2022	A Survey of the Software Test Methods and Identification of Critical Success Factors for Automation		The study examines various projects to discern and pinpoint the crucial elements that affect the testing effectiveness of a selection of projects. The objective is to identify elements that influence test efficiency and ascertain the right cases for automation. The research seeks to investigate the potential relationships among contributing factors and, ultimately, delve into the efficiency of test automation.

Table 4.1: Secondary studies

significant elements of in-game testing, such as metrics used and testing goal definitions. Plus, the trends of the testing tools and testing goals can be useful to have an overall understanding of the situation considering both WL and GL to include both academic literature and what is used by game developers.

4.4 Conducting

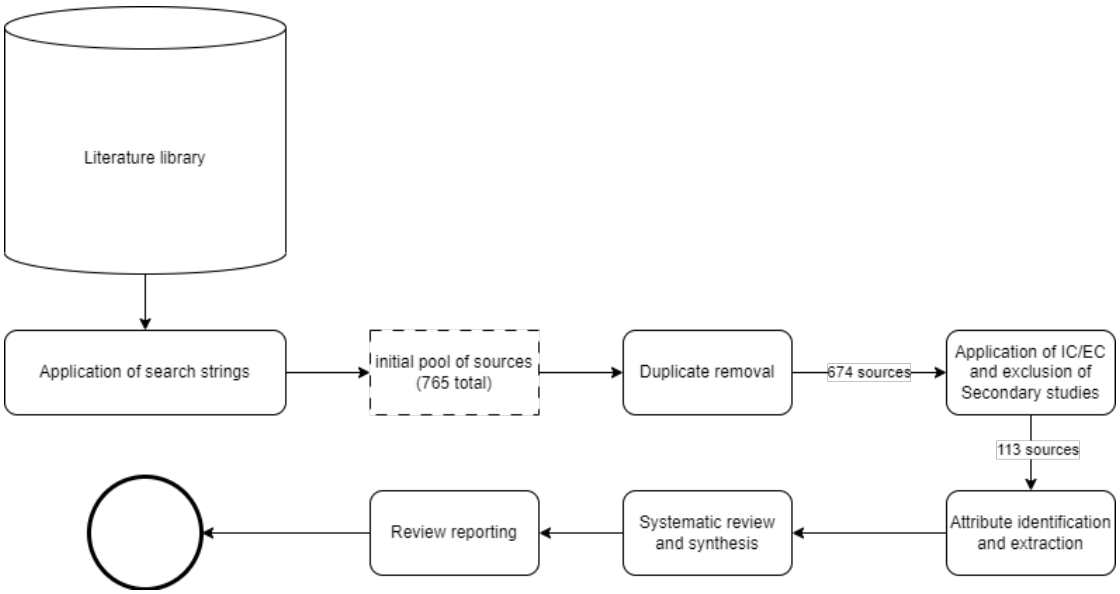

Figure 4.1: Process of the literature review

The following section presents the methodology implemented during the Conducting phase and its subsidiary stages. These encompass the identification of literature sources, creation of search strings, development of the paper selection strategy, and establishment of the data extraction protocol. A summarization of this procedure is depicted in the diagram shown in 4.1.

4.4.1 Research questions

To facilitate the creation of a taxonomy of codes to meet the research objective, a set of research questions was defined to guide the study. The answers to these questions aim to give inputs to the formulation of the taxonomy. As a secondary objective, the study aims to explore the present offer, both academic and industrial, of testing tools for video games

- RQ1: Testing methods and metrics
  - RQ1.1: What are the testing levels addressed in the video game industry?

  - RQ1.2: What is the goal of the tests?
  - RQ1.3: Which metrics are defined and used for automated video game tests?

- RQ2: Tools, software and approaches
  - RQ2.1: Which are the available testing tools and approaches for video games?
  - RQ2.2: Which type of bug is covered by the testing tools/approaches?

RQ1 is intended to classify different types of tests available in the gaming industry. While RQ2 aims to identify the tools that can be improved or extended during the second part of the thesis.

4.4.2 Acquiring data

To enable the the study's reproducibility, I limited the search results from 2012 to the end of 2022. In order to acquire all the articles in the repositories analyzed, I used [22], [23] and a Python script written for this activity that can be found here: [24]. To organize and facilitate the analysis, the data acquired has been saved in Zotero [25] and Notion [26]; Zotero is a tool often used for paper analysis and citation while Notion allows the creation of databases with relationships that have been used for tool and metric trends evaluation.

Repository selection

The repository selected mainly for white lists are:

- Scopus
- IEEE Xplore
- ACM Digital Library
- Science Direct
- Springer Link
- Google Scholar

For the GL only 2 website / search engine has been used:

- Google search
- Game developer

Search string

Once the repository is selected, searching strings are created to find the most pertinent results. Several search strings have been used due to different limitations in the advanced search in the repositories.

- String1:

```
1    allintitle: (game OR games OR gaming) (test OR tests OR
   testing) (automation OR automated OR automatic)
2
```

- String2:

```
1    (test OR testing OR tests)
2
```

- String3:

```
1    ((("Document Title":"game*" OR "Document Title":"gaming")
   AND ("Document Title": "test*") )(("Document Title":"game*"
    OR "Document Title":"gaming") AND ("Document Title": "test
   *") AND ("Document Title":"automat*" OR "Abstract":"automat
   *" OR "Document Title":"tool*" OR "Abstract":"tool*" ) ))
2
```

- String4:

```
everywhere: (game* AND test* AND automat*)
intitle: test*

```

- String5:

```
[[Abstract: game*] OR [Abstract: gaming]] AND [Abstract:
test*] AND [Abstract: automat*] AND [E-Publication Date:
(01/01/2012 TO 12/31/2022)]

```

Repository	Search String	constraints	results
Google Scholar	String 1	-	62
Scopus	String 1	-	34
IEEE Xplore	String 3	-	47
ACM Digital Library	String 5	-	162
Science direct	String 5	-	86
Springer link	String 4	computer science general article English	24
Game developer	String 2	Programming Design Art	252
Google Search	String 1	first 10 pages*	99

Table 4.2: Research table
(*this constrain was applied due to effort boundaries)

4.4.3 Acquisition of search results

The full research pool extracted contains 766 articles, after the removal of duplicates, only 684 articles remain. These 684 articles have to be analysed and filtered according to the following inclusion and exclusion criteria described in the next section.

Note: The research in Springer link was downloaded in two different batches:

one between 2012 - 2020 and the other between 2020 - 2022, due to the limit of the CSV download set at 1000 entries

4.4.4 Inclusion criteria

To ensure gathering only the sources relevant to our research goals, I defined the following inclusion Criteria (IC) and exclusion criteria (EC):

- IC1: The source is directly related to the topic of automated game testing
- IC2: The source is written in a language that is directly comprehensible by the author: English or Italian
- IC3: Source has been published from 2012 to the end of 2022
- IC4: The source is an item of white literature with the full text available for download and is published in a peer-reviewed journal or conference proceedings; or, the source is an item of 1st or 2nd tier Grey Literature

Exclusion Criteria are not listed since they are essentially the opposite of the Inclusion Criteria.

After applying these criteria, the pool of articles is narrowed down to 113 (57 White Literature and 56 Grey Literature).

4.4.5 Quality assurance

In order to evaluate the quality of the sources, only a distinction between white and grey literature has been performed.

4.4.6 Snowballing

Snowballing was not applied since the resulting articles are enough for this analysis.

4.5 Results

As depicted in Fig. 4.2 there has been a substantial surge of interest in the field of game testing in recent years.

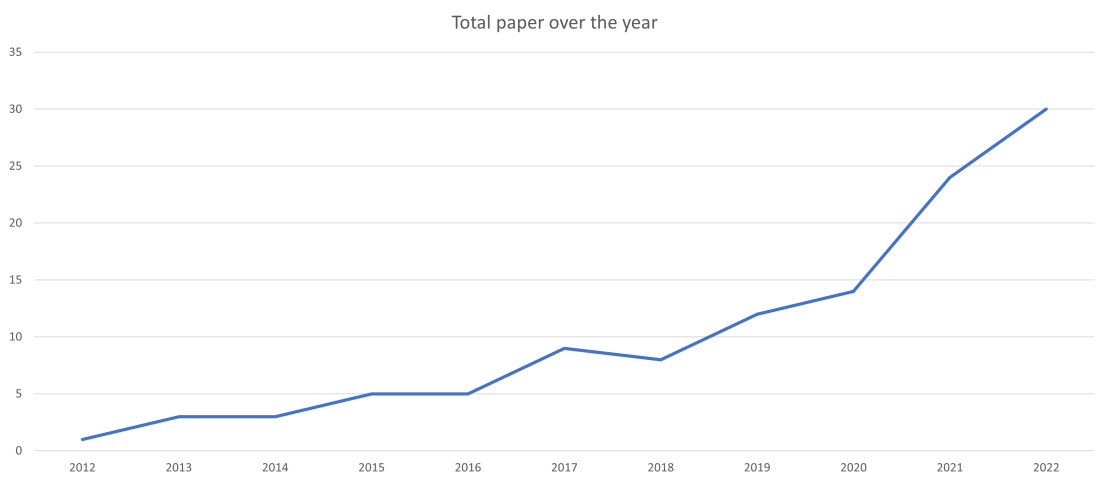

Figure 4.2: Number of paper released over the years

4.5.1 RQ1.1 - Testing Levels

Over the years, as shown in Fig 4.4 and Fig 4.5, there has been an observable increase in the number of articles addressing an extensive range of testing levels. Notably, advancements in machine learning, image recognition, and agent-based testing methods have allowed System level testing and acceptance testing to come to the fore. The flexibility offered by the lasts satisfies the vast diversity seen in the types and requirements of video games, thereby providing nearly comprehensive coverage.

The table 4.3 contains the papers in which the testing level is covered.

Fig 4.3 shows the resulting taxonomy of Testing levels found in the literature.

Despite the differences in the challenges addressed by general software testing and game testing, both domains share the main testing level classification:

Unit testing

Unit testing is a software testing method by which individual units of source code are tested without other external dependencies. It is considered the smallest testing part that can be tested [3] [7] [27] [28] [29]. [30] includes component testing described in [31] as a subgenre of Unit tests since is a special case of it. In the case of Component testing the component (or unit) can have dependencies with other components, which are mocked to isolate the test to that single component under test. Component testing is also called Actor testing [3]. Code units are examined independently in this process. Connections to other code modules are imitated through the utilization of dependency injection and mocking. Constructing unit tests can be costly, though the expense diminishes with improved tools and accumulated experience. For your most challenging and intricate code, the code that demands absolute reliability, unit testing is deemed highly beneficial [30].

Integration testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group [27] [30]. Modules can be code related, leaving the possibility to mock some part of the code to test different combinations of integrations [31] [7], or interaction between assets and game objects [32]. To do so the test can load only portions of the game and check small interactions, this is also called "Map testing" [3]. Designing for disability is very interesting to us at the moment. This is where you design parts of your system (think components or microservices) to eventually be thrown out. The whole system, the integration of the parts, is designed in such a way that the parts can be thrown out and replaced with little impact or damage to the system as a whole. The system continues to function and over time we replace entire sections of it [30].

System testing

System testing is testing for the main (or risky) flow of the application [7]. It is performed to check the overall behaviour of the system and interaction between the components which are combined as a system [27]. In [31] this testing level is described as the test that is usually performed by removing input layer triggering functions directly on the system. Anyways it can be considered as a subgenre of System testing since the overall System is tested but a single layer/component. In [30] system testing is called smoke testing or full build testing. A decent logging

and metrics output is suggested to improve system testing efficiency [30].

Acceptance Testing

Acceptance testing is a quality assurance (QA) process that determines to what degree an application meets end users' approval. It is used by clients to assess the final product by playing the game in different stages of the development [7] [33]. Depending on the stage, acceptance testing can be defined as Beta or Alpha testing in which unfinished versions of the game are played and evaluated. Beta testing can be open or closed depending on the people that can access and play the game.

Testing levels	is covered in	Total paper
Unit testing	[29],[34],[28],[35],[36],[37],[38],[39], [40]	9
System testing	[41],[42],[43],[11],[44],[45],[46],[47], [48],[49],[50],[51],[52],[53],[54],[55], [27],[56],[57],[58],[59],[60],[61],[62], [35],[63],[36],[64],[37],[65],[66],[67], [68],[69],[70],[71],[72],[73],[74],[75], [76],[77],[78],[79],[80],[81],[82],[83], [84],[85],[86],[87],[88],[89],[90],[91], [92],[93],[39],[12],[94],[95],[96],[33], [40],[32],[97],[98],[99],[100],[101],[102], [103],[104],[105]	75
Acceptance Testing	[106],[107],[45],[52],[55],[108],[37],[65], [75],[109],[82],[91],[92],[110],[93],[96], [101],[10],[104],[111]	20
Integration testing	[41],[43],[49],[50],[52],[53],[112],[56], [28],[59],[35],[113],[36],[37],[73],[114], [38],[81],[84],[87],[88],[93],[39],[40], [32],[98],[99],[103]	28

Table 4.3: Testing levels found in the literature

4.5.2 RQ1.2 - Testing goals

Extracting a testing goal taxonomy was challenging due to the discrepancies and overlaps in the definitions found during the review. The most common testing

Figure 4.3: Testing level taxonomy

goal is functionality testing since it is the most generic definition and it is the first thing that developers try to test because it is easy and because of common coding practices such as test-driven development (TDD). It is followed by regression and balance testing due to their importance in the development process. Having a balanced game is one of the key success factors for game development while regression testing is useful for the maintenance of a game. Even though User Experience (UX) testing is crucial, its complexity in automation has not gained substantial popularity in the literature. In fact, the limited number of articles

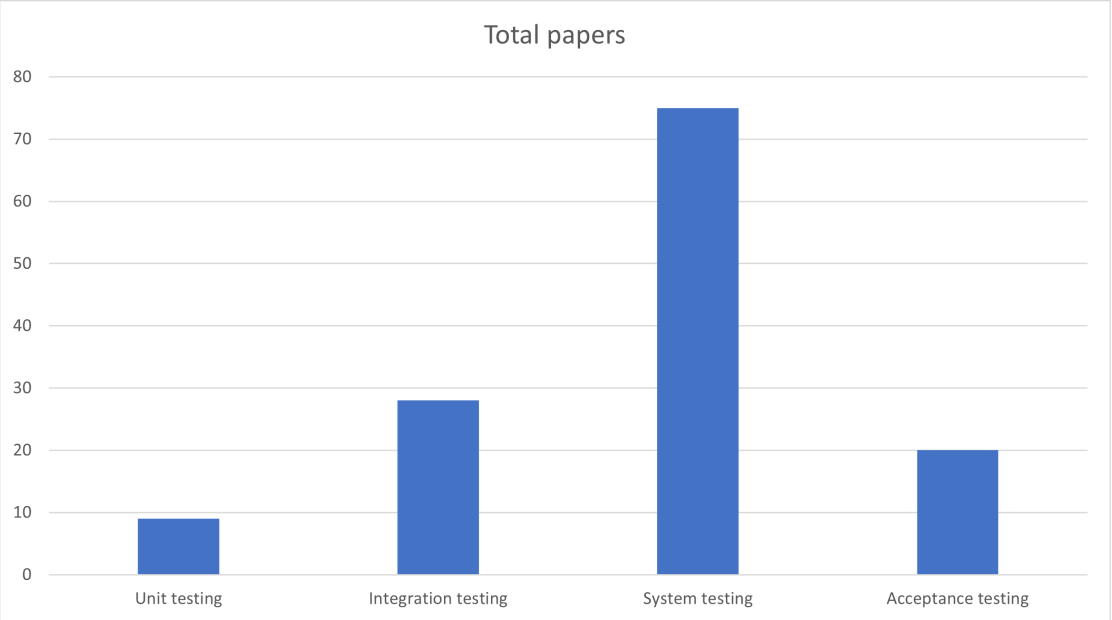

Figure 4.4: Number of articles published that cover that specific testing level

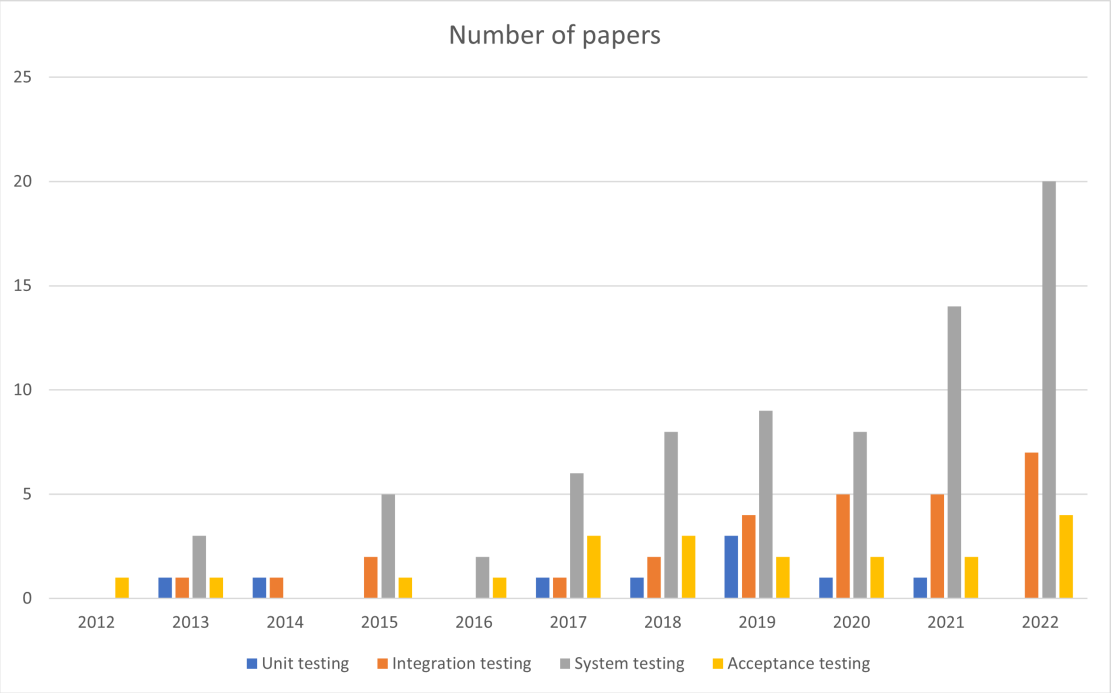

Figure 4.5: Number of paper over the year that covers that specific testing level

addressing this topic employ innovative and revolutionary approaches for the sector that are not yet being used by development companies. Security, Recovery, and tree testing were not covered at all by the literature since these types of testing are often performed manually while, in the filtered literature, only automated techniques were considered.

The following are the testing goals extracted from the literature:

Functionality testing

A detailed definition can be found in other papers like in [2], [115], [116] and [117]: the goal of functional testing is to ensure that the game is behaving as expected. To do so, games could be checked by verifying their functionalities, software code, the control flow of events, or the flow of data [2]. In [118], [119], [116], [120], [121], [122], [123], [115] [124], [125] the definition of Functionality testing in videogames is very generic. According to that, functionality tests look for general problems within the game itself or its user interface, such as stability issues, game mechanic issues, and game asset integrity. The definition became problematic, including stability issues and asset integrity, since it overlapped with other testing goals like visual testing, performance testing, and soak testing. In [126] functional testing is only related to the back-end component where the back-end component is defined as code, libraries and scripts in a videogame. The same criteria were applied to [124] which includes the application's performance during instant restart, switch-off and crash situations. Physical correctness described by [2] as the state of physical properties in the game world such as collisions, frictions, gravity, etc; is also considered a specific type of functionality testing because physics in the game can be strongly bounded to its functionalities, especially in the 3D games that have physics engines. In [125], tests on natural phenomena (like waves, wind, clouds), bullet movement etc are considered 3D testing. According to that definition, 3D testing also includes realistic animations, the performance of 3D objects and other parts of the game that make it realistic and, in this case, there was the same problem of overlapping definitions mentioned before. However, these physics phenomena can also be present in 2D games but, since the mentioned mechanics can be implemented in both 2D and 3D games, there is no need to distinguish between 3D to 2D environments, so the name "3D / 2D testing" is more appropriate. [75], [117] and [127] also consider tests on the menu and UI-related functions, in this case, UI testing can be included in functionality testing only if the features are tested and not the usability nor asset integrity. [115] considers functionality testing the tests on installation, social network options, payment gateways, active functionalities when the application is in minimized mode and many more.

Regression testing

Regression test is defined by [80] as follows: " Given a game M(S, A,r,Y) and its updated version M'(S', A',r',Y'), regression testing aims to generate test cases that capture the differences between the two versions as much as possible such that the regression bugs are more likely to be detected. Intuitively, regression testing aims at maximizing the exploration of the differential behaviours of two versions of the game ".

Regression testing aims to find bugs related to changes/modifications being made over due course of time. This type of testing confirms attributes like permanence, uniformity, usability and functionality of the software [27].

This is also performed once a bug is found and the programmers have fixed The aim is to check whether the bug is still there and whether the fix caused something else to break [118] [119] [116] [120] [122] [128] [126] [115] [124] [90]. With this game testing technique, a developer could re-run the previously conducted tests and compare the current vs. old results to see if there are any errors [122] [115]. In [117], regression testing aims to run the test cases on older devices, browsers and OS versions. The given definition is similar to the one that other articles give for compatibility testing. It seems that in [117] there is some confusion and some definitions are mixed, for this reason, the definition given by [117] is discarded. In [41], there are mentioned some typical problems related to design changes that cause regression bugs. These design changes are categorized in:

- Location change
  - entity location changes carried out by developers due to level design changes.
- Layout changes
  - change in the shape of the game world such as adding new obstacles
- Logic changes
  - changes related to the game's logic for example the behaviours of certain interactable objects such as the connection between certain buttons on certain doors.

According to [41], the main problem of regression testing video games was the overall effort and cost related to a minor change in the game design. When a

regression test fails, this does not imply that a bug is introduced and the test has to be modified according to the new design. Consequently, this escalates the time, effort, and financial resources required for the regression testing process, making it a significant challenge for video game developers.

Balance testing

This testing objective focuses on verifying the fairness of the game and the balance of the game's parameters [2] also in terms of level difficulty [129]. This type of testing is something unique to games and cannot be used in general software [129]. Balance testing can be done well only with a vast knowledge of game design and how the target audience responds to different difficulty levels. For this reason, this type of test is tough and the use of AI techniques to emulate different playstyles seems to be the answer to this problem [43].

[121] affirms that the balancing of a game can be defined as the operability of software (according to ISO 25010 [130] definition). In [94] gameplay testing is explained using examples. It can be considered as balance testing even if some specific examples can also fit other testing goals. I.e. checking if spawning conditions are matched is strongly related to functionality testing since it could be related to logic errors in the code. Another example classified as gameplay testing is the presence of visual bugs when a player enters inside a car after pressing a certain mapped button, this could be categorized as visual testing. However, these types of bugs alter the balance of the game.

Flow testing

In [32] flow testing is meant to encapsulate the game's responses, as perceived by the users, into a visual representation. The model, which consists of flow, events, actions, states, primitives, and terminator elements, visually conveys the testing process by illustrating and traversing the potential pathways. Game design correctness, Progression and learnability [2] can be considered as a subgenre of flow testing. Game design correctness is described as the testing goal meant to test various aspects that affect the user experience. However, the example given by Albaghajati et al. related to incorrectly placed game objects and the bugs related to that kind of situation are closer to the functional testing goal. World holes, stuck spots, violation of game rules etc. can sometimes relate to specific functionalities of the game; however, there are situations in which the progression of the game may be broken by these kinds of bugs. So this testing goal is between Flow testing

and Functional testing. Progression and learnability are defined as the testing goal that aims to verify that the player is going to be able to learn the game, learn the game, progress through levels, and complete the game. In [97] game flow testing is described as the testing approach in which the progression of the game is not guaranteed.

Multiplayer testing

This testing objective aims at multiplayer games and testing the networking stability and capacity [2]. It is also called multi-user testing, it also tests all the features of the game related to multiplayer mode; in fact, this type of test is one of the most challenging type and time-consuming types of testing [75]. Some authors in this survey report [3] claim that multiplayer testing requires many players simultaneously in the game world, with computer-controlled opponents and different game servers but, as it can see in [104], this isn't required. It is possible to use agents and bots to test the overall multiplayer system. However understanding multiplayer game design, and how to test efficiently as a team is required knowledge for this type of testing [129].

The tests should also ensure that all connectivity methods (modem, LAN, internet) are working [118] [119]

(UX) Usability testing

Usability testing is not the same as fun-factor testing [3]. It tests the experience given to gamers in terms of usability and playability. This is one of the main aspects of mobile game mobile apps that are tested by executing various user-centred operation scenarios in diverse environmental contexts [131]. The operability of software is a category used to define software quality that encompasses both the aspects of UX: usability and fun factor. In the case of usability testing some factors can be included, i.e. the learnability of the game mechanics, if the user can easily start playing, and the realism of the game (in terms of physics and graphics) can be included in usability testing [121]. In [129] realism testing is defined as a type of testing on its own since, in some cases like in simulators, is one of the most important aspects to test.

In this case, user interaction, game responsiveness and background events such as interruptions and battery consumption are checked [117] [75].

Usability and user experience involve any aspect of a video game with which players interact, like menus, audio, artwork, underlying game mechanics, etc [7]. In [129] audio testing is defined as a separate type of testing because of its complexity and unicity. In fact, in games, there is the unique use of sound that, not only should the audio play without stuttering or missing elements, it should also add to the gameplay. This requires extensive audio skills and a specific understanding of game audio.

The purpose of usability testing is to reveal areas of the game in which the player experience does not match the design intent [102].

In the context of virtual reality (VR) games, user experience (UX) is commonly assessed through subjective and objective methods. Subjective methods offer a convenient way to rate usability, comfort, satisfaction, and other relevant aspects [48].

(UX) Fun factor testing

This testing goal is on the emotional level of the player, it aims to test the fun factor in terms of engagement, stress excitement etc. According to ISO25010 definition, [130], It can be related to software operability. The fun factor of a game is strongly related to the attractiveness, the learning curve of game mechanics etc [121]. Fan factor testing is something unique to games and cannot be used in general software [129]. This type of test is one of the hardest to automate and this is one of the main reasons that slows the automation of the test process in the game industry. [3]. Fun factor testing is also called "playtesting" in [102], they also say that this kind of test is focused on players' opinions to illuminate areas of the game in which player experience does not map onto design intent. In [118] Player experience modelling can relate to fun factor testing since it refers to attempts to mathematically model player experience and predict a player's preference for liking or not a game.

Mobile game testing

This testing goal includes every test goal already explained but concerning the popular mobile game platform such as Android and iOS [119].

Performance testing

Tests the stability of the game in terms of performance using performance metrics such as FPS, RAM, GPU and CPU usage, average load time, battery usage [2] [75] [117] [128] [127] [33] [35]. These metrics are also used in ISO 25010 [130] to define the performance quality of software [121]. In [116] response time across different network types, in client and server transactions and other network-related parameters are also included in the definition. However, such parameters should be included in multiplayer testing and not in performance testing according to other definitions. Afterall some [116] and [117] define some interesting performance metrics like jittery connections, packet loss, data fragmentation, network coverage and peak load performance; these metrics can be used to evaluate the performance of a game but they are strongly related to the network part of the game that is usually related to multiplayer features. In [117] there is another testing goal called Graphic performance testing. Graphic performance is strongly related to graphic optimization while general performance testing is about hardware usage. For simplicity, I will consider Graphic performance testing as a subgenre of performance testing.

Explorative testing

Creative thinking and an unconventional approach are used here. No pre-formed conditions and tasks. It is an effective method that detects specific bugs and errors at any development stage, which would have gone unnoticed in the case of classical testing [126]. Explorative testing is executed on the entire system on the fly emphasizing more "imagination ability" rather than traditional testing using the concept of "Thinking and Investigation" [124]. Explorative testing is often based on agents that explore the entire game to find bugs in the game.

Visual testing

The purpose of this test goal is to check the visuals of the games., such as rendering, shaders, game UI, 3-D models, animations, etc. [2] [30]. This type of issue can also be detected with screenshots [3]. In [94] 3 testing goals can be associated with visual testing (they are defined with some practical examples):

- UI testing

  - When the player shoots someone, does the score update on the HUD? After the end of a match, does a specific menu appear on the screen?

- Rendering testing

  - Are post-processed effects visible after a specific event that should trigger that effect? is the camera centred correctly on the screen?

- Animation testing

  - Is the character moving using the selected animation in the right direction over a sequence of N frames?

Combinatorial testing

Test the game using a set of combinations of values of the parameters. Tests are systematically generated by identifying each distinct attribute (or parameter) which can be changed to data or configuration. [116] [126] [115] [124]. Some of the parameters of games include Events, settings, gameplay options, character attributes, customization options, and hardware configurations [120]. Combinatorial testing can use: category partition testing, pair testing and catalogue-based testing [115]. In [120], test a game with a different hardware configuration is included in the combinatorial tests, but this type of test is intended to verify compatibility problems between different devices, operating systems and browsers.

Load testing

Load tests are designed to determine the system's boundaries, for instance, how many players an MMO server can accommodate, the number of active sprites a screen can display, or the number of threads a specific program can execute concurrently [118] [116]. So it is designed to test heavy activity and whether the application can function property [119] [116] [122] [128]. In [117] Load testing is defined as a non-functional testing process that checks things like the maximum number of players that can play on a server, if the player can communicate with the server and how much memory the game uses. In some cases, like in the situation described by the last definition, Load testing is the union of performance and multiplayer testing where the scalability and the sustainability of the application are tested. In [33] what other papers call limit testing is called stress testing.

Localization testing

Localization testing aims to evaluate the game's quality, taking into account the cultural context and language of the target country. This becomes essential when a game is targeted for the global Game titles, content, and texts need to be translated and tested with devices in multiple languages [118] [119] [128] [123] [131]. Special attention should be given to regions like the MENA (Middle East/North Africa), considering aspects such as pseudo-localization testing, Arabic localization (Right-to-Left text support, Bi-directional displays), and local time/date, address formats, currency, and specific local requirements [125]. In [120], 2 types of issues related to localization testing are mentioned: Language issues, which refer to spelling, grammar, numeric formats, measurement system, voice-over, translation and so on and Visual issues which are related to the visual representation of the characters/string of the GUI like font issues', visual placement and characters that are not recognized.

Compatibility testing

Compatibility testing checks whether the game runs smoothly on different hardware and software configurations. The hardware encompasses brands of different manufacturers and assorted connected peripherals such as gamepads, joysticks, screens and other similar gaming paraphernalia [118] [119] [116] [121] [122] [126] [123] [131] [125] [115] [124]. The software compatibility is related to the co-existence between the game and other programs. For instance, many players run programs like Spotify and Team Speak in the background while they play [121]. In order to test the compatibility the test can run in different screen sizes, devices, browsers and operating systems [116]. This testing goal is particularly important in PC and mobile games since there is a large number of devices with diverse specifications. For this reason, developers have to be sure that the game works properly on different devices [102] even if this is a hard task. In fact, a lot of bugs related to compatibility issues are not found during the testing phase [120]. In [75] there is no clear distinction of compatibility testing; we can find testing on changes in screen resolutions and orientations under the "user interface and functionality" type of testing. However, the tests that check the correctness of the graphics in a different type of screens is related to compatibility testing. In [117] there is no definition of compatibility testing, however, the check on device compatibility is mixed in other descriptions (especially in UI/Functionality).

Clean room testing

In [32] clean room is defined as the process in which the game is tested on the assumptions of how the players will play the game. The test cases are generated based on the data of users' tendencies. Although [119], [116], [126], [115], [124], [120], [126] and [124] define it as the software development process intended to develop gaming software with a certifiable level of reliability. The main purpose of clean room testing is to produce minimal defects by combining mathematical and statistical reasoning and design refinement during test case generation and testing.

Play testing

The initial release of a game, often publicly available and sometimes unfinished, is commonly referred to as the "beta" version. During this phase, thousands of fans may discover bugs that went unnoticed by the developer's testers, as noted in references [118], [119], and [122]. However, it's important to remember that the primary focus of players during this phase should be evaluating the overall game experience in terms of fun factor, difficulty levels and balance, rather than solely identifying bugs. References [116], [122] and [115] emphasize this point. Based on these sources, although beta testing reveals both user experience issues and bugs, the primary objective ought to be assessing the gaming experience from the player's perspective. In fact in [124] both functional and non-functional elements are included in playtesting. [120] gives 4 different types of playtesting:

- Gross Playtesting
  - This is done when the first draft of the playable game is done in the development cycle. It is usually performed by the design team to check the gameplay and make sure that it's smooth
- In-house playtesting
  - This type of testing is performed by an in-house team or contract gamers. These gamers will be proficient in gaming, and they will go through every aspect of the gameplay to ensure that the gameplay is smooth and the game is interesting and can lure in a lot of users.
- Blind testing
  - Usually, a beta version of the game will be sent to selected players and will ask them to do a survey or log the issues they have found in the

gameplay. Since the game is tested by real users, insights gained will be much more useful than relying on someone who has preoccupied thoughts in mind.

- Final playtesting

  - The mechanics of the game won't be considered in this type of testing. The aesthetics of the game will be fine-tuned upon the suggestions of testers.

Soak testing

Soak Testing is a technique that consists in leaving the game running for a long time (hours or days) in various modes of operation such as idling, pausing, or at the title screen. This testing requires no user interaction beyond initial setup and is usually managed by lead testers. Automated tools may be used for simulating repetitive actions, such as mouse clicks [118] [119] [116]. Some logs can be generated during the soak testing phases to check the performance by combining performance testing with soak testing.

Ad hoc testing

In [32] Playtesting and ad hoc testing seem to be the same thing. However, according to other definitions like [120], [116] and [122], ad hoc testing is an unplanned testing technique that is generally utilized for breaking down the system where testers randomly test the app without test cases or any documents while in play testing the testers play the game and give feedbacks on the overall player experience of the game. So, the difference between ad hoc and playtesting is the actual objective of the tests.

Plus, due to its unplanned chaotic structure, ad-hoc testing can find bugs that can be found only when a random combination of actions happens [122]. This allows developers to find these errors at times especially when the developers try to fix a bug reported by some player.

Social media integration

Integration with social media is a critical aspect of your game. Many games allow users to share their scores with their social network in public or private feeds. To ensure full functionality and ease of use, this integration should be tested on Android and iOS devices, with various OS versions and device combinations. [117]. Even if this testing goal seems to be a subgenre of compatibility testing, some papers consider it as a separate testing goal due to its relevance in video game industries. I.e. the wildly successful game, Angry Birds, used social integration to keep their players hooked to the game. Players were prompted to link their Facebook accounts and invite their friends to play the game. Players received in-app rewards for recommending friends, which they could use to purchase in-game items [117]

Compliance testing

To ensure full functionality and ease of use, this integration should be tested on Android and iOS devices, with various OS versions and device combinations. [117]. Even if this testing goal seems to be a subgenre of compatibility testing, some papers consider it as a separate testing goal due to its relevance in video game industries. I.e. the wildly successful game, Angry Birds, used social integration to keep their players hooked to the game. Players were prompted to link their Facebook accounts and invite their friends to play the game. Players received in-app rewards for recommending friends, which they could use to purchase in-game items [117]

Security testing

Third-party tools including advertisements and payment gateways can present potential vulnerabilities that can be used by attackers [117]. This type of testing goal is important to ensure security, particularly in multiplayer games, where security properties like confidentiality, integrity, accountability and authenticity must be checked to deny unauthorized users or systems to read or modify protected data [121] [119] [128] [123] [33]. [75] and [125] outline the importance of checking the vulnerabilities and licensing restrictions every time a developer uses third-party code.

Recovery testing

This testing goal aims to check how well the game can be recovered from crashes, hardware failures, and other similar failures [128] [123]. The game is forced to fail, and later it is evaluated how it recovers from the failure conditions and the environment. [116]

Figure 4.6: Testing goals taxonomy Part 1

Testing Goal	Has been goal of	# papers
Functionality testing	[41],[42],[106],[43],[44],[45], [29],[49],[34],[52],[53],[54], [27],[108],[132],[57],[28],[58], [59],[60],[61],[62],[113],[36], [64],[66],[68],[69],[70],[71], [72],[74],[76],[133],[78],[79], [114],[38],[84],[85],[87],[89], [90],[110],[93],[39],[12],[94], [95],[96],[33],[40],[32],[100], [101],[102],[103],[104],[134], [10],[99],[81],[67]	63
Regression testing	[41],[49],[34],[108],[132],[60], [36],[64],[80],[83],[88],[12], [103],[38],[45],[90]	16
Balance testing	[43],[46],[55],[58],[78],[91], [92],[95],[98],[114],[10],[93], [69],[94],[71],[97]	16
Flow testing	[41],[112],[59],[70],[12],[94], [32],[103],[104],[114],[57],[100], [106],[49],[97]	15
Multiplayer testing	[106],[46],[57],[35],[36],[65], [70],[73],[96],[97],[104], [134],[10]	13
(UX) Usability testing	[107],[43],[46],[48],[109],[82], [84],[100],[102],[111],[38],[10]	12
Mobile game testing	[42],[45],[61],[62],[35],[64], [76],[85],[100],[102],[101],[77]	12
Performance testing	[46],[53],[60],[35],[63],[133], [38],[96],[104],[64],[36],[97]	12
Explorative Testing	[40],[114],[113],[84],[69],[50], [27],[81],[71],[79],[42]	11
Visual testing	[38],[101],[10],[100],[94],[27], [67],[85]	8
Combinatorial testing	[41],[46],[112],[113],[68],[32], [100],[103]	8

Table 4.4: Testing Goals found in the literature Part 1

Testing Goal	Has been goal of	# papers
Load testing	[46],[63],[36],[65],[96],[97], [38],[64]	8
Localization testing	[73],[82],[86],[88],[12],[38], [100]	7
Compatibility testing	[108],[86],[101],[102],[38],[49]	6
Clean Room testing	[41],[34],[103],[38]	4
(UX) Fun factor testing	[107],[114],[10]	3
Play testing (Alpha/Beta)	[106],[10]	2
Soak testing	[12],[104]	2
Ad hoc testing	[34],[53]	2
Social media integrations	[10]	1
Compliance testing	[12]	1
Security testing		0
Recovery testing		0
Tree testing		0

Table 4.5: Testing Goals found in the literature Part 2

4.5.3 RQ1.3 - Metrics

The literature revealed and defined 43 different metrics. The metrics can be used for four main purposes:

- Test quality (15 metrics)
  - How the test is good and easy to write/execute
- Game correctness (21 metrics)
  - How well the game is tested and covered by the tests
- Game balance (6 metrics)
  - How the game is balanced and playable
- Game performance (7 metrics)
  - How the game runs in terms of efficiency

Figures 4.12, 4.13, and 4.14 showcase the associated taxonomy of the metrics that were identified in the literature.

Figure 4.11 exhibits a clear emphasis on Game Correctness and Test Quality metrics, whereas there seems to be a noticeable deficiency of metrics related to Balancing Testing and Game Performance.

Figure 4.10 emphasizes the preponderance of the Test Execution Time metric. This measurement is frequently employed to assess the efficiency of the test itself. It also serves as an indicator of how effectively the chosen approach (like various Machine Learning models) is functioning. Furthermore, the metrics like the number of player states and the number of objects are used for evaluating agent behaviours, exploration effectiveness, and visual recognition capabilities. As seen in Figure 4.10, the number of sounds tested is one of the least common metrics. This is indicative of the fact that sound testing is not a frequently explored area in game testing.

Test execution time

Time required to execute an entire test or a subpart of it. In [107], it is time to process and analyse a gameplay video. To have a more precise metric, execution time is evaluated for each phase of the test [68].

Number of player states

For platformers, the player character's position and velocity are part of the player's state. In other games, the number of keys obtained and the set of opened doors would be player-state variables since they will alter during a single playthrough. The placement of the walls and doors is part of the problem instance because it differs between level designs but is static within a playthrough. Finally, the rules that prevent the player from walking through walls are part of the fundamental rules, which are the same for each level design [112]. In [87], the number of unique states reached is used. For regression testing the number of visited states is compared with the previous version of the game [80]. In [84] player states are only related to player location because that test aimed to explore the playable map. Graph-based coverage such as Vertex (or node) coverage can be used as a metric for navigation or explorative tests. However, not all of them would make sense (e.g. trying to cover all non-cycling paths would quickly become unfeasible) [113]. Also in [71] and [104] the number of paths (that can be attributed to the number of player states) is used to evaluate the test coverage of a specific level.

In [88], [69] the nav-mesh goals and unique positions found in the tests can be considered player states.

In [106], game states are considered as different steps in a game such as a start, play, pause, win and lose. During the gameplay, the player changes game states when some conditions are met performing state transitions.

In [110], instead of the number of different states, the size of the files containing the state information is evaluated.

To summarize, the player state is characterized by each variable affecting specific goals' reachability. This metric can be useful for explorative agent-based tests.

Num of bugs found

The number of already existing (known or unknown) or injected bugs found. This metric is more related to the quality of the test rather than the quality of the code that has been tested. However, especially in agent-driven tests, this metric can be helpful to compare different versions of the tested game with or without a specific bug. If the test is not able to recognize that specific bug the agent may not work correctly and some kind of adjustments are required. Regression bugs that were triggered during regression testing can also be evaluated in this coverage in addition to known, unknown and injected bugs [80]. In [69] the number of bugs found by the agents during training are evaluated, these bugs are not necessarily identified as game-related issues. On the other hand, the bugs detected by their second algorithm are identified as game issues.

Num of object

The interactable object coverage refers to the ratio of triggered interactable objects to the total number of interactable objects in a given context [59]. The count of virtual objects present either within a scene or the entire project is denoted by [59]. The number of buttons (interactable), doors and obstacles in a single level [41], [68]. The quantity of detected objects is measured as indicated by [84]. These objects can encompass UI elements such as buttons or images [100], in this case, this metric is used to evaluate the quality of the test. In the case of [99] the number of objects are considered as the number of different weapons tested. In [79] objects counted are walls, collectables and enemies in the tested level.

Num of in-game variable

In [46] this metric is used for agents used in a racing game. In this case, the variables include parameters such as average speed, number of collisions, final score, mid-air time, number of crashes, nearest car and lap time. These statistics differ depending on the agents' playstyle and can be useful for game balance activities. In [107] continuously track game statistics like health level, stamina and weapon used. They are used to check if the game reacts correctly with visual effects that the game should enable when these statistics reach certain thresholds like flashing icons, life bar colours and other graphical effects. In [91] in-game variables like damage dealt, weapons equipped, turns taken, cards played etc are used to evaluate the deck's strength. In [79] loses, wall collisions, collectable items collected and close enemies are used as game statistics. In [71] game events relate to the number of monsters, treasures, doors and deaths in the level played.

Num of image checked

The number of template images that were created and used for each game [76] [62]. The number of UI images collected and detected [100], [62] and [89]. In this case, this metric is used to evaluate the quality of the test. In [86], the number of the images checked is evaluated and compared with the number in which these images contain graphical bugs. In [62] and [90] to check widgets or GUI states Widget Similarity and State Similarity are evaluated. In [86] and [62] test quality is evaluated using these metrics on the test results:

- Precision
  - "Precision presents the proportion of correctly classified screenshots as UI glitch among all screenshots predicted as UI glitch"
- Recall
  - "Recall indicates the proportion of correctly classified screenshots as UI glitches among all screenshots that have UI display issues"
- Accuracy
  - "Accuracy reflects the trained model's ability to make correct decisions on the test set. The more correct samples the model predicts, the higher accuracy it will output"

- F1-score
  - "F1-score is calculated from the precision and recall of the test and it reflects the harmonic mean of precision and recall. The highest possible value of an F1-score is 1 which indicates perfect precision and recall, and the lowest possible value is 0 if either precision or recall is zero"
- Mean average precision

Memory usage

the amount of memory consumed by the game during performance tests, typically measured in megabytes (MB). This metric provides insights into the game's memory footprint and helps evaluate its efficiency in managing system resources.

FPS value

FPS (Frames Per Second) value is a performance metric that measures the number of frames rendered per second in a game. It is commonly evaluated using three key indicators: minimum, average, and maximum FPS.

Num of steps

The number of game steps performed. Notice that steps can be mandatory or optional, in [76] there is a clear distinction between optional and mandatory steps in their metrics. In [74] [71] [95] this metric is defined as test sequence length or trajectory, which is the number of player actions performed in a test run (these actions do not necessarily have to be useful to the progress of the game).

Code coverage

lines of code (LoC) that are executed during the running of the test suite. It provides an indication of the proportion of the codebase that is exercised by the tests [87].

Lines of code (LoC) written for testing

Line of code written in the test script [76] or Line of code of the project [59] [62] [96] [27]. In [27] the complexity of the code is also evaluated with the LoC using the cyclomatic number.

Num of functions

In [39] the number of logic and the number of functions are used. In [27] the functions are called instructions, also classified using the cyclomatic complexity.

CPU usage

The percentage of the CPU's computational capacity utilized during performance tests. It measures the amount of processing power consumed by the game during gameplay scenarios.

Difficulty

In [55] a difficulty score is evaluated for each level using their tool. In [91] empirical and simulated difficulty are evaluated and compared.

Num of graph edges

The number of graph edges that connect a player state to another (see num of player state definition) [112]. In [103] and [27] graph edges are called transitions between player states, and the transition coverage is evaluated.

Level size

Room or level size in terms of tiles or units[41]. In [88] the size of the levels implemented in Unreal Engine 5 is evaluated in meters.

Win rate

In [91] (called success rate in [95]) the number and percentage of victories are evaluated to test the strength of the card decks. In [104] the possibility to complete a procedurally generated level is tested, to do so the win rate of the bots that play the level is considered.

Agent training time

Time spent to train agents, in [40] training time is measured in minutes and the number of rounds while in [88] this time is measured in timesteps.

Num of persona

Time spent to train agents, in [40] training time is measured in minutes and the number of rounds while in [88] this time is measured in timesteps.

Num of levels

The number of levels (or puzzles) is used as a metric in [44]. In [34] the number of isolated platforms that are used to test specific features of the game are considered.

Num of clients

The number of bots connected in a single server [65] [104]. It represents the number of simulated clients or automated agents interacting with the game server during testing or gameplay scenarios.

Num of classes

The number of different classes refers to the count of distinct classes present in a project [96] [39]. It quantifies the number of unique classes or object-oriented structures within the project's codebase.

Sampling rate and frequency

Audio sampling rate and bin frequency used to compare FFT transform [12]. In addition to audio signals, it is worth noting that other types of signals, such as EEG signals, also require a specific sampling rate. The sampling rate of EEG signals plays a crucial role in evaluating the accuracy and limitations of the test [48].

Num of input paths

Number of execution paths through the input-handling code identified [87]. To categorize better the input paths these factors can be added to this metric:

- if the path contains a button or a key
- if the path contains an axis
- input device (if keyboard or controller)

In [99] the number of action sequences tested is used as a metric.

Mutations

Changes in terms of location or logic changes in a level. This is useful to test the robustness of the level and the test that uses dynamic testing (in the case of [41]).

Change in terms of the card changed in a specific deck [91].

Time required to master a task

The time it takes for the agent to master a task can be used as an indicator of how difficult the game would be for a human player, the time required can be evaluated also as the number of frames executed [114]. In [98] is considered the number of attempts to win and the win rate per run.

Network activity

Network activity refers to the monitored data traffic, encompassing both incoming (ingress) and outgoing (egress) communication, that is initiated by the game [35]. This metric is particularly valuable in mobile online games, especially considering that data usage may be constrained by internet service providers.

App startup time

Startup time refers to the measurement of the duration it takes for a game to initialize and become fully operational [35]. It evaluates the impact and efficiency of the game's startup process by quantifying the time required for the game to start.

Num of decision

count of decisions made to to achieve a specific goal, such as solving a maze [33]. This metric quantifies the number of choices or actions taken by an agent or player to navigate and successfully complete the given task. It provides insights into the decision-making complexity and strategy employed during the gameplay.

num of action

In [99] the number of different actions is used as metrics. These actions can include various player interactions such as moving, attacking, bashing, character selection, and combining movements with bashing.

confusion matrix

Confusion matrixes are used in [109] to evaluate the emotions predicted against the actual emotions in a gameplay scenario.

Image recognition time

This metric is used to evaluate the efficiency of UI feature recognition. This time depends on the hardware but also the resolution of the game.

Num of behaviour tree

The study conducted in [78] examines the impact of code issues, imposed metrics, and tournament evaluation on the number of distinct behaviour tree types generated. The analysis sheds light on how these factors influenced the initial set of proposed behaviour trees. The different types of Behaviour trees (BTs) are:

- Number of BTs created
- Number of unclassified BTs
- Number of BTs that crashed or encountered other issues
- Number of BTs rejected by the tournament evaluation

Num of hit

The number of actions that hit an operable object. It is evaluated as the number of total hits, number of unique hits, hit ratio and number of valid operations in a specific time window [42].

Num of observations

Observations are specific player feedback (in terms of actions, sentences, and physical and verbal reactions such as the player that is yelling at because it is too hard or yawning in front of boring moments). Observations are divided into 3 primary categories, these categories are further subdivided into other micro categories as follows [10]:

- Design
  - Game design

  - Visual design
  - Audio design
- Product
  - Quality
  - Time-related
- Context-sensitive
  - Player specific
  - Multiplayer
  - Livestream-based

Num of attempts

The count of attempted subgoals before successfully completing the level's testing task [41].

Num of speech

the count of speech instances or samples that have been tested, as part of a specific evaluation or analysis [12].

Num of animations

the count of animations that have been tested as part of a specific evaluation or analysis [12].

Num of sounds

the count of sound assets that have been tested as part of a specific evaluation or analysis [12].

Mutant score

In [103] mutants are created in the Lab Recruits application in which the association between buttons and doors is changed, i.e., a link between a button and a door is removed. The mutation score is computed as the ratio of killed mutants to the total number of mutants generated

Num of Belief desire intention (BDI)

The Number of backtracking performed in a level. i.e. after entering a second room, the player needs to back to the first room to complete a required task to complete the second room.

Milliseconds (ms) to render frames

Represents the milliseconds needed to render four frames in response to an agent's action. [63]

Num Source files

the count of distinct source code files present in a project [59]. It provides insights into the size and complexity of the project's code structure.

4.5.4 RQ2.1 - Tools and approaches

From the literature, 129 tools/approaches have been discovered which 50 are free, specific to video games and can discover at least one type of bug like [135], [136], [137], [138] and much more.

To ensure the quality of a game, a large number of generic GUI testing tools are commonly utilized at the system testing level. In contrast, other testing frameworks that check into the majority of the game's logic tend to be game-specific.

In this section only these 50 tools/approaches are described, the rest of the tools are considered for the bug coverage statistics.

RiverGame

The tool also addresses the challenge of input priorities and test scheduling, facilitating dynamic testing efforts. RiverGame's architecture is engine-independent and platform-agnostic, making it compatible across different devices and operating systems. This flexibility extends to programming languages as well, enabling non-technical stakeholders to write tests and expected behaviours for the game under scrutiny.

A standout feature is its sound processing techniques that automatically evaluate in-game sounds. In addition, the tool evaluates statistical metrics registered by the end user to assess performance aspects.

RiverGame improves the performance of evaluation processes by utilizing efficient methods for pose recognition and object detection.

RiverGame implements the testing features as follows [6]:

- Behavior-Driven Development (BDD)
  - a methodology that fosters collaboration among stakeholders and allows both technical and non-technical individuals to use natural language to describe the purpose and expected outcomes of software features. This approach, aided by the Behave library, enhances test reusability and accessibility, particularly for those without programming experience.
- computer vision analysis
  - RiverGame employs a combination of methods to test expected behaviours. It leverages external technologies like Tesseract OCR [139] from OpenCV [140] for text recognition and utilizes techniques such as template matching or scene segmentation for the object or feature recognition in images. Additionally, it is equipped to detect environmental changes through specific object class training and motion analysis of objects in reconstructed 3D space from 2D image frames. This setup allows RiverGame to cover a wide array of testing requirements.
- Animation testing
  - RiverGame tackles issues such as character immobility or inconsistent movement direction. The process is twofold. Firstly, it identifies a set of fixed points in the scene, acting as reference points. Secondly, it utilizes

MoveNet to extract the skeleton of the character in each test frame. The character's movement trajectory relative to the fixed point is then compared to the expected trajectory defined by the user. This method is versatile, applicable not only to human figures but also to other entities like animals, vehicles, and buildings that can be represented with a skeletal structure.

- Sound testing
  - RiverGame employs automated testing methodologies to verify in-game sound accuracy. It tests background sounds using the Librosa library [14] and the FFT transform to convert and compare the spectrum of the sounds. For character dialogues, it utilizes Facebook's wav2vec [13] 2.0 model to convert the voice to text, which is then compared with the original text for similarity.

ivXR

The ivXR framework, largely written in Java, is an advanced game testing platform that employs specialized agents for automating tests in both 2D and 3D gaming environments [141]. It goes beyond traditional automated testing techniques, by addressing the dynamic and non-deterministic nature of modern games through goal-driven, adaptive, and reasoning agent-based testing. This is achieved using aplib, a Java library module within ivXR, that provides a Domain Specific Language (DSL)-like fluency while retaining the broad features and robust tools of Java for the creation of intelligent test agents [50].

ivXR is inspired by the Belief-Desire-Intent (BDI) concept, where agents possess a belief that represents their understanding of their current environment, and their own goals symbolizing their desire.

It introduces the concept of 'goal structures' and 'tactics'. A 'goal structure' is a tree that contains basic goals as leaves and goal-combinators as nodes, used to decompose complex goals into simpler subgoals. A 'tactic' is a method to hierarchically combine basic actions using tactic-combinators. This enables the formulation of complicated testing tasks purely at the goal level, without specifying the required tactics, significantly increasing the ease of testing [27].

The ivXR framework, with its adaptable and extensible architecture, eases the integration of additional testing tools [27], thereby amplifying its capabilities in game testing. This feature highlights the framework's inherent versatility and

promising prospects in the expansive domain of automated game testing.

To demonstrate its capabilities, the iv4XR project provides a demo application, LabRecruits, in which it executes various test cases.

Wuji

Wuji is an agent-based testing tool which explores game space [40].

Iftikhar et al.

Automated model based testing tool for platform games that uses UML state machine.

Ariyurek et al.

Defines both synthetic agents, trained in a completely automated manner, and human-like agents, trained on trajectories used by human testers using RL and MCTS [63]. To test a game the focus of these agents, instead of maximizing the game score, is finding defects [74].

Bergdahl et al.

An approach to augment existing manually written test scripts with reinforcement learning [59] [114].

ICARUS

This tool consists in an agent that play the game detecting and reporting crash, stuck bugs and performance bugs. It can also be used for aesthetic bugs such as graphical, animation, sound, or spelling issues. But in these cases ICARUS does not have any reporting feature and a human that monitor these kind of issues while ICARUS agents play the game is required [60].

Hernández Bécares et al.

Testing tool based on record and play. It can be used to automate beta testing to check changes in the source code and the game's playability [108].

Unity test framework

Unity's testing tool for integration and unit tests [142].

PathOS

PathOS+ is a playtesting tool that uses AI playtesting data to help enhance expert level designer evaluation through an user interface. [93].

PathOS

is the first version of PathOS+, it is a testing tool built to aid the level design process of game developers by simulating the navigation of any 3D game [52] [93].

GameDriver

A freemium testing tool with multiple features like object identification, input recording (record and play testing), multi-platform execution, method execution and continuous delivery integration [49].

Varvaressos et al.

This tool allows run time monitoring of different game states reached by the player. It can also be integrated with bug reporting platform such as Mantis [110].

Paduraru et al.

The framework is capable of automatically generating behaviors for game agents across various difficulty levels, ensuring an adequate level of diversity. By doing so, it enables the creation of a larger number of automated tests, reducing the reliance on human effort for identifying defects in the source code or potential logic exploits [78].

Wu et al.

Regression testing MMORPG,A tool that automatically performs regression testing in video games. It has been used to test different versions of MMORPGs [80].

Unreal testing framework

A functional testing framework is designed to do gameplay-level testing, which works by performing one or more automated tests. Most tests that are written will be functional tests, low-level core or editor tests [38]. Unreal automation system, which is built on top of the testing framework, provides the ability to perform Unit testing, feature testing and content stress testing. It also contains an FBX Test builder to test FBX files and a screenshot comparison tool to check the visual representation of the game [143].

POCO

UI Automation test framework [144]

Crushinator

Crushinator is a framework that provides a game-independent testing tool that implements different testing methods. It incorporates Model-based testing and exploratory testing. It can also test server limits, defects and performances by simulating large numbers of virtual clients [96].

MAuto

MAuto records the user actions in the game and replays the tests on any Android device. MAuto uses image recognition, through AKAZE features, to record the test cases and the Appium framework to replay the user actions automatically [101].

Song

This framework consists in an automatic game-testing system that blends an adversarial inverse reinforcement learning algorithm with multi-objective evolutionary optimization. The system is designed to maintain the quality of various games across the market, requiring minimal manual adjustments for each game [54].

Albaghajati et al.

This approach uses coloured Petri nets representations of the software workflow to automatically test a game [58].

Dandey et al.

An automated testing strategy that performs simulation-based testing using record and replay testing strategy [39].

Paduraru et al.

This tool uses computer vision techniques to detect game states and expected behaviours [94].

García-Sánchez et al.

A playtesting approach that improves and accelerates the balancing process of card games (Hearthstone [145]). It uses an evolutionary algorithm to analyse some possible card combinations in decks [91].

AirTest

AirTest is an E2E UI testing tool for mobile and Windows videogames [146].

Inspector: Pixel-based agent

Using only screenshots/pixels as input for automated game testing and build a general game testing agent, Inspector, that can be easily applied to different games without deep integration with games [84].

Nelson and Yasunobu Think-aloud testing

This approach discovers quality and design issues by observing the live streams. Observations are acquired by paying particular attention to the player's behaviour outside of the game including things such as yawing, snacking or chatting with the viewers and then classified using predefined categories (see Observations metrics) [10].

Kljajic et al. client-server game testing

A testing approach to test the connection in a client-server multiplayer game [73].

WHENet

A tool that gives real-time head pose estimation in a video [109].

Ye et al.

GUI testing tool that detects GUI images and widgets in mobile games [89].

Kwon et al.

A framework that can automatically analyse facial expressions in video in a remote environment, the results can be. The expressions analysed can be categorized in emotions during the gameplay [109].

Sestini et al.

The CCPT testing method combines curiosity and imitation learning to train agents to explore certain game levels [69].

Lee et al.

Difficulty prediction tool used for balancing games using moving target acquisition [92].

EEG UX testing for VR Games

It uses Electroencephalography (EEG) signals and brain functional connectivity (FC) to test VR Games. This method objectively measures the overall player experience without the explicit detection of bugs or emotions [48].

AKAZE

AKAZE is a testing approach that records user interactions, exporting them for playback in Appium. After the recording phase, it uses image recognition to identify objects in the screenshots taken during the record and play phase [147].

Yamamoto et al.

The approach chosen in [82] to reduce false positive test results uses OpenCV image recognition algorithms to detect and access the hand-drawn GUI elements on the screen, allowing them to be interacted with from within automated test scripts [82].

GAutomator

GAutomator, or Game Automator, is an open-source testing automation framework specifically designed for mobile games. Mirroring the design of Android's UIAutomator, it can interact with engine components like GameObjects and their components [85].

xUnit

http://xunit.net/ is a community-centred, open-source tool for unit testing within the .NET Framework [148].

KRF Level generator

The tool is designed to automatically generate new building blocks in the style of the original game using Procedural Content Generation (PCG) techniques. These blocks are then assembled to create fresh levels. Additionally, this tool incorporates functionality to ensure that the generated content is playable, maintaining a suitable level of challenge and coherence within the game [55].

DroidGamer

DroidGamer is a GUI traversal-based Android game testing tool that uses deep learning models to recognize interactable GUI widgets. It adopts a GUI model traversal algorithm and a new GUI state equivalence criterion over the widget recognition results of the deep learning models [62].

Marczak et al. Feedback-based gameplay tool

A methodology that has been used to gather data on player behaviour by analysing video and audio streams. This innovative approach involves automated analysis of game interface features, which serve as indicators of player behaviour and significant gameplay events [107].

Altom

Altom is a freemium Test automation framework that supports several testing features like functional testing, performance testing, load testing, usability testing, compatibility testing, security testing, regression testing and a great number of testing tools [45].

Go-Explore

Go-Explore is a tool that broadens the scope of behavioural procedural personas, incorporating the facet of player experience. This tool pioneers the concept of generative agents, which can mimic both the actions and perceived experiences of human players within a gaming setting. Though not explicitly applied to direct game testing, these agents can be integrated with other testing utilities to enhance automated playtesting that closely simulates human behaviour [46].

GLIB

A tool that is based on a code-based data augmentation technique that detects visual bugs [86]. It can detect these graphical glitches: Abnormal colour block, Random noise, Partial repetition, frame overlay, Object missing, Abnormal text, Overexposed, Black border

GoExplore

Efficiently discovers challenging software bugs and comprehensively explores complex environments without the need for human demonstration or knowledge of the game dynamics

Mawhorter et al. Softlock model

Check if a map is playable and if it has some Softlock inside it where the player can be stuck [112].

SPADE

Python scripts used to control a bot in an online game. It has an API and can be used for testing purposes [57] [70].

RELINE

An approach exploiting RL to train agents able to play a given game while trying to load test it with the goal of minimizing its FPS [63].

UnityActionAnalysis

Static analysis and exploration tool to explore game valid states [149].

TestifyInput

Simolator of keyboard and mouse inputs, used in the context of Video Game Test Automation [99].

Interactive design exploration

Game level authoring tool that uses synthetic testers, which enable the control of different play-styles and skill levels. It also implements a graphical interface to help designers to design levels in each part of the game [43].

4.5.5 RQ2.2 - Bugs Discovered

Bugs discovered in the literature are described in this section, and statistics about bugs coverage with the testing tools/approaches found are shown in this graph 4.15

Table 4.6: List of filtered testing tools/approaches Part 1

Tool / approach	Covered Bugs	Mentioned in	Used in
Mawhorter et al. Softlock model	Stuck		
SPADE	Logical, Event, Balance		
Inspector: Pixel-based agent	Graphical, UI, Camera, Logical, Interaction	[88]	
iv4xr	Logical, Interaction, Stuck, Position, Event, Action, Interrupt	[12],[94],[27]	
RELINE	UX, Graphical		
Unity Action Analysis	Stuck, Action		
RiverGame	Audio, Graphical, Performance, Logical	[71]	
TestifyInput	Logical		
Interactive design exploration	Balance, Logical, Stuck		
Unity test framework	Stuck, Logical, Interrupt, Event, Camera, Interaction, Action, Position	[113],[132], [126],[30]	[32],[81], [28],[29]
Wuji	Stuck, Interrupt, Logical	[54],[41],[59],[86], [114],[63],[87],[12], [80],[84],[62],[69], [94],[68],[89],[27], [58],[81],[71]	
MAuto	Graphical, UI, Camera, Logical	[113],[81]	

Logical

Logical bugs often do not break the game like the previous two types of bugs but lead to unexpected results (e.g., errors in score computation). This type of bug is usually caused by incorrect implementation of game logic [40].

Table 4.7: List of filtered testing tools/approaches Part 2

Tool / approach	Covered Bugs	Mentioned in	Used in
GameDriver	Graphical, Logical, UI, Interaction, Event, Position, Interrupt, Stuck, Action	[99],[113],[56]	
ICARUS	Stuck, Event, Logical, Interaction, Interrupt, Performance, Position, Action	[63],[52],[12],[113], [84],[74],[94],[27], [71]	
GoExplore	Stuck, Interaction, Position		
GLIB	Graphical		
Go-Explore	Performance, Interrupt		
Altom	Graphical, Logical, UI, Info, Performance, UX		[45]
Marczak et al.	UX, Graphical, UI		
DroidGamer	Graphical, UI		
Crushinator	Performance, Logical	[87],[66]	
PathOS	Balance, Stuck, Position	[69],[63],[79]	
KRF Level generator	Balance		
POCO	UI	[146],[89]	
xUnit	Logical		[30]
GAutomator	UI, Logical		
Unreal testing framework	Logical, UI, Graphical, Event, Interaction, Interrupt	[126],[56]	
Iftikhar et al.	Stuck, Interrupt, Position, Event, Action, Logical	[76],[87],[40],[63], [103],[113],[84],[74], [69],[68],[27],[37], [81],[71]	

UI

Bugs are related to the user interface of the player such as misrepresented elements, elements allocated in the wrong position, or elements not visible on the screen.

Stuck

Stuck bugs, also called soft lock errors [112], are bugs which freeze the game. Game players face limitations in their capacity to maintain interactions due to potential bugs that arise when a player enters an abnormal state, causing them to become stuck and unable to progress further in the game. [40] [112].

Graphical

In [86] a detailed description is used to detect UI glitches. They categorized these bugs into 8 categories:

- Abnormal colour block
- Random noise
- Partial repetition
- Frame overlay
- Object missing
- Abnormal Text
- Overexposed
- Black border

Event

Event bugs refer to situations where the game does not behave as expected during specific in-game events.

Interrupt / Crash

Leads the entire game to crash and exit. For example, the zero-dividing problem, memory leak issue or recursive function calls will result in a game crash [40]. The game crashes or stops rendering [69].

Gaming balance

Which disrupts the equilibrium of play between human participants and AI players, thereby deviating from the initial intentions of the designers. [40].

Interaction (Collision)

Interaction or Collision Bugs occur when the collision or the interaction of a specific game object is absent or deviates from the expected behaviour. This can manifest as game objects passing through one another, objects not responding to contact, or inconsistent reactions to player-initiated actions.

Action

Action bugs refer to inconsistencies or faults in the expected behaviours of specific in-game actions.

Position

Position Bugs pertain to the inappropriate or erroneous placement of game objects.

Performance

Bugs that affect the performance of the machine that is running the game, performance metrics are often used to evaluate and quantify this type of issue.

User experience

Problems that downgrade the experience of the player in terms of usability [40] and emotions.

Camera

Camera bugs often refer to issues arising with the in-game camera system. These can include the camera getting stuck or moving erratically, displaying an improper view of the game environment, or not following the player as intended.

Info

Info Bugs involve inconsistencies between the information displayed to the user and the actual in-game values. These may lead to incorrect data presentation or misrepresentation of game statistics, causing confusion for the player.

AI

AI or NPC Bugs pertain to issues with the game's artificial intelligence or non-player characters.

Audio

Bugs related to the audio of the game, the most common bugs are [12]:

- Sound not played due to an incorrect trigger
- Sound that is interrupted or covered by different sounds due to in-game events
- Not understandable or wrong speech played

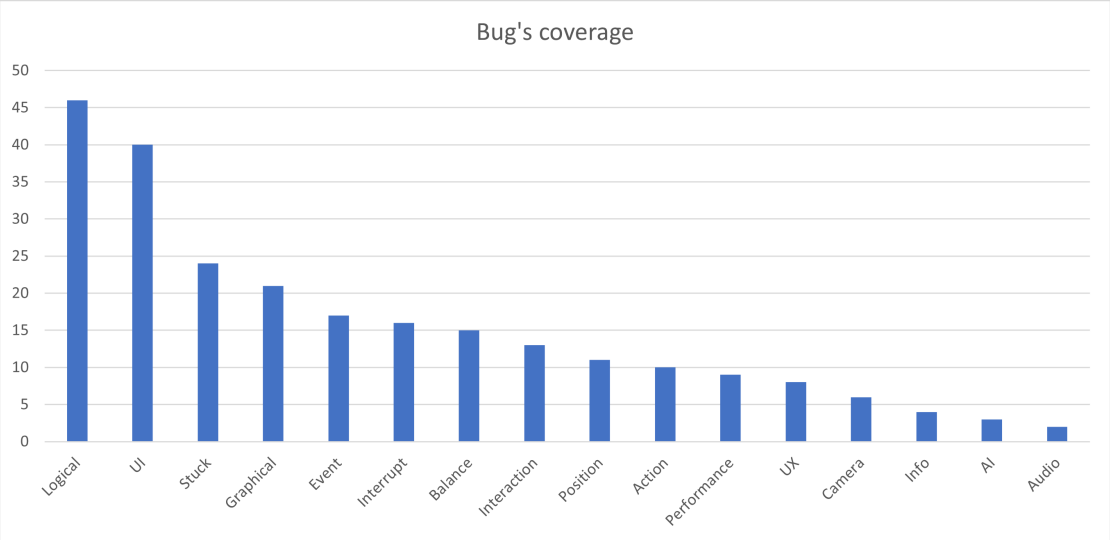

Figure 4.15: Number of tools that discover that specific bug

4.6 Considerations

The review of the literature in this section highlights a growing trend. Over recent years, there has been an escalating interest among researchers and developers in the field of video game testing automation, as evidenced by the increasing number of published papers on this topic (Fig 4.2. This surge not only reflects the dynamic nature of the video game industry but also underscores the pressing need for enhanced tools and methodologies in game testing.

On examining RQ1.1 (testing levels), it is clear that the focus is shifting towards high-level testing systems such as acceptance and system testing. This preference possibly stems from the immense variety of games and corresponding test cases. Creating universally applicable tools or frameworks is a complex task. However, employing high-level testing systems increases the flexibility and adaptability of these tools, allowing them to cater to a broad range of contexts. Moreover, this approach enables a closer emulation of real user interactions, facilitating an assessment of the user's gaming experience.

Considering RQ1.2 (testing goals), it is apparent that the current definitions of testing goals are ambiguous and could benefit from being more precise. This precision could assist developers in maintaining game quality and identifying the most suitable testing methodologies for assessing critical aspects of the game. It

is also evident that functionality testing and regression testing are the dominant themes within the literature. Despite their importance in maintaining core game quality, this focus is insufficient. The success of a game title relies heavily on usability and the overall gaming experience, which currently poses a significant challenge in testing due to the inherent complexity of the games.

In addressing RQ1.3, the literature reveals two primary types of metrics: those focused on the correctness of the game and those centred around the performance of tests and utilized models. The predominance of metrics related to test performance illuminates a crucial divergence between the needs of developers and the areas of focus for researchers. This disconnection is further underscored by [7], which insightfully explores the gap between academia's automated video game testing solutions and the practical necessities of game developers. This discrepancy is also evident in papers that employ agents in exploratory testing, where the primary concern often lies in the quality of the machine learning models used by the agents, rather than the efficacy of game testing.

Reviewing RQ2, it is evident that the existing tools and approaches for game testing automation, while varied and capable of detecting a wide range of bugs, still require further refinement before they can be successfully applied to real-world projects. The more innovative approaches are often tested within reduced and simplified contexts, limiting their potential applications. For these tools to demonstrate their true efficacy and adaptability, they need to be tested within larger and more complex gaming environments.

Finally, two frameworks stood out during this review—RiverGame and iv4xr. RiverGame offers an interesting approach to testing elements closer to the user experience, while iv4xr presents a flexible, modular structure that allows the seamless integration of various testing tools. iv4xr's versatility, in particular, presents opportunities to comprehensively test games using agents and machine learning models, simulating the real-world application of the game.

Figure 4.7: Testing goals taxonomy Part 2

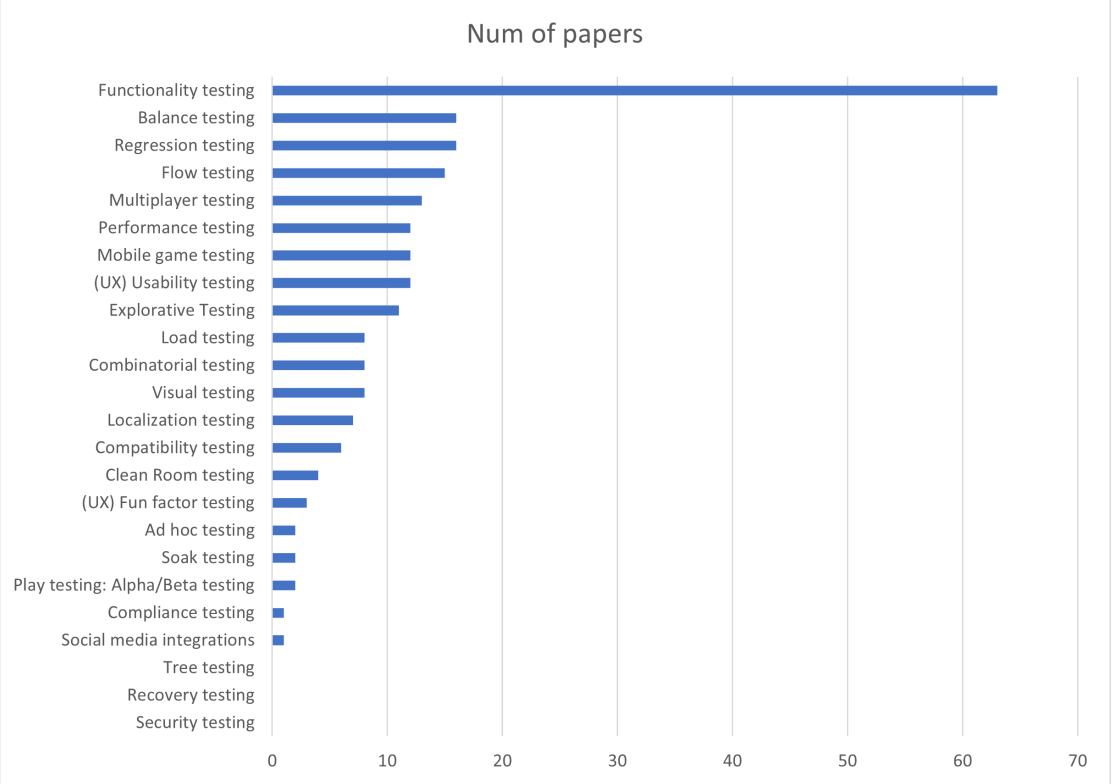

Figure 4.8: Number of papers with the specific testing Goal

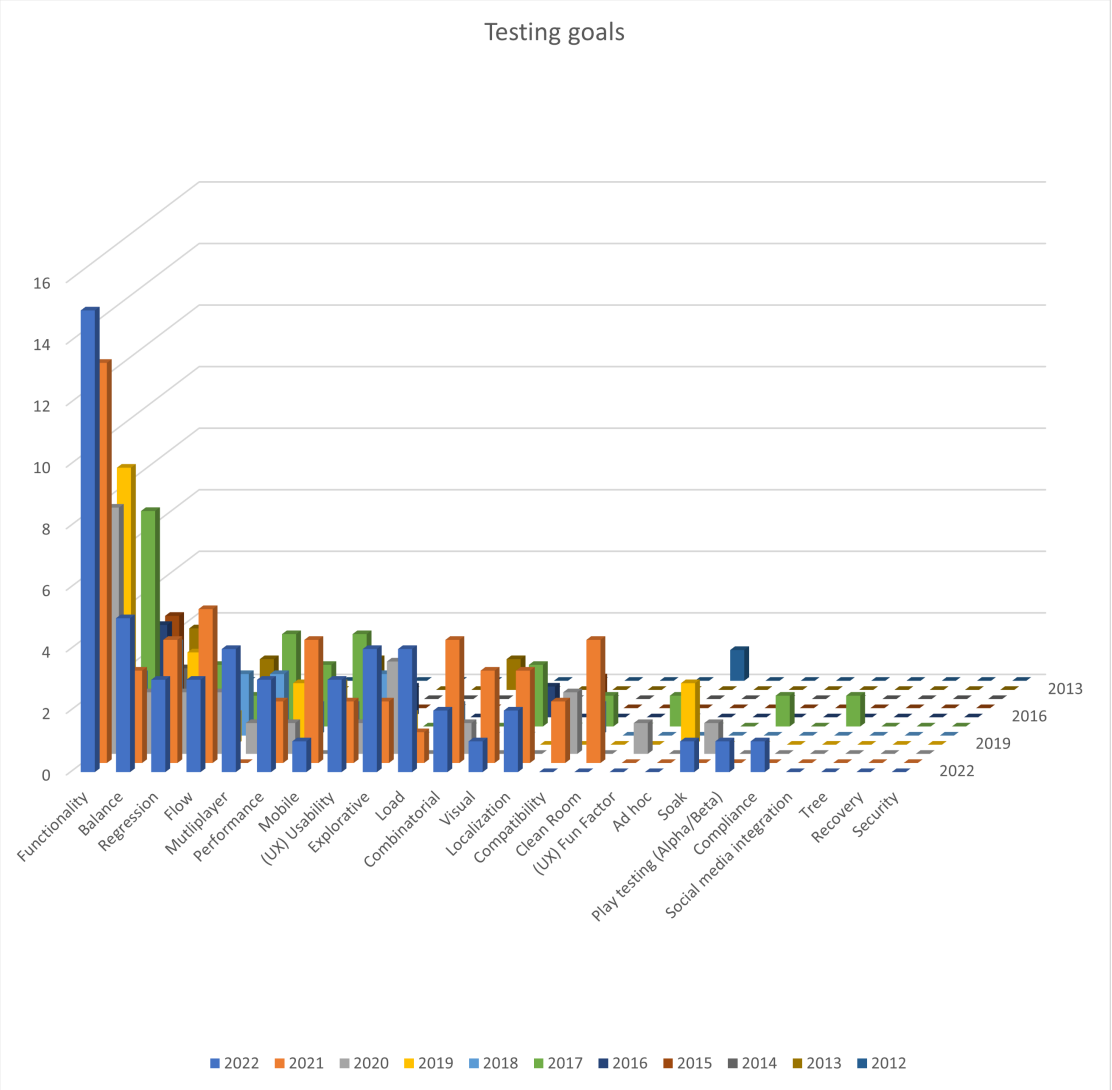

Figure 4.9: Testing goals over the years

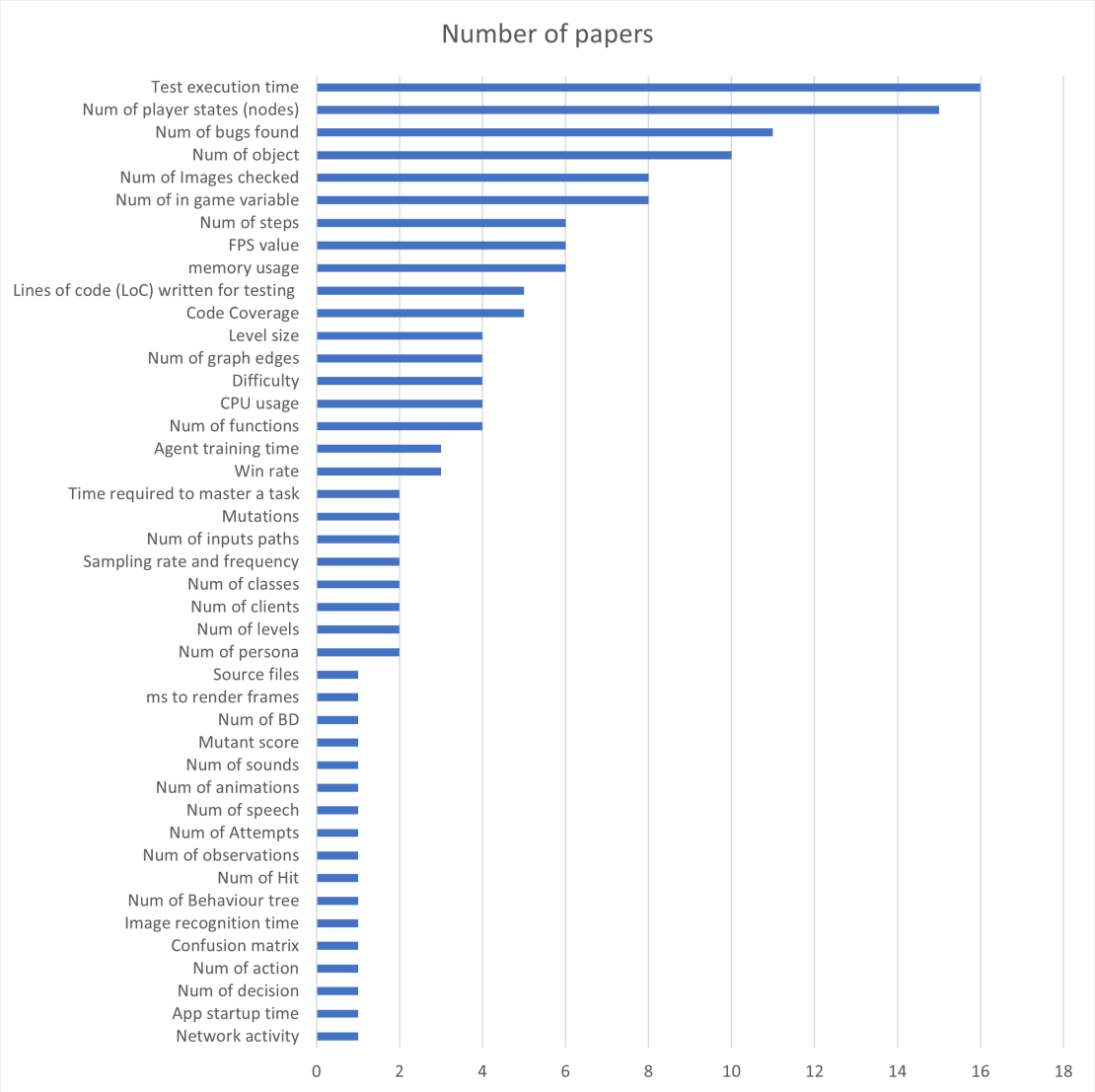

Figure 4.10: Metrics popularity, in this graph there is shown how much a metric has been used

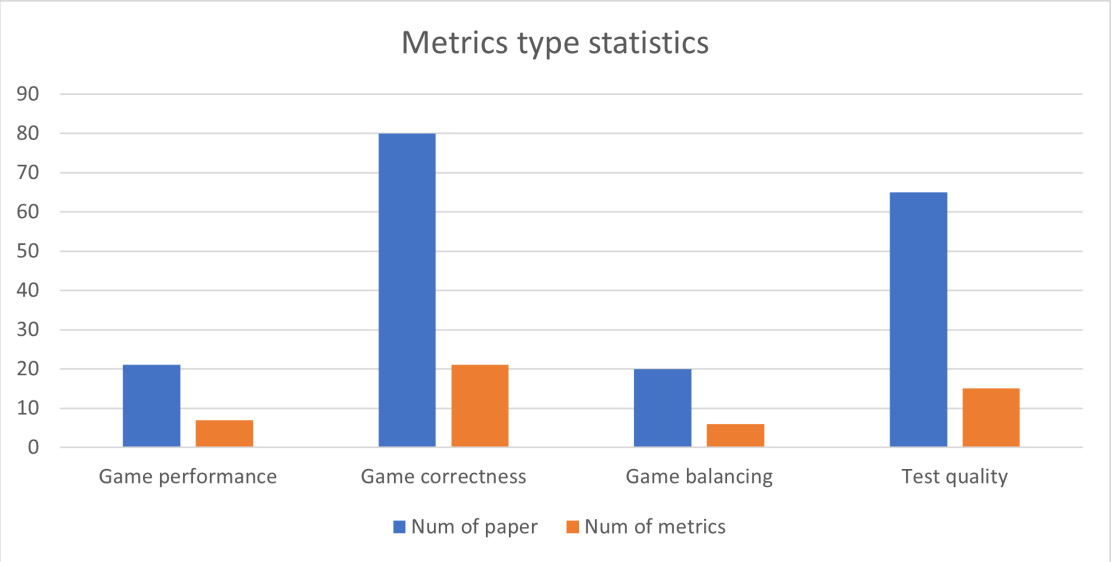

Figure 4.11: Popularity of type of metrics

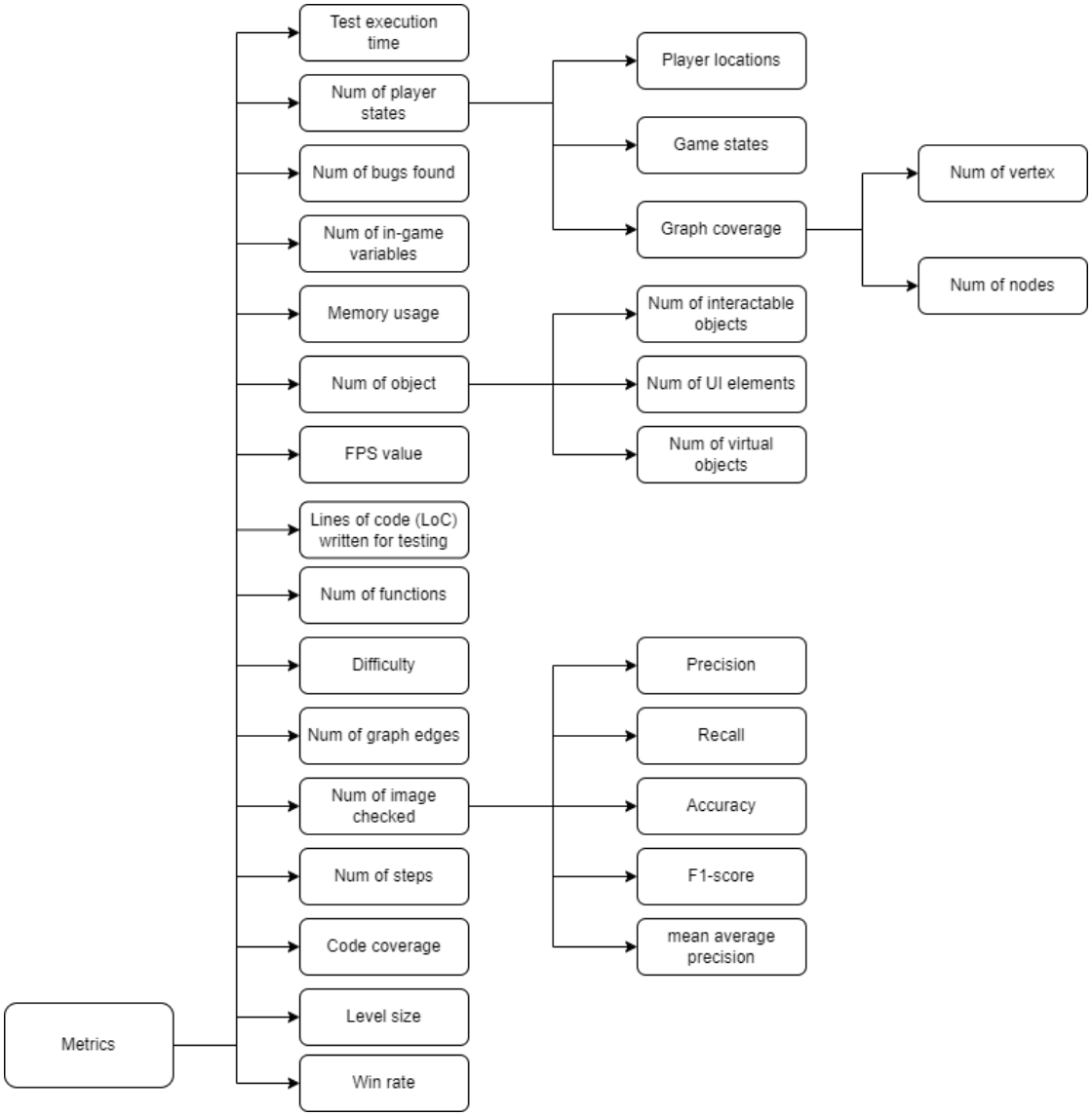

Figure 4.12: Metrics taxonomy part 1

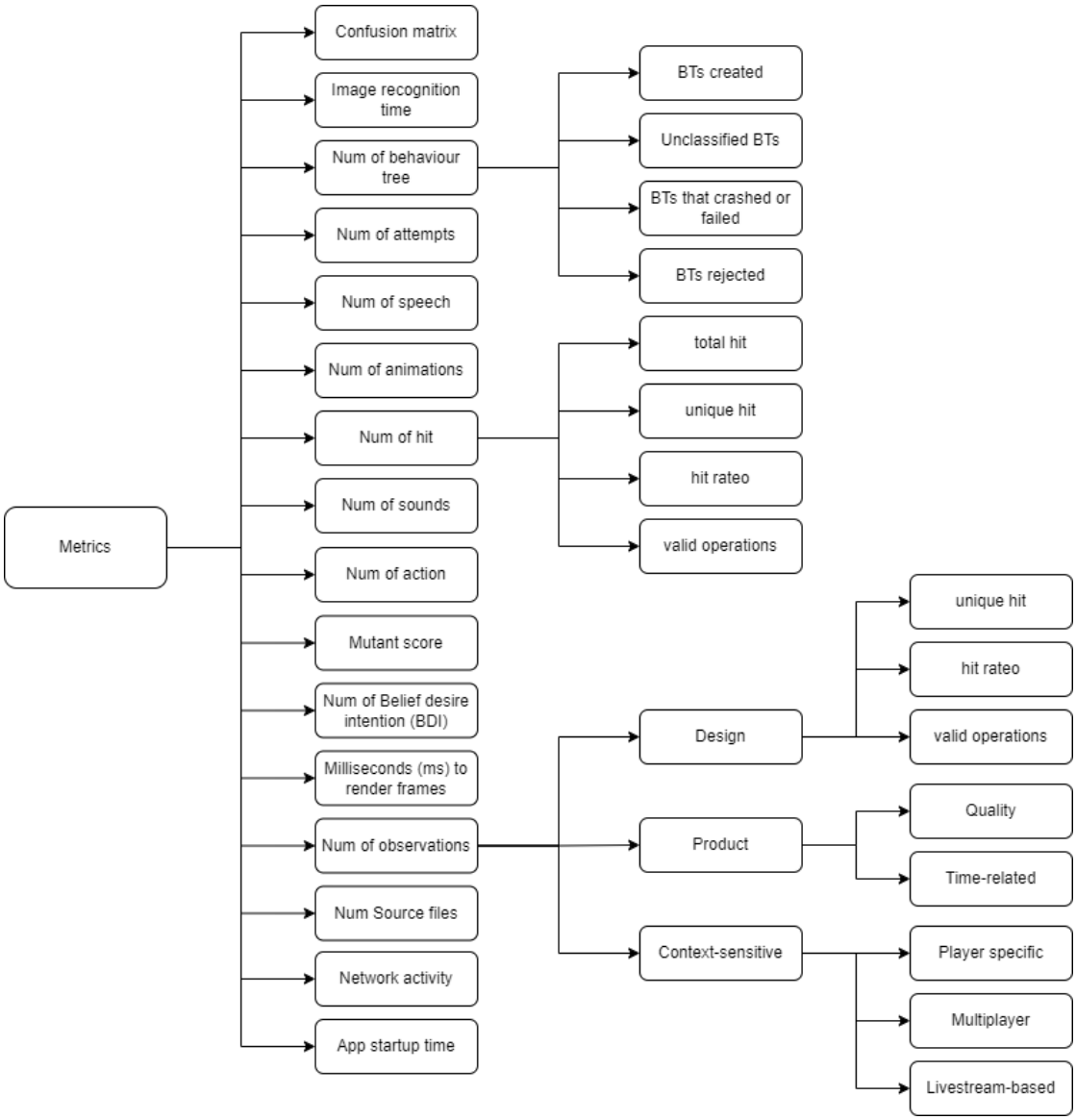

Figure 4.13: Metrics taxonomy part 2

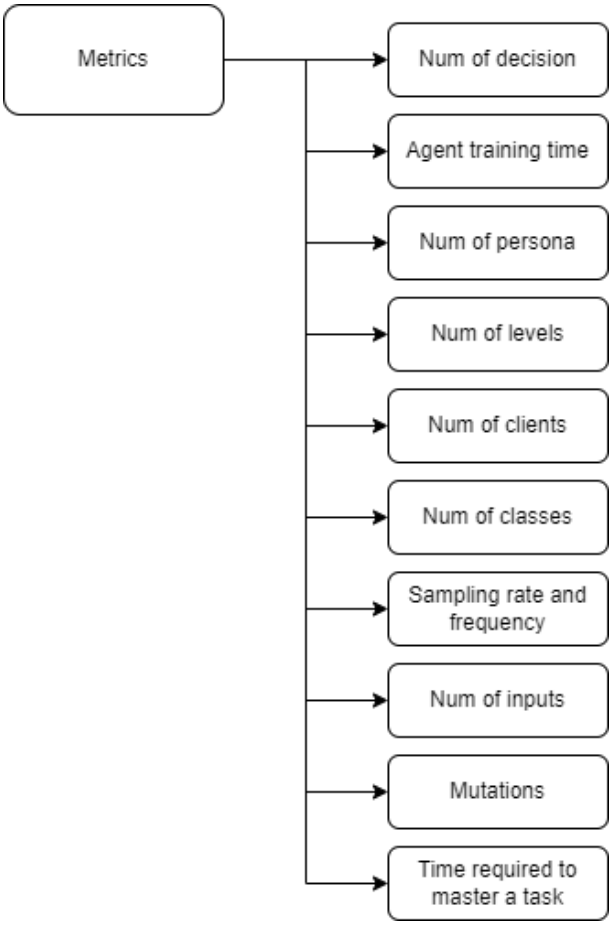

Figure 4.14: Metrics taxonomy part 3

Chapter 5

Sound testing

This chapter exposes the second part of the thesis, wherein the literature review pinpointed a void in the domain of sound testing. In this more practical section of the study, this issue was tackled, providing a preliminary insight into the potential solutions for addressing the most prevalent bugs.

5.1 River Game and iv4XR

RiverGame is an innovative tool designed to facilitate automated testing for game developers. It analyzes various aspects of a game including the rendered output, sound production, entity movements, performance, and statistical data. Notably, it addresses sound testing concerns, often raised by industry partners, which pertain to audio assets not playing correctly or being interrupted unnecessarily.

RiverGame's sound testing framework confirms the accuracy of game-generated sound through a defined methodology. The tool uses two different testing techniques depending on the context:

- For testing background music and effects like engine noise or explosions, the Librosa [14] library is used to convert sound data from the time domain to the frequency domain using the FFT transform. The difference between the two spectrograms - the one that should have played and the one that actually played - is compared. The decision on whether they belong to the same class is evaluated using a statistical T-test.

- For in-game character dialogues, RiverGame employs Natural Language Processing (NLP) to transform the voice into text. It uses Facebook's wav2vec [13] model for this conversion, which leverages Deep Learning and Transformers methods. The similarity check involves calculating the ratio between the original text and the longest common substring (LCS) from the converted voice and the database entry.

Iv4XR is one of the most promising testing frameworks for video games, it is written mostly in Java and it provides modules for agent-based testing, exploratory testing, reinforcement learning-based testing, motion sickness on VR, difficulty estimation and persona agents and emotion prediction-based testing. It also provides a solid demo [5] useful to get started with the framework and try to develop and test new modules. When the tests in the Demo project run, first they open a build version of LabRecruit (that is a simple puzzle game) and then a connection with the interface provided by the game itself is established. Thanks to the connection established, the game creates the level, spawns a defined amount of agents and plays the game controlling each agent in the scene depending on the indication provided by Java tests. The tests pass if specific conditions of the game state are satisfied, these conditions can be checked using special observable objects that monitor in-game variables.

In the following section, the first implementation of sound testing integrated with the iv4xr framework is described.

5.2 Implementation

After conducting an in-depth study of the aforementioned tools and carrying out tests to better understand their functionalities, I gained substantial familiarity with them. Subsequently, the first objective of this practical part was the implementation of a sound recognizer.

5.2.1 Sound recognizer system

The initial challenge in this phase was the resolution of audio compatibility issues. A common audio format was necessary for the sound comparison process. The decision was to use the Waveform Audio File Format (wav) [150]. The chosen specifications included a sample rate of 44100Hz, a 16-bit sample size, mono, and

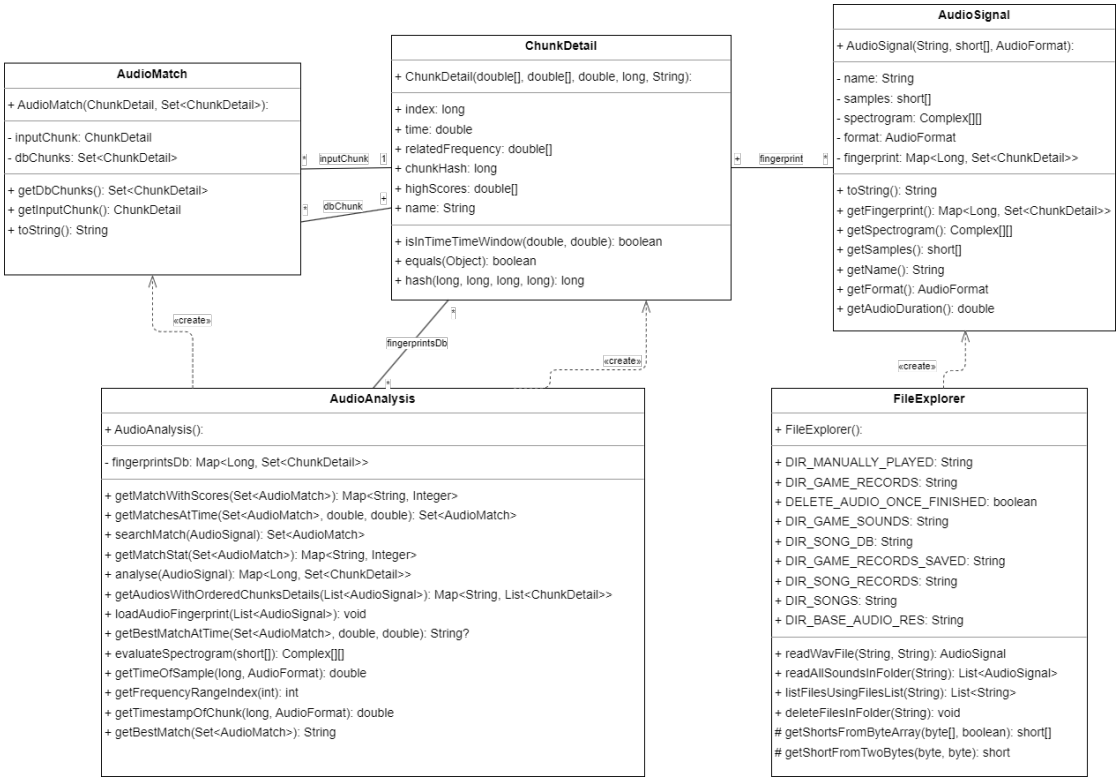

Figure 5.1: Class Diagram of audio analyser

little-endian.

Audio recorder and converter

The sound assets in the game were converted to this chosen format using an online converter [151], thereby creating a database of in-game sounds and eliminating potential compatibility issues.

Next, a Python script was developed, using the Librosa library [14], to record and analyze the audio stream from the Realtek's Stereo Mix while the game was running on a Windows 11 system. The recorded audio was saved in the chosen format. For the purpose of this thesis, a recording duration of 60 seconds was found to be sufficient for completing each chosen levels in the LabRecruits demo.

Audio analyser

Once validated the audio recorder, the next phase involved the implementation of the audio analyser and recognizer. The corresponding class diagram is displayed in Fig 5.1.

The process starts with the reading of a wav file, leading to the creation of an AudioSignal object. This object is constructed by inputting the audio samples, represented as an array of shorts (16-bit sample size). The constructor of the AudioSignal then proceeds to evaluate the Spectrogram of the audio input, through the Fast Fourier Transform (FFT), and a Map, which are utilized in the recognition algorithm. It also stores the 'shorts' array that contains the samples of the audio in the time domain, along with the format, which is intended for debugging purposes.

The FFT [152] is an algorithmic technique employed for the efficient computation of the Discrete Fourier Transform (DFT) and its inverse. The DFT is defined for a sequence of N complex numbers $x_0, x_1, ..., x_{N-1}$ as follows:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N}$$

where for each $k \in \{0, 1, ..., N-1\}$, X_k is a complex number.

FFT is used to translate a signal from its time domain to its frequency domain. In the context of audio analysis, this translation allows for the visualization and understanding of the different frequencies that constitute the sound at any given time.

Although the calculation of DFT directly from the formula is straightforward, it requires N^2 complex multiplications, making it computationally expensive for large values of N. The FFT, however, exploits the symmetrical properties of the twiddle factors $e^{-2\pi i k n/N}$ to reduce the computational complexity to $O(N \log N)$, making it a much more efficient method for calculating the DFT, especially for large sequences of data.

Thus, the FFT is utilized to convert a signal from its original time domain to the frequency domain, isolating different frequencies present in a sound at any given moment.

Nonetheless, an individual FFT provides only a snapshot of the frequencies present at one specific time. A more complete view of how the frequencies in a

sound evolve over time requires multiple FFTs at regular intervals (time chunks).

The product of this process is a series of FFTs, each providing information about the frequencies in a different temporal section of the sound. When these FFTs are graphically represented next to each other, with time on the x-axis, frequency on the y-axis, and the intensity of a particular frequency at a certain time shown through color or brightness, a spectrogram is created.

Hence, a spectrogram can be considered a sequence of FFTs conducted on successive temporal sections of the sound. The size of these segments can be configured in the AudioConfig class, providing flexibility in the frequency analysis's granularity over time (useful for short sounds present in the assets of the game).

Upon calculation of the spectrogram, the subsequent step is to evaluate the Map that holds the song's fingerprint. This Map, designed such that the keys correspond to the numbers that distinctly represent the stored chunks, has keys that correspond to the numbers uniquely representing these stored chunks. Each chunk's value holds information about the song's name and the chunk's timestamp. Each chunk's identifier is determined through a hash function, which takes four numeric values (p1, p2, p3, p4) as input along with a configurable fuzz factor. The numeric values taken as input of the hash function are the frequencies related to the maximum magnitude peak within four distinct frequency ranges. This is how the unique identifier for each audio 'chunk' is evaluated:

$$
\begin{aligned}
&(p_4 - (\mathrm{mod}(p_4, \mathrm{FUZ\_FACTOR}))) \times 10^8 \\
&+ (p_3 - (\mathrm{mod}(p_3, \mathrm{FUZ\_FACTOR}))) \times 10^5 \\
&+ (p_2 - (\mathrm{mod}(p_2, \mathrm{FUZ\_FACTOR}))) \times 10^2 \\
&+ (p_1 - (\mathrm{mod}(p_1, \mathrm{FUZ\_FACTOR})))
\end{aligned}
$$

Audio matcher

Before the execution of tests, it is necessary to load the game's sounds into a Map named "fingerprintDb". To do so, a static method is provided by AudioAnalyser class, which also contains the previously mentioned fingerprintDb map. The structure of this map is similar to the fingerprint map present in AudioSignal but it contains the chunks of all the sounds loaded. Consequently, a singular identifier used as a Map key might contain chunks from different songs.

With a particular gameplay record, the method "getMatchesAtTime" can generate data about the sounds present in the specified time window by creating a Set of AudioMatches. A match is recognized when the identifier of a specific Chunk of the analyzed record is within the time window and its identifier exists in the fingerprintDb. Inside the AudioMatches are stored the ChunkDetail of the record's chunk that instigated the match and all the Chunks bearing the same identifier.

Statistics can be extracted, using the appropriate AudioAnalysis method, from a set of AudioMatches and used to evaluate the best match and the presence of a specific sound. These statistics contain the number of matched chunks for each sound in the database. The higher this number, the higher the probability that the sound was played in that time window.

To discern the optimal configuration, a receiver operating characteristic (ROC) curve was plotted (5.2). Each line within the ROC represents a specific configuration of chunk size and fuzz factor, while each point on the line indicates the coordinates of the true positive ratio (TPR) and false positive ratio (FPR) with varying matching thresholds. A sound is only recognized as present within the time window if the number of matches are greater than the matching threshold; otherwise, the sound is considered absent. The most effective configurations were found to be as follows: Without any background:

- Threshold = 3, Fuz Factor = 2, Chunk Size = 512 samples
- Threshold = 11, Fuz Factor = 3, Chunk Size = 512 samples
- Threshold = 2, Fuz Factor = 2, Chunk Size = 1024 samples

with the background music's volume deliberately reduced to 30% using the volume mixer 5.3: Threshold = 13, Fuz factor = 3, Chunk size = 512

5.2.2 Integration with iv4xr

The sound recognition system, capable of verifying the presence or absence of a sound within a specified time window of recorded gameplay, can be employed to ascertain if a specific event has activated its associated sound. This section seeks to identify the time windows in which a sound should be audible. The iv4xr's demo proves to be a perfect fit for this purpose, mirroring real-life scenarios in video games where LabRecruits levels are played by agents, devoid of temporal information regarding the occurrence of specific events.

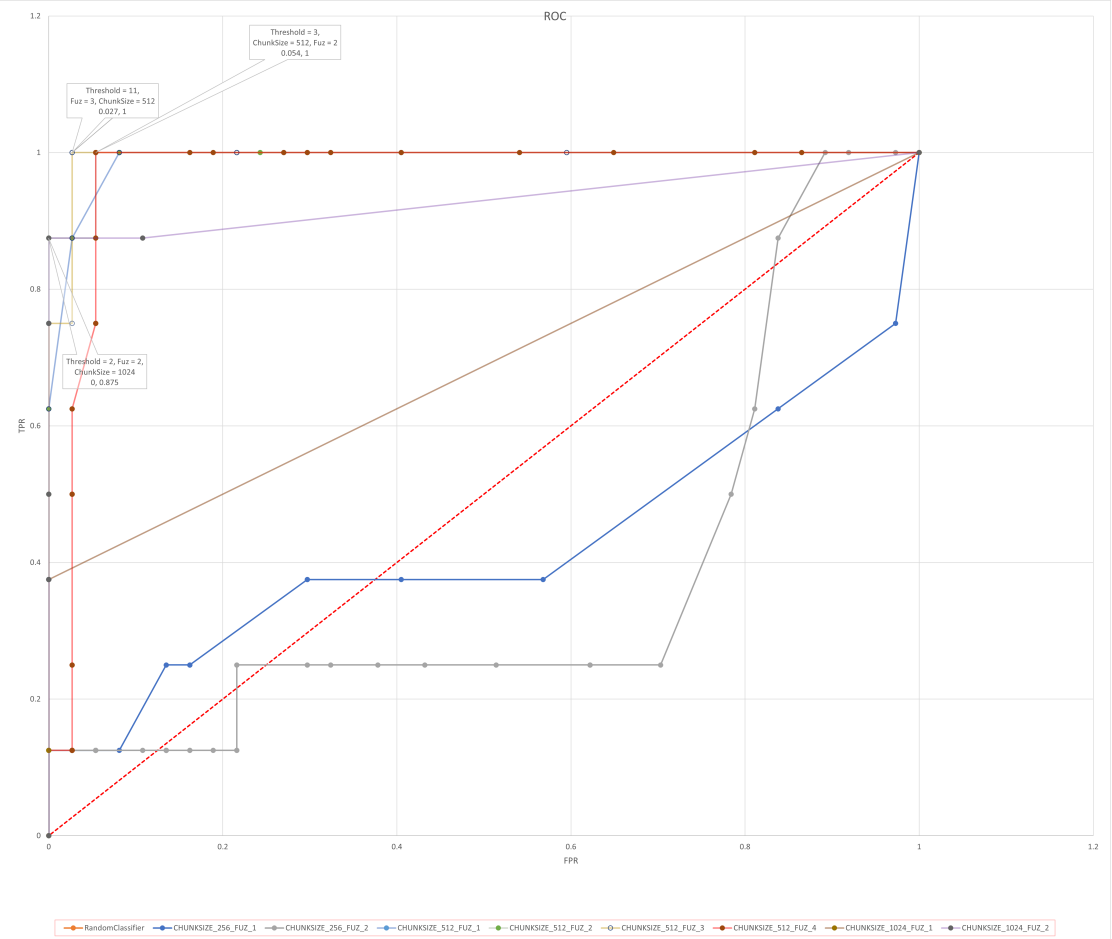

Figure 5.2: ROC curve without noises

Initially, the game and the audio recording system are launched simultaneously. To keep track of events like button presses, monster attacks, or fire damage, custom observers were implemented for agents and environmental variables.

With each event observed, the details are recorded and stored in the event list.. Upon the level's completion and the subsequent saving of the gameplay record, the recorded gameplay undergoes analysis. The system then verifies the presence of sounds corresponding to the events observed during gameplay.

Using this approach (the flow of a test is described in the sequence diagram in fig 5.4) three different levels were tested, with each test checking at least three events per level.

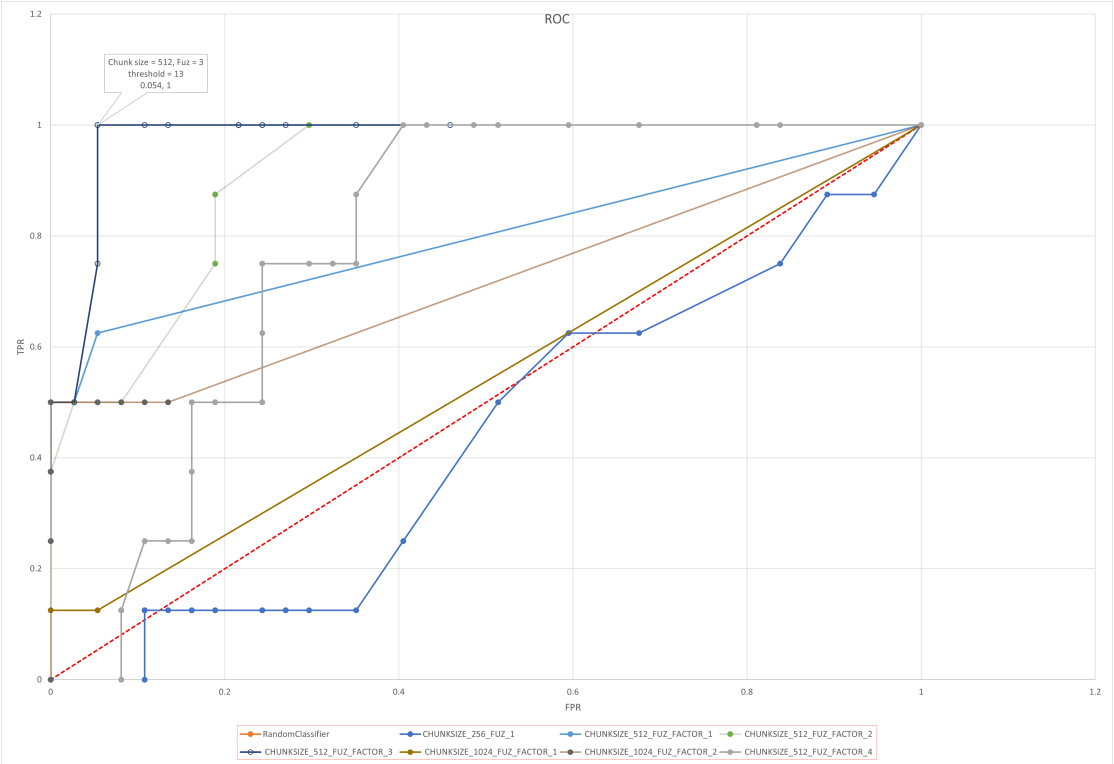

Figure 5.3: ROC curve with background music

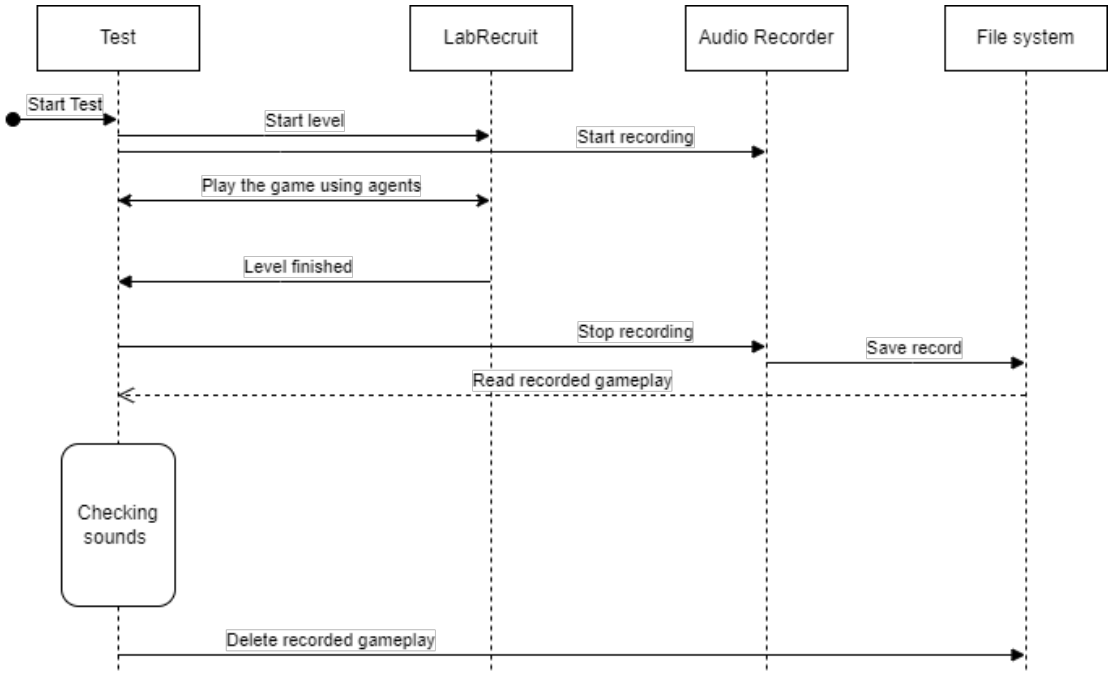

Figure 5.4: Test execution sequence diagram

Chapter 6

Future work

This thesis has proposed a comprehensive approach to game testing automation, particularly focusing on audio testing. The following are potential areas for further investigation and development:

6.1 Literature review enhancements

To refine the outcomes of the literature review and achieve more precise results, future studies could implement known quality assessment methods for the examined papers. Furthermore, a snowballing technique, which involves tracing references in initially selected studies to identify further relevant research, could be employed to uncover additional pertinent studies.

6.2 Advancements in Audio Testing

6.2.1 Expanding Case Studies

As part of the continuous improvement of this study, it is crucial to enhance the external validity of the approach. One way to achieve this would be to increase the number of case studies examined. The use of multiple case studies could help to avoid overfitting the threshold to a single project, providing a more reliable and generalizable understanding of the effectiveness of the proposed approach.

Moreover, examining a variety of projects would help to discern whether the success of the approach is due to its inherent efficacy or whether it's the simplicity of the project considered that contributes to the success. Thus, the study of more complex or diverse projects could yield valuable insights into the robustness and versatility of the approach.

6.2.2 Improvements in sound recognition

- **Temporal sequence verification:** The sound detection process could be improved by verifying the temporal sequence of the chunk matches. This approach could lead to more effective recognition of false positive results.
- **Background sound removal:** To account for scenarios where background music or noise is present, noise cancellation or suppression technology could be developed and implemented.
- **Sound effect recognition:** As video games often apply different effects to sounds to enhance player immersion, recognizing and testing these sound effects can be a promising area for further exploration.

6.2.3 Integration of Speech-to-Text testing

- **Grammar Checker:** The current speech-to-text feature used in RiverGame could be enhanced with a grammar checker. This addition would complement the existing similarity check, which requires expected text, thus making the testing process more flexible and extensive. Instead of checking for exact matches, the tool would validate the grammatical correctness and comprehensibility of the spoken text.
- **Speech-to-Text in iv4xr:** The existing speech-to-text solution provided by RiverGame can be integrated into the iv4xr framework. This would enhance the testing capabilities by allowing verification of in-game dialogues and voiceovers.
- **Testing in Noise Overload Scenarios:** Future research could use iv4xr agents to simulate 'noise overload' scenarios, where numerous sounds are triggered simultaneously, potentially overwhelming the player. In such scenarios, sound recognition and speech-to-text testing can be applied to ascertain whether the game's audio remains understandable amid the noise overload.

References

[1] James Batchelor Editor-in-Chief. *GamesIndustry.Biz Presents... The Year In Numbers 2018*. GamesIndustry.biz. Dec. 17, 2018. URL: https://www.gamesindustry.biz/gamesindustry-biz-presents-the-year-in-numbers-2018 (visited on 07/01/2023).

[2] Aghyad Albaghajati and Moataz Ahmed. «Video Game Automated Testing Approaches: An Assessment Framework». In: *IEEE Transactions on Games* 15.1 (2020), pp. 81–94. ISSN: 2475-1510. DOI: 10.1109/TG.2020.3032796.

[3] Cristiano Politowski, Fabio Petrillo, and Yann-Gaël Guéhéneuc. «A Survey of Video Game Testing». In: *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. 2021 IEEE/ACM International Conference on Automation of Software Test (AST). May 2021, pp. 90–99. DOI: 10.1109/AST52587.2021.00018.

[4] Riccardo Coppola and Emil Alégroth. «A Taxonomy of Metrics for GUI-based Testing Research: A Systematic Literature Review». In: *Information and Software Technology* 152 (Dec. 1, 2022), p. 107062. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2022.107062. URL: https://www.sciencedirect.com/science/article/pii/S0950584922001719 (visited on 04/02/2023).

[5] *Iv4xr-Project/iv4xrDemo: A Demo of Agent-Based Testing Using Iv4xr*. URL: https://github.com/iv4xr-project/iv4xrDemo (visited on 07/01/2023).

[6] *Unibuc-Cs/Game-Testing: Prototype for a Game Testing Framework Using AI Methods*. URL: https://github.com/unibuc-cs/game-testing (visited on 07/02/2023).

[7] Cristiano Politowski, Yann-Gaël Guéhéneuc, and Fabio Petrillo. «Towards Automated Video Game Testing: Still a Long Way to Go». In: *2022 IEEE/ACM 6th International Workshop on Games and Software Engineering (GAS)*. 2022 IEEE/ACM 6th International Workshop on Games and Software Engineering (GAS). May 2022, pp. 37–43. DOI: 10.1145/3524494.3527627.

[8] *Learn Software Testing Tutorial - Javatpoint*. URL: https://www.javatpoint.com/software-testing-tutorial (visited on 07/02/2023).

[9] Mario Linares Vasquez, Kevin Moran, and Denys Poshyvanyk. *Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing*. Jan. 18, 2018. DOI: 10.48550/arXiv.1801.06267. arXiv: 1801.06267 [cs]. URL: http://arxiv.org/abs/1801.06267 (visited on 07/11/2023). preprint.

[10] Bailey Eric Nelson and Ito Yasunobu. «Livestreaming for User Testing Context-Rich Observation of Game Player Behavior». In: *2017 International Conference on Management Science and Engineering (ICMSE)*. 2017 International Conference on Management Science and Engineering (ICMSE). Aug. 2017, pp. 228–237. DOI: 10.1109/ICMSE.2017.8574431.

[11] Tariq King. *Transforming Gaming with AI-driven Automation - Sogeti*. Sogeti, provider of technology and engineering services. 2022. URL: https://www.sogeti.com/ai-for-qe/section-4-1-automate-see/chapter-5/ (visited on 07/02/2023).

[12] Ciprian Paduraru, Miruna Paduraru, and Alin Stefanescu. «RiverGame - a Game Testing Tool Using Artificial Intelligence». In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2022 IEEE Conference on Software Testing, Verification and Validation (ICST). Apr. 2022, pp. 422–432. DOI: 10.1109/ICST53961.2022.00048.

[13] *Wav2vec 2.0: Learning the structure of speech from raw audio*. URL: https://ai.facebook.com/blog/wav2vec-20-learning-the-structure-of-speech-from-raw-audio/ (visited on 07/05/2023).

[14] *Librosa/Librosa: Python Library for Audio and Music Analysis*. URL: https://github.com/librosa/librosa (visited on 07/05/2023).

[15] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. «Guidelines for Including Grey Literature and Conducting Multivocal Literature Reviews in Software Engineering». In: *Information and Software Technology* 106 (Feb. 1, 2019), pp. 101–121. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2018.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584918301939 (visited on 07/01/2023).

[16] Staffs Keele. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. 2007.

[17] *Towards Rigor in Reviews of Multivocal Literatures: Applying the Exploratory Case Study Method - Rodney T. Ogawa, Betty Malen, 1991*. URL: https://journals.sagepub.com/doi/10.3102/00346543061003265 (visited on 07/05/2023).

[18] *Grey Literature in Library and Information Studies*. URL: https://www.degruyter.com/document/doi/10.1515/9783598441493/html?lang=en (visited on 07/01/2023).

[19] *Shades of Grey: Guidelines for Working with the Grey Literature in Systematic Reviews for Management and Organizational Studies - Adams - 2017 - International Journal of Management Reviews - Wiley Online Library*. URL: https://onlinelibrary.wiley.com/doi/10.1111/ijmr.12102 (visited on 07/01/2023).

[20] Imants Zarembo. «ANALYSIS OF ARTIFICIAL INTELLIGENCE APPLICATIONS FOR AUTOMATED TESTING OF VIDEO GAMES». In: *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference* 2.0 (0 June 20, 2019), pp. 170–174. ISSN: 2256-070X. DOI: 10.17770/etr2019vol2.4158. URL: http://journals.ru.lv/index.php/ETR/article/view/4158 (visited on 03/31/2023).

[21] S. M. Bindu Bhargavi and V. Suma. «A Survey of the Software Test Methods and Identification of Critical Success Factors for Automation». In: *SN Computer Science* 3.6 (Aug. 20, 2022), p. 449. ISSN: 2661-8907. DOI: 10.1007/s42979-022-01297-5. URL: https://doi.org/10.1007/s42979-022-01297-5 (visited on 03/31/2023).

[22] *Table Capture, GeorgeMike.Com*. URL: https://www.georgemike.com/tablecapture/ (visited on 07/01/2023).

[23] *Publish or Perish*. URL: https://harzing.com/resources/publish-or-perish (visited on 07/01/2023).

[24] MattiaRiola. *Table Extractor from Google Search*. Mar. 10, 2023. URL: https://github.com/MattiaRiola/TableExtractor (visited on 07/01/2023).

[25] *Zotero | Your Personal Research Assistant*. URL: https://www.zotero.org/ (visited on 07/06/2023).

[26] *Notion – The All-in-One Workspace for Your Notes, Tasks, Wikis, and Databases*. URL: https://www.notion.so/ (visited on 07/06/2023).

[27] I. S. W. B. Prasetya et al. «An Agent-Based Approach to Automated Game Testing: An Experience Report». In: *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*. A-TEST 2022. New York, NY, USA: Association for Computing Machinery, Nov. 9, 2022, pp. 1–8. ISBN: 978-1-4503-9452-9. DOI: 10.1145/3548659.3561305. URL: https://dl.acm.org/doi/10.1145/3548659.3561305 (visited on 03/31/2023).

[28] Lior Tal. *Introduction to Unity Test Tools*. Game Developer. May 20, 2014. URL: https://www.gamedeveloper.com/programming/introduction-to-unity-test-tools (visited on 07/02/2023).

[29] Ruben Gonzalez. *Dependency Injection on Unity*. Game Developer. Sept. 8, 2020. URL: https://www.gamedeveloper.com/programming/dependency-injection-on-unity (visited on 07/02/2023).

[30] Ash Davis, Adam Single, Mark Hogben, and Leigh Mannes. *Testing for Game Development*. Game Developer. 2016. URL: https://www.gamedeveloper.com/programming/testing-for-game-development (visited on 07/02/2023).

[31] Game Developer staff. *Bring Light into Darkness - Â How to Add Tests to an Existing Project*. Game Developer. Sept. 30, 2020. URL: https://www.gamedeveloper.com/programming/bring-light-into-darkness---how-to-add-tests-to-an-existing-project (visited on 07/02/2023).

[32] Arlinta Christy Barus, Roy Deddy Hasiholan Tobing, Dani Novita Pratiwi, Siska Adelina Damanik, and Jenny Pasaribu. «Mobile Game Testing: Case Study of a Puzzle Game Genre». In: *2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*. 2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT). Oct. 2015, pp. 145–149. DOI: 10.1109/ICACOMIT.2015.7440194.

[33] A. Nikas. «Automated Gui Testing for Games Using Pseudo-DSL». MA thesis. 2015.

[34] Brendan LoBuglio. *Welcome to Rabbit Hell! Reliable AI Locomotion with TDD*. 2019. URL: https://www.gamedeveloper.com/programming/welcome-to-rabbit-hell-reliable-ai-locomotion-with-tdd (visited on 07/02/2023).

[35] Doug Stevenson. *Test Your Game with Firebase Test Lab for Android*. The Firebase Blog. 2017. URL: https://firebase.blog/ (visited on 07/02/2023).

[36] Robert Masella. *Automated Testing of Gameplay Features in 'Sea of Thieves'*. 2019. URL: https://www.gdcvault.com/play/1026366/Automated-Testing-of-Gameplay-Features (visited on 04/20/2023).

[37] Afza Kazmi, Anam Fatima, Arsalan Idris, and Shujaat Hussain. «Adaptive Usage Statistical Testing for 3D Gaming Applications». In: *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*. 2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube). Nov. 2018, pp. 1–6. DOI: 10.1109/ICECUBE.2018.8610975.

[38] Unreal Engine. *Automation System Overview*. 2021. URL: https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/Automation/ (visited on 07/02/2023).

[39] Santosh Raj Dandey. «An Automated Testing Framework for the Virtual Cell Game». In: (2013). URL: https://library.ndsu.edu/ir/handle/10365/23082 (visited on 03/31/2023).

[40] Yan Zheng 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)*. 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Nov. 2019, pp. 772–784. DOI: 10.1109/ASE.2019.00077.

[41] Samira Shirzadehhajimahmood, I. S. W. B. Prasetya, Frank Dignum, Mehdi Dastani, and Gabriele Keller. «Using an Agent-Based Approach for Robust Automated Testing of Computer Games». In: *Proceedings of the 12th International Workshop on Automating TEST Case Design, Selection, and Evaluation*. A-TEST 2021. New York, NY, USA: Association for Computing Machinery, Aug. 23, 2021, pp. 1–8. ISBN: 978-1-4503-8623-4. DOI: 10.1145/3472672.3473952. URL: https://dl.acm.org/doi/10.1145/3472672.3473952 (visited on 03/30/2023).

[42] Chenglong Sun, Zhenyu Zhang, Bo Jiang, and W.K. Chan. «Facilitating Monkey Test by Detecting Operable Regions in Rendered GUI of Mobile Game Apps». In: *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). Aug. 2016, pp. 298–306. DOI: 10.1109/QRS.2016.41.

[43] Hirotaka Suetake, Tsukasa Fukusato, Christian Arzate Cruz, Andy Nealen, and Takeo Igarashi. «Interactive Design Exploration of Game StagesUsing Adjustable Synthetic Testers». In: *Proceedings of the 15th International Conference on the Foundations of Digital Games*. FDG '20. New York, NY, USA: Association for Computing Machinery, Sept. 17, 2020, pp. 1–4. ISBN: 978-1-4503-8807-8. DOI: 10.1145/3402942.3402982. URL: https://dl.acm.org/doi/10.1145/3402942.3402982 (visited on 03/30/2023).

[44] Grendel games. *Automating Level Testing with AI - Grendel Games*. July 2, 2019. URL: https://grendelgames.com/automating-level-testing-with-ai/ (visited on 07/02/2023).

[45] Wael Awad. «Game Testing Automation Guidance». fi=Ylempi AMK-opinnäytetyö|sv=Högre YH-examensarbete|en=Master's thesis|. 2021. URL: http://www.theseus.fi/handle/10024/505977 (visited on 03/30/2023).

[46] Matthew Barthet, Ahmed Khalifa, Antonios Liapis, and Georgios Yannakakis. «Generative Personas That Behave and Experience Like Humans». In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. FDG '22. New York, NY, USA: Association for Computing Machinery, Nov. 4, 2022, pp. 1–10. ISBN: 978-1-4503-9795-7. DOI: 10.1145/3555858.3555879. URL: https://dl.acm.org/doi/10.1145/3555858.3555879 (visited on 03/30/2023).

[47] «Automation in the Game Testing. Approaches and Solutions». In: 2020. URL: https://er.knutd.edu.ua/bitstream/123456789/15306/1/ITPF2020_P232-235.pdf.

[48] Guanhua Hou, Hua Dong, and Yang Yang. «Developing a Virtual Reality Game User Experience Test Method Based on EEG Signals». In: *2017 5th International Conference on Enterprise Systems (ES)*. 2017 5th International Conference on Enterprise Systems (ES). Sept. 2017, pp. 227–231. DOI: 10.1109/ES.2017.45.

[49] GameDriver. *GameDriver: Home*. GameDriver. 2019. URL: https://gamedriver.io/ (visited on 07/02/2023).

[50] I. S. W. B. Prasetya and Mehdi Dastani. «Aplib: An Agent Programming Library for Testing Games». In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 13, 2020, pp. 1972–1974. ISBN: 978-1-4503-7518-4.

[51] WeTest. *How to Automate Unity Games Using Altunity Tester*. EuroSTAR Huddle. Aug. 8, 2022. URL: https://huddle.eurostarsoftwaretesting.com/how-to-automate-unity-games-using-altunity-tester/ (visited on 07/02/2023).

[52] Samantha Stahlke, Atiya Nova, and Pejman Mirza-Babaei. «Artificial Players in the Design Process: Developing an Automated Testing Tool for Game Level and World Design». In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, Nov. 3, 2020, pp. 267–280. ISBN: 978-1-4503-8074-4. DOI: 10.1145/3410404.3414249. URL: https://dl.acm.org/doi/10.1145/3410404.3414249 (visited on 03/31/2023).

[53] Ruben Torres Bonet. *Unity Immediate and the Art of Automating Playtests*. Game Developer. 2019. URL: https://www.gamedeveloper.com/design/unity-immediate-and-the-art-of-automating-playtests (visited on 07/02/2023).

[54] Zihe Song. «An Automated Framework For Gaming Platform To Test Multiple Games». In: *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). Oct. 2020, pp. 134–136.

[55] Simon Liu et al. «Automatic Generation of Tower Defense Levels Using PCG». In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. FDG '19. New York, NY, USA: Association for Computing Machinery, Aug. 26, 2019, pp. 1–9. ISBN: 978-1-4503-7217-6. DOI: 10.1145/3337722.3337723. URL: https://dl.acm.org/doi/10.1145/3337722.3337723 (visited on 03/31/2023).

[56] Designbeep. *Learn the Basics of Automated Game Testing - Designbeep*. 2022. URL: https://designbeep.com/2022/06/28/learn-the-basics-of-automated-game-testing/ (visited on 07/02/2023).

[57] Markus Schatten, Bogdan Okreša Đurić, and Igor Tomičić. «Towards an Application Programming Interface for Automated Testing of Artificial Intelligence Agents in Massively Multi-Player on-Line Role-Playing Games». In: *Central European Conference on Information and Intelligent Systems*. Faculty of Organization and Informatics Varazdin, 2018, pp. 11–15.

[58] Aghyad Albaghajati and Moataz Ahmed. «A Co-Evolutionary Genetic Algorithms Approach to Detect Video Game Bugs». In: *Journal of Systems and Software* 188 (June 1, 2022), p. 111261. ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111261. URL: https://www.sciencedirect.com/science/article/pii/S0164121222000292 (visited on 03/31/2023).

[59] Xiaoyin Wang. «VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes». In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. ICSE '22. New York, NY, USA: Association for Computing Machinery, Oct. 19, 2022, pp. 232–236. ISBN: 978-1-4503-9223-5. DOI: 10.1145/3510454.3516870. URL: https://dl.acm.org/doi/10.1145/3510454.3516870 (visited on 03/30/2023).

[60] Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. «Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving». In: *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*. CHI PLAY

'17 Extended Abstracts. New York, NY, USA: Association for Computing Machinery, Oct. 15, 2017, pp. 153–164. ISBN: 978-1-4503-5111-9. DOI: 10.1145/3130859.3131439. URL: https://dl.acm.org/doi/10.1145/3130859.3131439 (visited on 03/31/2023).

[61] T-Plan. *Game Test Automation - T-Plan*. 2017. URL: https://www.t-plan.com/game-test-automation/ (visited on 07/02/2023).

[62] Bo Jiang, Wenlin Wei, Li Yi, and W.K. Chan. «DroidGamer: Android Game Testing with Operable Widget Recognition by Deep Learning». In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS). Dec. 2021, pp. 197–206. DOI: 10.1109/QRS54544.2021.00031.

[63] Rosalia Tufano, Simone Scalabrino, Luca Pascarella, Emad Aghajani, Rocco Oliveto, and Gabriele Bavota. «Using Reinforcement Learning for Load Testing of Video Games». In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. New York, NY, USA: Association for Computing Machinery, July 5, 2022, pp. 2303–2314. ISBN: 978-1-4503-9221-1. DOI: 10.1145/3510003.3510625. URL: https://dl.acm.org/doi/10.1145/3510003.3510625 (visited on 03/30/2023).

[64] AltTester. *AltTester - Test Automation Tools for Unity Apps and Games*. AltTester. 2022. URL: https://alttester.com/alttester/ (visited on 07/02/2023).

[65] Adity M. Sidiq, Jati H. Husen, and Sri Widowati. «A Bot Approach-Based Capacity Testing Automation for Online Video Games». In: *Jurnal ELTIKOM : Jurnal Teknik Elektro, Teknologi Informasi dan Komputer* 6.2 (Nov. 17, 2022), pp. 118–125. ISSN: 2598-3288. DOI: 10.31961/eltikom.v6i2.550. URL: https://eltikom.poliban.ac.id/index.php/eltikom/article/view/550 (visited on 03/31/2023).

[66] Sidra Iftikhar, Muhammad Zohaib Iqbal, Muhammad Uzair Khan, and Wardah Mahmood. «An Automated Model Based Testing Approach for Platform Games». In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). Sept. 2015, pp. 426–435. DOI: 10.1109/MODELS.2015.7338274.

[67] Jesper Lehtinen. «Automated GUI Testing of Game Development Tools». Metropolia Ammattikorkeakoulu, 2016.

[68] Samira Shirzadehhajimahmood, I. S. W. B. Prasetya, Frank Dignum, and Mehdi Dastani. «An Online Agent-Based Search Approach in Automated Computer Game Testing with Model Construction». In: *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*. A-TEST 2022. New York, NY, USA: Association for Computing Machinery, Nov. 9, 2022, pp. 45–52. ISBN: 978-1-4503-9452-9. DOI: 10.1145/3548659.3561309. URL: https://dl.acm.org/doi/10.1145/3548659.3561309 (visited on 03/31/2023).

[69] Alessandro Sestini, Linus Gisslén, Joakim Bergdahl, Konrad Tollmar, and Andrew D. Bagdanov. «Automated Gameplay Testing and Validation with Curiosity-Conditioned Proximal Trajectories». In: *IEEE Transactions on Games* (2022), pp. 1–14. ISSN: 2475-1510. DOI: 10.1109/TG.2022.3226910.

[70] Markus Schatten, Igor Tomičić, Bogdan Okreša Đurić, and Nikola Ivković. «Towards an Agent-Based Automated Testing Environment for Massively Multi-Player Role Playing Games». In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). May 2017, pp. 1149–1154. DOI: 10.23919/MIPRO.2017.7973597.

[71] Sinan Arıyürek. «AUTOMATED VIDEO GAME TESTING USING REINFORCEMENT LEARNING AGENTS». In: (2022).

[72] KMS SOLUTIONS. *Level Up Your Testing Game With Automation Testing Life Cycle*. 2022. URL: https://blog.kms-solutions.asia/level-up-your-testing-game-with-automation-testing-life-cycle (visited on 07/02/2023).

[73] Haris Kljajic and Oskar Karlsson. *Applying Automated Testing in an Existing Client-Server Game: A Pursuit for Fault Localization in Quake 3*. 2015.

[74] Sinan Ariyurek, Aysu Betin-Can, and Elif Surer. «Automated Video Game Testing Using Synthetic and Humanlike Agents». In: *IEEE Transactions on Games* 13.1 (Mar. 2021), pp. 50–67. ISSN: 2475-1510. DOI: 10.1109/TG.2019.2947597.

[75] Ville-Veikko Helppi. *The Basics Of Test Automation For Apps, Games And The Mobile Web*. Smashing Magazine. 2015. URL: https://www.smashingmagazine.com/2015/01/basic-test-automation-for-apps-games-and-mobile-web/ (visited on 07/02/2023).

[76] Gabriel Lovreto, Andre T. Endo, Paulo Nardi, and Vinicius H. S. Durelli. «Automated Tests for Mobile Games: An Experience Report». In: *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). Oct. 2018, pp. 48–488. DOI: 10.1109/SBGAMES.2018.00015.

[77] Timea Pusok. *How to: Automated Tests for Unity Mobile Apps with Appium and AltUnity*. Game Developer. Apr. 22, 2021. URL: https://www.gamedeveloper.com/programming/how-to-automated-tests-for-unity-mobile-apps-with-appium-and-altunity (visited on 07/02/2023).

[78] Ciprian Paduraru and Miruna Paduraru. «Automatic Difficulty Management and Testing in Games Using a Framework Based on Behavior Trees and Genetic Algorithms». In: *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS). Nov. 2019, pp. 170–179. DOI: 10.1109/ICECCS.2019.00026.

[79] Oleguer Canal Anton. «Automatic Game-Testing with Personality: Multi-task Reinforcement Learning for Automatic Game-Testing». 2021. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-309091 (visited on 03/31/2023).

[80] Yuechen Wu, Yingfeng Chen, Xiaofei Xie, Bing Yu, Changjie Fan, and Lei Ma. «Regression Testing of Massively Multiplayer Online Role-Playing Games». In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sept. 2020, pp. 692–696. DOI: 10.1109/ICSME46990.2020.00074.

[81] Felix Nilsson and Jesper Nilsson. «Comparing Automated Testing Approaches for FPS Games». 2021.

[82] Masato Yamamoto, Evgeny Pyshkin, and Maxim Mozgovoy. «Reducing False Positives in Automated OpenCV-based Non-Native GUI Software Testing». In: *Proceedings of the 3rd International Conference on Applications in Information Technology*. ICAIT'2018. New York, NY, USA: Association for Computing Machinery, Nov. 1, 2018, pp. 41–45. ISBN: 978-1-4503-6516-1. DOI: 10.1145/3274856.3274865. URL: https://dl.acm.org/doi/10.1145/3274856.3274865 (visited on 03/30/2023).

[83] Michail Ostrowski and Samir Aroudj. «Automated Regression Testing within Video Game Development». In: *GSTF Journal on Computing (JoC)* 3.2 (July 23, 2013), p. 10. ISSN: 2010-2283. DOI: 10.7603/s40601-013-0010-

4. URL: https://doi.org/10.7603/s40601-013-0010-4 (visited on 03/31/2023).

[84] Guoqing Liu, Mengzhang Cai, Li Zhao, Tao Qin, Adrian Brown, Jimmy Bischoff, and Tie-Yan Liu. «Inspector: Pixel-Based Automated Game Testing via Exploration, Detection, and Investigation». In: *2022 IEEE Conference on Games (CoG)*. 2022 IEEE Conference on Games (CoG). Aug. 2022, pp. 237–244. DOI: 10.1109/CoG51982.2022.9893630.

[85] Tencent. *Tencent/GAutomator: Automation for Mobile Games - GitHub*. Tencent, 2017. URL: https://github.com/Tencent/GAutomator (visited on 07/02/2023).

[86] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. «GLIB: Towards Automated Test Oracle for Graphically-Rich Applications». In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, Aug. 18, 2021, pp. 1093–1104. ISBN: 978-1-4503-8562-6. DOI: 10.1145/3468264.3468586. URL: https://dl.acm.org/doi/10.1145/3468264.3468586 (visited on 03/30/2023).

[87] Sasha Volokh and William G.J. Halfond. «Static Analysis for Automated Identification of Valid Game Actions During Exploration». In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. FDG '22. New York, NY, USA: Association for Computing Machinery, Nov. 4, 2022, pp. 1–10. ISBN: 978-1-4503-9795-7. DOI: 10.1145/3555858.3555898. URL: https://dl.acm.org/doi/10.1145/3555858.3555898 (visited on 03/30/2023).

[88] Cong Lu, Raluca Georgescu, and Johan Verwey. «Go-Explore Complex 3D Game Environments for Automated Reachability Testing». In: *IEEE Transactions on Games* (2022), pp. 1–6. ISSN: 2475-1510. DOI: 10.1109/TG.2022.3228401.

[89] Jiaming Ye, Ke Chen, Xiaofei Xie, Lei Ma, Ruochen Huang, Yingfeng Chen, Yinxing Xue, and Jianjun Zhao. «An Empirical Study of GUI Widget Detection for Industrial Mobile Games». In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, Aug. 18, 2021, pp. 1427–1437. ISBN: 978-1-4503-8562-6. DOI: 10.1145/3468264.3473935. URL: https://dl.acm.org/doi/10.1145/3468264.3473935 (visited on 03/31/2023).

[90] Sampo Tuisku. «Automated Regression Testing for Cloud Based Mobile Games». 2020.

[91] Pablo García-Sánchez, Alberto Tonda, Antonio M. Mora, Giovanni Squillero, and Juan Julián Merelo. «Automated Playtesting in Collectible Card Games Using Evolutionary Algorithms: A Case Study in Hearthstone». In: *Knowledge-Based Systems* 153 (Aug. 1, 2018), pp. 133–146. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.04.030. URL: https://www.sciencedirect.com/science/article/pii/S0950705118301953 (visited on 03/31/2023).

[92] Injung Lee, Hyunchul Kim, and Byungjoo Lee. «Automated Playtesting with a Cognitive Model of Sensorimotor Coordination». In: *Proceedings of the 29th ACM International Conference on Multimedia*. MM '21. New York, NY, USA: Association for Computing Machinery, Oct. 17, 2021, pp. 4920–4929. ISBN: 978-1-4503-8651-7. DOI: 10.1145/3474085.3475429. URL: https://dl.acm.org/doi/10.1145/3474085.3475429 (visited on 03/31/2023).

[93] Atiya Nova, Stevie Sansalone, Raquel Robinson, and Pejman Mirza-Babaei. «Charting the Uncharted with GUR: How AI Playtesting Can Supplement Expert Evaluation». In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. FDG '22. New York, NY, USA: Association for Computing Machinery, Nov. 4, 2022, pp. 1–12. ISBN: 978-1-4503-9795-7. DOI: 10.1145/3555858.3555880. URL: https://dl.acm.org/doi/10.1145/3555858.3555880 (visited on 03/31/2023).

[94] Ciprian Paduraru, Miruna Paduraru, and Alin Stefanescu. «Automated Game Testing Using Computer Vision Methods». In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). Nov. 2021, pp. 65–72. DOI: 10.1109/ASEW52652.2021.00024.

[95] Electronic Arts and Linus Gisslén. *Re • Work 2021: Automated Game Testing - EA Games*. Electronic Arts Inc. 2021. URL: https://www.ea.com/seed/news/re-work-2021-automated-game-testing (visited on 07/02/2023).

[96] Christopher Schaefer, Hyunsook Do, and Brian M. Slator. «Crushinator: A Framework towards Game-Independent Testing». In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). Nov. 2013, pp. 726–729. DOI: 10.1109/ASE.2013.6693143.

[97] Tommy Thompson. *How Tom Clancy's The Division Manages AI Online*. Game Developer. Dec. 10, 2018. URL: https://www.gamedeveloper.com/design/how-tom-clancy-s-the-division-manages-ai-online (visited on 07/02/2023).

[98] Mihail Morosan and Riccardo Poli. «Lessons from Testing an Evolutionary Automated Game Balancer in Industry». In: *2018 IEEE Games, Entertainment, Media Conference (GEM)*. 2018 IEEE Games, Entertainment, Media Conference (GEM). Aug. 2018, pp. 263–270. DOI: 10.1109/GEM.2018.8516447.

[99] Auguste Jerlström. «Utilizing Input Simulation for Video Game Test Automation : A Case Study». 2022. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-321473 (visited on 03/30/2023).

[100] Xudong Li, Dajun Zhou, Like Zhang, and Yanqing Jing. «Human-like UI Automation through Automatic Exploration». In: *Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence*. ISBDAI '20. New York, NY, USA: Association for Computing Machinery, Jan. 4, 2021, pp. 47–53. ISBN: 978-1-4503-7645-7. DOI: 10.1145/3436286.3436297. URL: https://dl.acm.org/doi/10.1145/3436286.3436297 (visited on 03/30/2023).

[101] J. Tuovenen, M. Oussalah, and P. Kostakos. «MAuto: Automatic Mobile Game Testing Tool Using Image-Matching Based Approach». In: *The Computer Games Journal* 8.3 (Dec. 1, 2019), pp. 215–239. ISSN: 2052-773X. DOI: 10.1007/s40869-019-00087-z. URL: https://doi.org/10.1007/s40869-019-00087-z (visited on 03/30/2023).

[102] Maxim Mozgovoy and Evgeny Pyshkin. «A Comprehensive Approach to Quality Assurance in a Mobile Game Project». In: *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia*. CEE-SECR '18. New York, NY, USA: Association for Computing Machinery, Oct. 12, 2018, pp. 1–8. ISBN: 978-1-4503-6176-7. DOI: 10.1145/3290621.3290835. URL: https://dl.acm.org/doi/10.1145/3290621.3290835 (visited on 03/31/2023).

[103] R. Ferdous, F. Kifetew, D. Prandi, I.S.W.B. Prasetya, S. Shirzadehhajimahmood, and A. Susi. «Search-Based Automated Play Testing of Computer Games: A Model-Based Approach». In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12914 LNCS (2021), pp. 56–71. ISSN: 0302-9743. DOI: 10.1007/978-3-030-88106-1_5.

[104] Tommy Thompson. *The Secret AI Testers inside Tom Clancy's The Division*. Game Developer. 2020. URL: https://www.gamedeveloper.com/design/the-secret-ai-testers-inside-tom-clancy-s-the-division (visited on 07/02/2023).

[105] Ru Cindrea. *AltUnityTester – Testing Unity Games and Apps Using Appium*. SmartBear.com. 2018. URL: https://smartbear.com/blog/guest-blog-testing-unity-games-using-appium/ (visited on 07/02/2023).

[106] David Andrade. «Designing Cloud-Based Gameplay Automation: Exploratory Software Testing, Game State-Analysis, and Test-Driven Development (TDD) Applied to Robotic Process Automation (RPA)». In: *International Journal of Intelligent Computing Research* 13 (June 30, 2022), pp. 1125–1135. DOI: 10.20533/ijicr.2042.4655.2022.0137.

[107] Raphaël Marczak, Jasper van Vught, Gareth Schott, and Lennart E. Nacke. «Feedback-Based Gameplay Metrics: Measuring Player Experience via Automatic Visual Analysis». In: *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*. IE '12. New York, NY, USA: Association for Computing Machinery, July 21, 2012, pp. 1–10. ISBN: 978-1-4503-1410-7. DOI: 10.1145/2336727.2336733. URL: https://dl.acm.org/doi/10.1145/2336727.2336733 (visited on 03/31/2023).

[108] Jennifer Hernández Bécares, Luis Costero Valero, and Pedro Pablo Gómez Martín. «An Approach to Automated Videogame Beta Testing». In: *Entertainment Computing* 18 (Jan. 1, 2017), pp. 79–92. ISSN: 1875-9521. DOI: 10.1016/j.entcom.2016.08.002. URL: https://www.sciencedirect.com/science/article/pii/S1875952116300234 (visited on 03/31/2023).

[109] Seungjin Kwon, Jaehyun Ahn, Hyukgeun Choi, Jiho Jeon, Doyoung Kim, Hoyeon Kim, and Shinjin Kang. «Analytical Framework for Facial Expression on Game Experience Test». In: *IEEE Access* 10 (2022), pp. 104486–104497. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3210712.

[110] Simon Varvaressos, Kim Lavoie, Sébastien Gaboury, and Sylvain Hallé. «Automated Bug Finding in Video Games: A Case Study for Runtime Monitoring». In: *Computers in Entertainment* 15.1 (Mar. 1, 2017), 1:1–1:28. DOI: 10.1145/2700529. URL: https://dl.acm.org/doi/10.1145/2700529 (visited on 03/31/2023).

[111] Timothy Ryan. *AB Split Testing Hell or Highwater*. Game Developer. Dec. 21, 2016. URL: https://www.gamedeveloper.com/programming/ab-split-testing-hell-or-highwater (visited on 07/02/2023).

[112] Ross Mawhorter and Adam Smith. «Softlock Detection for Super Metroid with Computation Tree Logic». In: *Proceedings of the 16th International Conference on the Foundations of Digital Games*. FDG '21. New York, NY, USA: Association for Computing Machinery, Oct. 21, 2021, pp. 1–10. ISBN: 978-1-4503-8422-3. DOI: 10.1145/3472538.3472542. URL: https://dl.acm.org/doi/10.1145/3472538.3472542 (visited on 03/30/2023).

[113] I.S.W.B. Prasetya et al. «Navigation and Exploration in 3D-game Automated Play Testing». In: A-TEST 2020 - Proceedings of the 11th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, Co-located with ESEC/FSE 2020. 2020, pp. 3–9. ISBN: 978-1-4503-8101-7. DOI: 10.1145/3412452.3423570.

[114] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. «Augmenting Automated Game Testing with Deep Reinforcement Learning». In: *2020 IEEE Conference on Games (CoG)*. 2020 IEEE Conference on Games (CoG). Aug. 2020, pp. 600–603. DOI: 10.1109/CoG47356.2020.9231552.

[115] QAble Testlab. *9 Different Types of Game Testing Techniques - Testbytes*. Testbytes. Mar. 6, 2022. URL: https://www.testbytes.net/ (visited on 07/02/2023).

[116] Bultman Aaron. *A Complete Guide to Game Testing - Its Types and Processes*. 2022. URL: https://www.headspin.io/blog/game-testing-a-complete-guide-to-its-types-and-processes (visited on 07/02/2023).

[117] Johnny Lam. *Testing Mobile Games | Perfecto by Perforce*. 2021. URL: https://www.perfecto.io/blog/testing-mobile-games (visited on 07/02/2023).

[118] *Game Testing - Wikipedia*. In: *Wikipedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Game_testing&oldid=1162271871 (visited on 07/02/2023).

[119] Avanish Pandey. *Game Testing | 13 Types of Techniques for Game Testing*. Astaqc Consulting. Feb. 4, 2023. URL: https://astaqc.com/software-testing-blog/game-testing-13-types-of-techniques-for-game-testing/ (visited on 07/02/2023).

[120] Testscenario. *Types of Game Testing - Testscenario*. 2022. URL: https://www.testscenario.com/types-of-game-testing/ (visited on 07/02/2023).

[121] Johan Hoberg. *Game Testing: Exploring the Test Space*. Game Developer. Aug. 21, 2014. URL: https://www.gamedeveloper.com/programming/game-testing-exploring-the-test-space (visited on 07/02/2023).

[122] Antonio Torres. *5 Different Types of Game Testing Techniques*. Game Developer. May 22, 2019. URL: https://www.gamedeveloper.com/design/5-different-types-of-game-testing-techniques (visited on 07/02/2023).

[123] pflb. *Performance Testing for Online Games and Game Servers*. Dec. 21, 2021. URL: https://pflb.us/blog/performance-testing-for-online-games-and-game-servers/ (visited on 07/02/2023).

[124] QAble Testlab. *7 Different Types of Game Testing Techniques - QAble*. QAble. Mar. 8, 2021. URL: https://www.qable.io/7-different-types-of-game-testing-techniques/ (visited on 07/02/2023).

[125] Zoheb Khan. *Game Testing: Importance, Types, Techniques, Tools and Benefits*. 2021. URL: https://www.bugraptors.com/blog/game-testing-importance-types-techniques-tools-and-benefits (visited on 07/02/2023).

[126] Sergey Almyashev. *What Tools Do You Need for Automation Video Game Testing?* Dec. 20, 2021. URL: https://zapple.tech/blog/test-automation-frameworks/what-tools-do-you-need-to-automate-video-game-testing/ (visited on 07/02/2023).

[127] Mammoth-AI. *Finding Bugs in Game Testing - Mammoth-AI*. Mammoth-AI. Aug. 5, 2021. URL: https://www.mammoth-ai.com/finding-bugs-in-game-testing/ (visited on 07/02/2023).

[128] Game-Ace. *Your Ultimate Guide to Game Testing - Game-Ace*. Game-Ace. 2021. URL: https://game-ace.com/blog/guide-to-game-testing/ (visited on 07/02/2023).

[129] Johan Hoberg. *Differences between Software Testing and Game Testing*. Game Developer. July 21, 2014. URL: https://www.gamedeveloper.com/programming/differences-between-software-testing-and-game-testing (visited on 07/02/2023).

[130] *ISO 25010*. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (visited on 07/04/2023).

[131] Virtusa. *Mobile Game Testing Services | Virtusa*. 2021. URL: https://www.virtusa.com/solutions/mobile-game-testing (visited on 07/02/2023).

[132] Huixian Hu and Lu Lu. «Automatic Functional Testing of Unity 3D Game on Android Platform». In: 2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control. Atlantis Press, Apr. 2016, pp. 1136–1140. ISBN: 978-94-6252-173-5. DOI: 10.2991/icmemtc-16.2016.225. URL: https://www.atlantis-press.com/proceedings/icmemtc-16/25852327 (visited on 03/31/2023).

[133] modl.ai. *Modl:Test | Automated Game Testing Using AI Bots*. modl.ai | AI Engine for game development. 2021. URL: https://modl.ai/our_products/modl-test/ (visited on 07/02/2023).

[134] Vadeesh Budramane. *Gaming Test Automation - AlgoShack*. AlgoShack. Aug. 16, 2022. URL: https://www.algoshack.com/gaming-test-automation/ (visited on 07/02/2023).

[135] *Selenium*. Selenium. URL: https://www.selenium.dev/ (visited on 07/05/2023).

[136] *Appium*. URL: https://appium.io/docs/en/2.0/ (visited on 07/06/2023).

[137] *Robotium*. RobotiumTech, May 27, 2023. URL: https://github.com/RobotiumTech/robotium (visited on 07/06/2023).

[138] *Test Apps on Android*. Android Developers. URL: https://developer.android.com/training/testing (visited on 07/06/2023).

[139] *Tesseract OCR*. tesseract-ocr, July 5, 2023. URL: https://github.com/tesseract-ocr/tesseract (visited on 07/05/2023).

[140] *OpenCV*. OpenCV. URL: https://opencv.org/ (visited on 07/06/2023).

[141] *iv4XR*. GitHub. URL: https://github.com/iv4xr-project (visited on 07/06/2023).

[142] Unity Technologies. *Unity Test Framework for Video Game Development | QA & Testing | Unity*. URL: https://unity.com/how-to/unity-test-framework-video-game-development (visited on 07/06/2023).

[143] *Automation System Overview*. URL: https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/Automation/ (visited on 07/05/2023).

[144] *POCO*. URL: https://github.com/AirtestProject/Poco (visited on 07/05/2023).

[145] *Hearthstone*. URL: https://hearthstone.blizzard.com/en-us (visited on 07/05/2023).

[146] AirTest. *How to Test Games Based on the Unity3D Engine*. 2020. URL: https://airtest.doc.io.netease.com/en/tutorial/11_test_Unity3D_game/ (visited on 07/02/2023).

[147] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J. Davison. «KAZE Features». In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 214–227. ISBN: 978-3-642-33783-3. DOI: 10.1007/978-3-642-33783-3_16.

[148] *Home > xUnit.Net*. URL: https://xunit.net/ (visited on 07/05/2023).

[149] *UnityActionAnalysis: Automatically Analyze the Input-Handling Code of Unity Games to Determine Valid Game Actions*. URL: https://github.com/USC-SQL/UnityActionAnalysis (visited on 07/06/2023).

[150] *WAV*. In: *Wikipedia*. July 6, 2023. URL: https://en.wikipedia.org/w/index.php?title=WAV&oldid=1163811156 (visited on 07/07/2023).

[151] *Online Audio Converter | Convert Your Audio Files to MP3, WAV, FLAC, OGG and More*. URL: https://onlineaudioconverter.com/ (visited on 07/07/2023).

[152] Michael T. Heideman, Don H. Johnson, and C. Sidney Burrus. «Gauss and the History of the Fast Fourier Transform». In: *Archive for History of Exact Sciences* 34.3 (Sept. 1, 1985), pp. 265–277. ISSN: 1432-0657. DOI: 10.1007/BF00348431. URL: https://doi.org/10.1007/BF00348431 (visited on 07/08/2023).