Emanuela Guglielmi

Analyzing Gameplay Videos to Support Video Game Developers

From Issue Detection to Game Sessions Reproduction

PhD Thesis

University of Molise, Italy

November, 2024

Università degli Studi del Molise

Dipartimento di Bioscienze e Territorio

Corso di Dottorato in Scienze Applicate

Ciclo XXXVII

S.S.D. IINF-05/A

PhD Thesis

Analyzing Gameplay Videos to Support Video Game Developers

From Issue Detection to Game Sessions Reproduction

Coordinator	Tutor	Co-Tutors
Chiar.mo Prof. Filippo Santucci De Magistris	Chiar.mo Prof. Simone Scalabrino	Chiar.mo Prof. Rocco Oliveto
		Chiar.mo Prof. Gabriele Bavota

Candidate
Emanuela Guglielmi

Accademic Year 2023/2024

Abstract

The gaming industry has seen significant growth in recent years. Video games present a unique challenge for testing due to their large number of states, which are reached through the different choices of players. Millions of players worldwide daily produce vast amounts of gameplay videos. Although such videos have significant potential as a testing resource since they might document video game issues, their use remains not widely explored in current research. This PhD thesis addresses this gap by proposing (i) strategies to automatically extract meaningful information from gameplay videos, and (ii) a technique to replicate the sequence of in-game actions. We introduce GELID, an approach that analyzes gameplay videos to identify game-related issues by extracting significant segments based on textual features. We observed that detecting performance and balance issues using textual features alone is challenging, which led us to develop targeted solutions. The first one is HASTE, which identifies performance issues through visual analysis, finding areas where stuttering events occur. Additionally, we conducted a study to capture players' *real* engagement levels by gathering self-reported data from participants during gameplay sessions, providing a more accurate measure than previous reliance on facial expression interpretations. Finally, we present RePlay, a method that replicates player actions from gameplay videos to reproduce sequences leading to specific issues. Our findings demonstrate the value of automated gameplay video analysis in enhancing game quality assurance processes. Feedback from practitioners involved in our studies supports the relevance of these methodologies in improving the efficiency of game testing.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

1.1 Application Context

Video games are becoming an increasingly important form of expression in today's culture. Their sociological, economic, and technological impact is well recognized in the literature [81]. This pervasive popularity, particularly among younger generations, has contributed to the growth of the gaming industry, which is the most profitable entertainment industry in the world [136, 135, 137]. Market revenue is projected to grow by over $211 billion between 2024 and 2029,

reaching an estimated $667 billion by 2029, marking the seventh consecutive year of growth [40]. No longer confined to a niche, video games have shed their “nerdy” reputation and now engage a global audience of about 2.6 billion players [43]. As game titles become available on almost every digital device, video games have become one of the world’s most popular and enduring forms of entertainment. The video game industry offers an endless variety of titles to satisfy different requests [51]. Playing video games is progressively becoming a work for many: Some play for professional competitions (*e.g.,* in e-sports or speed-running), while others play to entertain (*e.g.,* streamers) especially on dedicated platforms such as YouTube [2] and Twitch [1]. At the end of 2021, Twitch users watched 6.13 billion hours of content, highlighting the immense popularity of the platform within the gaming community, reaching a peak of 9.89 million active streamers [42, 41].

Video games have been dominant in digital entertainment for decades, but only recently have they been recognized as valuable software artifacts by the software engineering (SE) community [107, 22]. Just like other software systems, video games can be affected by several types of functional and non-functional issues. Due to the complexity involved in developing this type of software, the costs and production times for video games can be significantly high. For example, Rockstar Games initially allocated a base budget of $140 million for the development of *Grand Theft Auto V*, yet by the time of its release, total costs had exceeded $265 million. Development began in April 2008, shortly after *Grand Theft Auto V*, and involved thousands of contributors over five years until its release in 2013 [88, 56]. Given these substantial investments, the presence of problems poses a significant risk to the companies producing such games, underscoring the critical need to minimize errors throughout development. For this reason, rigorous testing is essential to ensure quality, enabling developers to identify issues early and reduce costs.

Game developers conduct several phases of testing (*e.g.,* alpha testing, playtesting), where the beta testing phase is one of the most important [92]. This

step involves a large number of players to test different aspects—functionality, graphics, sound, and gameplay—to discover a wide range of potential issues. However, some problems occur only under specific and rare conditions, making them difficult to detect, even for human testers [22].

1.2 Motivation

Writing automated tests is far from trivial due to the need for an "intelligent" interaction triggering the states exploration. Even assuming such an ability to explore the game space, determining the correct behavior in a specific state usually requires human assessment. Additional complexity comes from the non-determinism in games due to multi-threading, diverse hardware configurations, distributed computing and artificial intelligence, and randomized mechanics designed to enhance game difficulty [107]. As a result, game developers leave the majority of game testing as a manual process [134, 114]. For example, one of the games being subject of the study by Zheng *et al.* [178] was manually tested by 30 players.

Given the few automated approaches available for quality control in video game development [134], many games are released with unknown problems that are revealed only once customers start playing [156]. The research community is actively working on proposing techniques supporting game developers in testing activities (see *e.g.,* [78, 143, 50, 82, 28, 38, 158]). These techniques simulate the player's behavior in controlled environments, allowing artificial agents to explore game functionalities and identify bugs.

Previous work started to highlight the value of gameplay videos in identifying bugs and encouraging players to submit relevant screenshots or videos along with bug reports [77, 150, 17, 98, 80]. However, despite the large amount of gameplay videos available, they remain a greatly unexplored but powerful source of information for identifying and understanding game problems. Since many

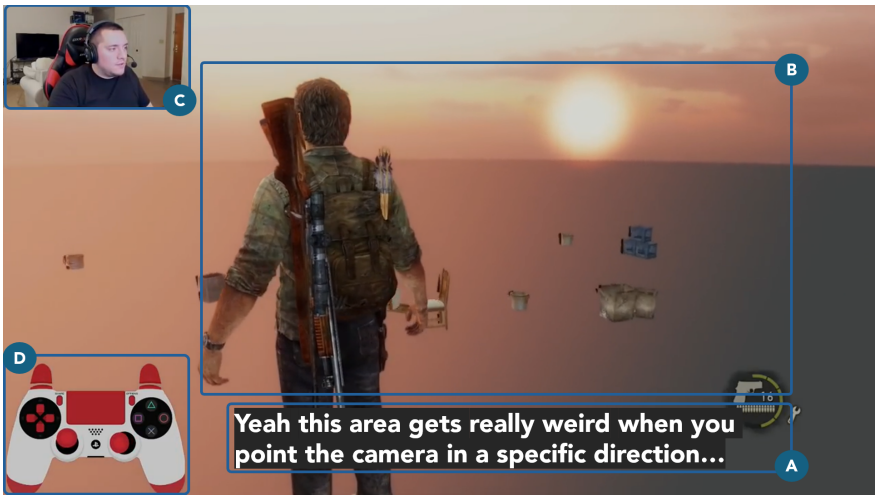


Figure 1.1: Example of gameplay videos where the streamers highlight the issues

streamers daily publish hours of gameplay videos, it is very likely that some of them experience such issues and leave traces of them in the uploaded videos, providing direct proof of several kinds of issues in action. For example, a gameplay video on the game *The Last of Us*[1] shows several problems experienced by the player following a specific sequence of actions. Automatic extraction of meaningful information from these videos could provide game developers with valuable insights, supporting them in the testing phases [98]. Analysis of these gameplay videos offers the possibility of identifying problems occuring during gameplay sessions, without the constraints of controlled test environments [150]. In addition, to simply revealing bugs, this analysis can reconstruct the precise sequences of actions that led to these problems, offering valuable insights into the behavior of the game under real-world conditions [80].

[1] `https://youtu.be/EFit0yoC8Ck?t=1884`

Fig. 1.1 shows an example in which developers can extract meaningful informations through the analysis of gameplay videos,[1] in order to automatically identify problems and define bug reports.

One of the most significant sources of information are the live comments provided by streamers through which they share their gaming experience. These comments are important because they capture players' instant reactions to in-game events and provide context to the game experience. For example, as we can see in Section A of Fig. 1.1, real-time comment highlights the presence of a problem that the streamer encountered during the game session. In detail, the streamer says "this area gets really weird when you point the camera in a specific direction [...]", in this way the streamer highlights the presence of unexpected game behavior. For this reason, the comments might help developers identify issues and thus becomes a crucial element for quality assessment and improvement.

Once a problem is highlighted, it is important to understand where it is occurring and what is happening. Through the analysis of section B in Fig. 1.1, it is possible to identify the gameplay context and the specific problem occurring. In detail, we can see that the streamer comes across a visual glitch, where the background disappears and elements of the environment appear to float.

The analysis of Section C in Fig. 1.1 allows to extract information from the streamer's face that can provide insights about the player's emotions during gaming sessions. For developers, observing these reactions can offer insights into how players feel when encountering specific scenarios, bugs, or gameplay mechanics. Emotional cues, such as frustration, surprise, or amusement, can help identify areas of the game that may need improvement or adjustment to enhance the overall player experience. In this case, the streamer appears to be showing a reaction of confusion, curiosity or surprise, as he encounters the graphical glitch displayed in section B. This expression suggests that the streamer is experiencing something unusual or unexpected.

Finally Section D in Fig. 1.1 displays a visual representation of controller input, showing the specific buttons or movements that the player is performing in real time. This overlay is particularly useful for analyzing the correlation between the player's actions and the game's responses. For example, by examining this section, developers can see exactly what inputs the player used to navigate the game or to trigger specific events, such as the camera movement that led to the visual glitch in section B. This input data are important for debugging, as it helps reproduce the conditions under which specific problems occur, providing a clear link between the player's actions and the game's behavior.

However, extracting meaningful information from these features poses several challenges. Streamers often use slang or informal language that can differ significantly from the terminology used by developers, making it difficult to identify keywords that highlight a problem. For example, a streamer might use the term "bugged out" to describe a graphics problem in which textures do not load properly, while developers might refer to the same problem as a "rendering error". This difference in terminology can create problems in identifying the specific type of issues. Visual analysis of game videos also requires the exclusion of non-game sections, such as advertisements or streamer interactions, and irrelevant elements such as discussion chat, to focus exclusively on game content. Analyzing streamers' facial expressions is challenging due to the lack of an objective reference point (*i.e.,* "oracle") that measures the streamer's level of engagement. Finally, when examining input-action sequences, it is important to note that different inputs can sometimes trigger the same outcome. This highlights the need for precise frame-by-frame analysis of both player inputs and corresponding gameplay to accurately identify and replicate the issue.

1.3 Thesis Objectives and Contributions

We formulate our thesis statement as follows:

> *"The automated analysis of meaningful information in gameplay videos can be adopted for enhancing the testing and quality assurance of video games, allowing issues to be identified and helping developers replicate them."*

By mining and analyzing gameplay videos, we aim to identify critical problems known in the literature [156] that may be missed during traditional testing phases. Lewis et al. [95] introduced a significant first step on the analysis of video game problems by creating a structured taxonomy of game bugs through the gameplay videos analysis. The taxonomy was build by identifying common patterns of defects, such as glitches or collision detection. The authors define this taxonomy with the goal of helping developers identify problems in video games and to support testers in building a more systematic approach to testing. Truelove et al. [156] extend the taxonomy by adding more detailed categories of issues. The authors provided a detailed classification of the most common issues and their severity in game development, many of which can go undetected during traditional testing phases (*e.g.,* logic, performance, balance, graphics).

In this thesis, we rely on the existing taxonomies by focusing on identifying the most relevant bug categories to identify key problems in video games. The first step is to identify and define feature extraction methods to extract meaningful information from gameplay videos.

As a first contribution we introduced GELID (**G**am**E**p**L**ay **I**ssue **D**etector), an automated approach that extracts meaningful segments of gameplay videos where streamers report problems. In detail, taken as input a set of gameplay videos related to a specific video game, the approach (i) breaks them down into meaningful segments that might contain bug reports, (ii) automatically

distinguishes informative from non-informative segments, also determining the type of problem reported (*e.g.,* logic or presentation), (iii) groups them according to the "context" in which they appear (*e.g.,* level or game area), and (iv) groups fragments related to the same specific problem (*e.g.,* game crashes). Although GELID is able to identify the presence of problems in gameplay videos, the results of our empirical validation of such an approach highlights the significant challenges in classifing issues related to game balance and performance.

This complexity is a result of the fact that both performance and balance require an understanding of the dynamic interaction between the player and the game environment, which subtitles alone cannot adequately represent. Therefore, more specialized techniques are required to address these types of issues effectively. Performance problems often manifest themselves as stuttering events *i.e.,* drops in frame rate. The relevance of stuttering events for game developers is well-known. Politowski *et al.* [120] reported that the developers partially automated game performance testing for two out of the five games considered in their study. Naughty Dog reportedly employed specialized profiling tools [153] while developing and testing *The Last of Us* for detecting stuttering events. To support developers in the identification of stuttering events, we introduce HASTE (**H**inter for g**A**me **ST**uttering **E**vents). HASTE splits gameplay videos into visually coherent slices and automatically classifies each extracted slice as *gameplay* or *non-gameplay* (*e.g.,* game settings, advertisement) using a machine-learning model. Given a gameplay video slice, HASTE highlights the relevant game-related parts of the frame, and finally checks each pair of consecutive frames and it identifies stuttering events in which the relevant pixels (*i.e.,* the ones depicting the game) are (almost) identical.

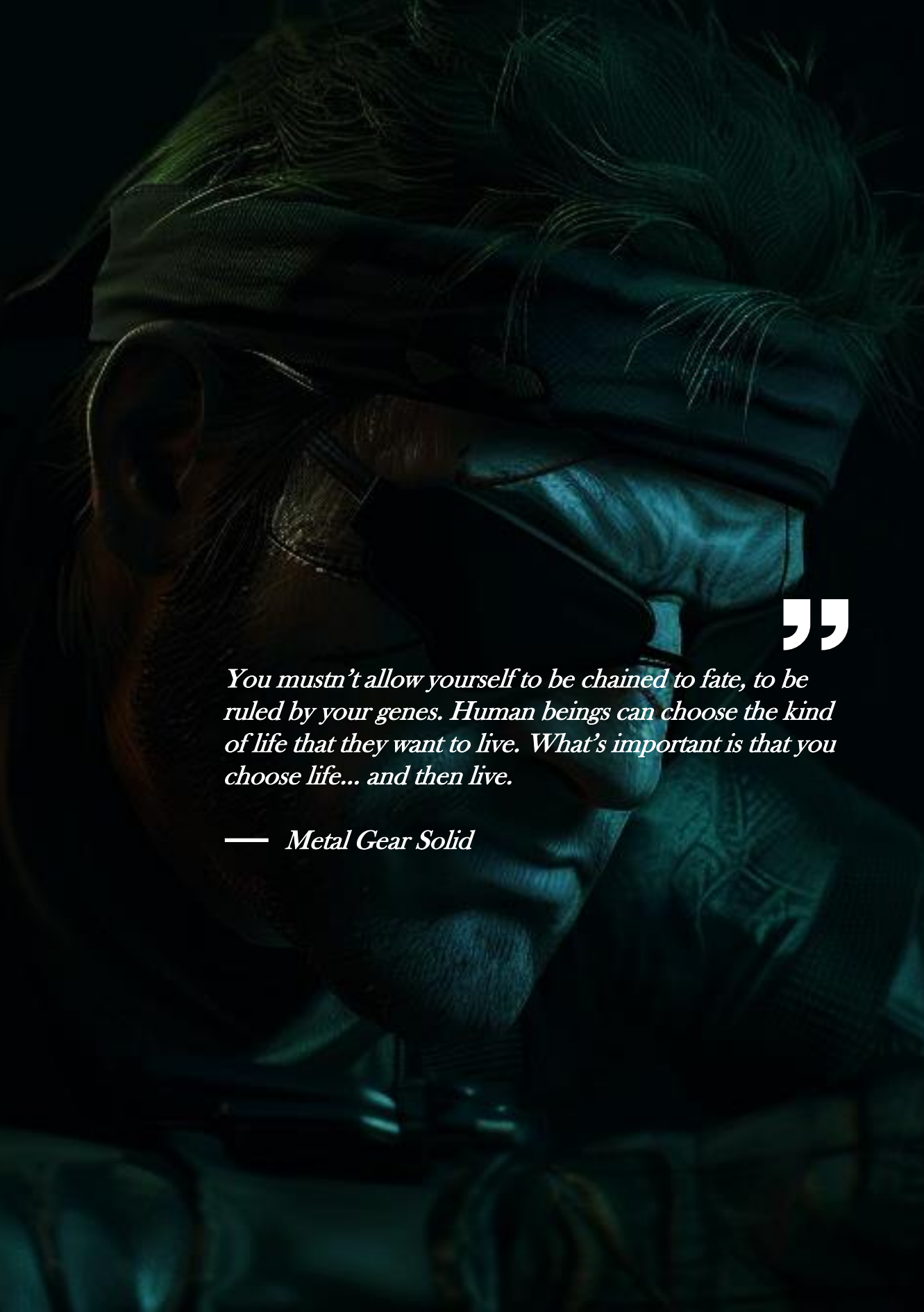
Balance-related problems might manifest themselves as lowered engagement of the player. The facial expression in the recorded face of the streamers can be exploited to achieve this goal. Recent studies have shown that these solutions can effectively evaluate engagement in video games [34, 85]. While such specialized approaches show promising results, their capabilities have only

been assessed on a subjective interpretation of the video content rather than on the direct feedback from the players themselves. Based on this limitation, we conducted a study aimed at assessing to what extent the real engagement can be captured by state-of-the-art approaches. We simulated real-time game sessions with players aimed at collecting a new dataset that provides the *real* level of engagement experienced by the players during gameplay sessions where we filmed their facial expressions and simultaneously recorded their gaming sessions. While there are approaches able to identify the parts of the gameplay where issues are documented, there is still an open problem to face: *How to replicate the sequence of actions that led to the manifestation of the issue?*

In the literature, Intharah and Brostow [80] introduce DeepLogger in order to reproduce issues through the anlysis of gameplay videos. However, DeepLogger presents some limitations in handling input types. Although it can capture discrete inputs (*i.e.,* keys pressed or not pressed), it does not currently support the extraction of continuous inputs, such as joystick movements or mouse positioning. This limitation leaves continuous input extraction as an open problem and limits DeepLogger's ability to predict user input logs only to games for which training data on discrete inputs are available. To overcome these constraints we present RePlay, an approach to extract the sequence of actions performed by a player to replicate a given gameplay through the analysis of gameplay overlays. The main assumption is that gameplay videos show on-screen gameplay actions. RePlay considers both discrete and continuous inputs. Our preliminary work shows that even a simple approach like the one we propose can replicate some gameplay videos (for a specific game).

1.4 Thesis Structure

The remainder of this thesis is organized as follows. In Chapter 2, we first provide an overview of video game testing, explaining the traditional techniques used from game developers. We then discuss problems in games identified in a taxonomy by mining of gameplay videos and existing techniques in the literature that enhance video game tesing through analysis of gameplay videos. In Chapter 3, we introduce GELID, an approach that automatically extracts meaningful segments of gameplay videos in which streamers report issues, and provide an analysis of the various types of issues present in video games by leveraging subtitle-based extraction from gameplay videos. Then, in Chapter 4 we present HASTE an approach for the automatic detection of stuttering events in gameplay videos that can be exploited to generate candidate bug reports. Given the importance of game balance and its impact on player engagement, in Chapter 5 we describe an experiment conducted with players to collect a new dataset. This dataset captures the *real* level of engagement experienced by players during gameplay sessions, allowing for a more accurate analysis of how game balance affects player enjoyment. The Chapter 6 introduce RePlay, simple approach which leverages the on-screen controls overlay available in some gameplay videos. Finally, in Chapter 7 we conclude this thesis, providing a summary of the contributions and the achieved results.



You mustn't allow yourself to be chained to fate, to be ruled by your genes. Human beings can choose the kind of life that they want to live. What's important is that you choose life... and then live.

— *Metal Gear Solid*

CHAPTER 2

Background and Related Works

Contents

2.1 Background: Video Game Testing

The large efforts that game developers invest in the game development process do not always allow them to discover or fix all the bugs in a game before releasing it to the market. Different from traditional software, video game development involves both functional elements, such as mechanics, and non-functional aspects, such as user experience [87, 114]. The quality assurance of video games has been extensively explored with a focus on analyzing the differences between traditional software development and video games development

[107, 134, 12]. Video game software testing focus not only on the common aspects of software in general, but also track and investigate problems that might be related to game balance, game physics, and entertainment aspects to ensure the quality of video games and the success of the testing process [134].

Video game testing is a crucial phase in the game life cycle: Ensuring functionality and user satisfaction before the game is released [12]. The testing process typically begins with test planning and design, in which testers define the specific aspects to be evaluated, such as new features or fixes in a build of the game [141]. The core of the process is test execution: testers play the game more than once, identifying bugs, defects, and problems [92]. They document their findings in detailed bug reports, which include steps to reproduce them. Once the developers have fixed the problems, testers run regression tests to ensure that the bugs have been fixed and that no new issues occur [15, 92].

Another key aspect is beta testing, in which selected players test the game under real-world conditions to provide feedback on game mechanics and balance. This phase is important for identifying problems not identified during internal testing. In addition, testers may engage in specialized forms of testing, such as compatibility testing, to verify the game's performance on different hardware configurations, and multiplayer testing, to ensure the stability of network functionality [92].

2.2 Related Works

Given the complexity of the video game testing process, several approaches have been proposed in the literature to support video game testing activities through new techniques [169, 22, 176]. Based on the advances in Deep Learning (DL) approaches for game testing, Wilkins et al. [169] conceptualize the visual bug detection problem as an anomaly detection problem. The authors introduce a Siamese state network specifically designed for this purpose. The al-

gorithm is evaluated on a series of Atari games, demonstrating its effectiveness in identifying anomalies. This anomaly detection solution requires significantly less data and training than other DL approaches, making it a promising direction for efficient and effective detection of visual bugs in games. Bergdahl et al. [22] present a framework for automated game testing that leverages Deep Reinforcement Learning (DRL) to improve test coverage and enables the discovery of unexpected game mechanics, exploits and bugs in various types of games. Zhang and Smith [176] introduce a new search system that allows users to locate specific moments within video games through natural language queries, solving the lack of direct search within the content of interactive games. Using computer vision and natural language processing, the system semantically matches visual and auditory content, allowing moments in narrative games to be retrieved based on descriptive queries.

Politowski et al. [122] performed a literature review on video game automated testing. The literature results show a rise in research topics related to automated video game. Politowski et al. [122] based on the results applied an online survey with video game developers in order to investigate how suitable automated testing techniques (from the academic literature) are for game developers. The survey results show that game developers are doubtful about using automated agents to test games. For this reason, they highlighted the need to develop new testing techniques to address the challenges posed by games development.

In recent years, there has been growing interest in the use of video analysis for supporting Software Engineering tasks [124, 19, 20, 172, 101]. Ponzanelli *et al.* [124] introduce CodeTube, a Web-based recommender system that analyzes the contents of video tutorials and is able to provide, given a query, cohesive and self-contained video fragments, along with related Stack Overflow discussions. Also, video-based bug reports are becoming increasingly popular for mobile applications [174, 57, 90]. Feng *et al.* [58] introduce CAPdroid, an automatic approach that use image processing and convolutional Deep Learning models to segment bug recordings, infer user action attributes, and generate subtitle

descriptions. Krieter *et al.* [86] present a method for analyzing mobile application usage in detail by generating log files based on mobile screen output.

The large amount of gameplay videos continuously produced and publicly released by many gamers on platforms such as Twitch [1] and YouTube [2] could be helpful to developers: Sometimes, gamers indirectly report issues while they play [63, 89].

Video games can encounter a wide range of challenges. In Section 2.2.1 we discuss the existing taxonomies that define the problem categories identified in video games. Also, in the next Section 2.2.2 the discussion mainly focuses on techniques aimed at analyzing gameplay videos through the extraction of useful information that aim to support developers for different purposes.

2.2.1 Taxonomies of Video Game Issues

Video games can suffer from a vast variety of problems. Lewis et al. [95] described the gameplay videos as “a rich resource.” They were the first to realize the possible usefulness of analyzing gameplay video. In detail, Lewis et al. [95] introduced a taxonomy of video game bugs based on a collection of gameplay videos, when the phenomenon was not yet so common.

Truelove et al. [156] extend the taxonomy intoduced by Lewis et al. [95] in order to identify the reported issues within video games. In their taxonomy, the authors reports 20 different kinds of issues. At the start, their goal was to label bug fixes based on the 11 types of bugs identified Lewis et al. [95]. However, the authors found that some fixes did not match any of the existing types. Based on this, the authors define basic descriptions of the initial 11 types of bugs present in the previous taxonomy [95]. When the authors labeled the bug fixes, if they found one that didn’t fit any of the existing labels, they created a new one. The authors labeled a total of 12,122 bugs using this taxonomy, drawn from 723 different updates notes.

Table 2.1 shows the main types of bugs found in update notes, categorized according to the taxonomy developed by Truelove et al. [156]. This classification outlines various issues that may arise during video games development, with a particular focus on post-release bugs, providing an overview of the key categories of defects observed in video games. The nine new types of bugs introduced in are marked in the table with an asterisk (*).

Recently, Butt et al. [29] conducted a multivocal literature review (MLR) that focused on analyzing bugs identified within the game development industry. The aim of this study is to provide a taxonomy of bugs for games that will help developers develop bug-resistant games, testers design and execute test cases for finding defects, and researchers evaluate approaches to game testing. In addition, they observed that manual approaches to game testing are still widely used. Only one of the approaches addresses sound bugs, while game balancing and ways to incorporate machine learning into game testing are trending in recent literature. Most game testing techniques are specialized and dependent on specific platforms.

2.2.2 Mining of Gameplay Videos

We discuss techniques proposed in the literature for quality assurance in video games through the analysis of gameplay videos and playtesting.

Generic Issues

A strategy to support developers in finding issues in video games consists in analyzing gameplay videos released by players. Since the phenomenon of publishing gameplay videos is relatively recent, only a few studies have emphasized their value for identifying problems in video games. Identifying quality issues in video games is a relevant issue explored in the literature [18, 134].

Table 2.1: Taxonomy of bugs identified by Truelove et al. [156]

Category	Description
Action	Errors in the ability/inability to perform actions.
Game Graphics	A certain visual aspect of the game world is being rendered incorrectly.
Artificial Intelligence	An NPC or AI does not behave in the intended manner.
Implementation Response	There are errors regarding the game's interactions with hardware or base software that ends up affecting performance.
Audio*	Game audio is not playing correctly.
Information	Information about the game world is not conveyed to the player correctly.
Bounds	An object exists outside the intended game space, such as being outside the game map or in a location that should not be accessible.
Interaction Between Object Properties*	Two or more object properties do not behave properly when interacting with each other.
Camera*	There is an issue with game camera through which the player views the game world.
Interrupted Event	An action in the game that was previously operating is terminated abruptly against expectations, or the action was not interrupted when it should have been.
Collision of Objects*	Objects do not behave properly when they collide or make contact with each other.
Object Persistence*	Game objects do not correctly enter or exit the game world.
Context State	Objects do not enter and exit states properly.
Position of Object	An object is not in the correct position or orientation within the game space over time.
Crash*	The game crashes or is otherwise forced to close.
Triggered Event*	Improper triggering of game events (Event A causes Event B).
Event Occurrence	Discrete events do not occur at the correct rate or order.
User Interface*	The appearance/position/behavior of UI elements are incorrect.
Exploit*	Players are able to improperly and unfairly gain game benefits through behavior unintended by developers.
Value	An in-game variable is not set to the correct value.

Video screen recordings are becoming increasingly common for users to communicate problems to developers because they effectively convey what the user sees [23, 47]. Similarly, the analysis of gameplay videos can play a key role in game testing. Through the analysis of gameplay videos, developers can identify and document software bugs that may not be easily detected by traditional testing methods.

D'Angelo et al. [49] propose PlayMyData, a dataset consisting of 99,864 cross-platform games containing relevant metadata for each game, *e.g.,* description, genre, rating, game video URLs, and screenshots. They present the most comprehensive dataset in the industry that can be used to support various automated tasks in Software Engineering. PlayMyData can be used to promote analysis based on the provided multimedia data.

Moreno-Ger et al. [106] present a new methodology to facilitate usability testing of games. The methodology suggests a list of improvements based on an in-depth analysis of many registered games. This approach is especially useful for playtesting games, as it provides a more systematic and focused approach to generating improvements.

Luo et al. [99] propose an approach for automatic event retrieval in e-sports game videos that requires manual data labeling, a fixed set of classes (events), and training of new models. In this study, the authors present a combined solution to reduce the computational resources and time required to apply a convolutional neural network (CNN) to extract events from e-sports game videos. This solution consists of techniques to train a CNN faster and methods to perform predictions faster.

Some works targeted the automated generation of a comprehensive description of what happens in gameplay videos (*i.e.,* game commentary). Examples of these works are the framework by Guzdial et al. [73] and the approach presented by Li et al. [96] modeling the generation of commentaries as a sequence-to-sequence problem, converting video clips to commentary. Lin *et al.* [97] conducted an in-depth study of gameplay videos posted by players

on the Steam platform aiming at automatically identifying the ones that report bugs. They observe that naïve approaches based on keywords matching are inaccurate. Therefore, they propose an approach that uses a Random Forest classifier [77] to categorize gameplay videos based on their probability of reporting a bug. The authors manually labeled 96 gameplay videos to train and test a Machine Learning-based approach for automatically detecting gameplay videos that report functional bugs. Lin et al. [97] rely on Steam[1] to find videos related to specific games. While Steam is mainly a marketplace for video games, it also allows users to interact with each other and share videos. On a daily basis, for 21.4% of the games on Steam, users share 50 game videos, and a median of 13 hours of video runtime Lin et al. [97]. Still, their approach works at video-level, and manually watching long gameplay videos classified as buggy still requires a considerable manual effort since a whole video can even last several hours. Also, they only distinguish bug-reporting videos from non-bug-reporting ones, without a more specific classification regarding the type of issue reported (*e.g.,* glitch or logic bug). This highlights a gap that needs to be filled to further help developers classify the video segment based on the type of problem identified.

Truelove et al. [157] intoduce an automated approach based on machine learning to identify whether a segment of a gameplay video contains occurrences of bugs. Thier approach is designed to process video segments regardless of the contents of the transcript text. However, even in this case the authors are unable to classify the type of issue. On the same line of research, an approach to generate automatic comments for videos by using Deep Convolutional Neural Networks was presented by Shah et al. [140].

Taesiri *et al.* [150] present a search method that retrieves relevant video from large archives of gameplay videos related only on game physics problem. Also, Taesiri et al. [151] presented an approach that depends solely on video content and does not require external information, such as video metadata, for bug dentification. Leveraging the zero-shot transfer capabilities of the language-

[1] `https://steamcommunity.com/`

image contrastive pretraining model, the proposed approach does not require any labeling or training of the data. The approach shows promising results in in-depth analysis of simple and compound queries, indicating that the method is useful for detecting objects and events in gameplay videos. The results of the study demonstrate the potential of the approach for applications, such as the creation of a video search tool tailored for identifying video game bugs, which could be of great use to quality control teams in finding and reproducing bugs. Motivated by the goal to identify bugs to improve the overall quality and player experience in video games Azizi and Zaman [17] introduce AstroBug, a framework designed to assist bug detection and testing in game development. The framework analyzes gameplay frames and identifies instances that deviate from the expected behavior, reporting the presence of bugs, using ConvLSTM networks. To evaluate the effectiveness of the framework, the authors conduct experiments on two FPS games, WOB and Unity FPS microgame. The results demonstrate the effectiveness of the framework in detecting bugs and show its potential for improving the bug detection process in game development. Ling et al. [98] present a method using Deep Convolutional Neural Networks (DCNN) to detect the most common glitches in video games. This study focuses on detecting graphical texture anomalies.

Accessibility features, are commonly used by gamers but are challenging to test manually due to the vast variety of games and the differences in how subtitles are displayed. Based on this, Gauk and Bezemer [64] introduce an automated approach ("EchoTest") to extract subtitles and spoken audio from a game video, convert them to text, and compare them to detect discrepancies such as typos, desynchronization, and missing text. EchoTest can be used by game developers to identify discrepancies between subtitles and spoken audio in their games, allowing them to better test the accessibility of their games.

Performance Issues

Developers rarely focus on maintaining older games, which in some cases leads to an impossibility of launching them on newer systems or to a decremetering of performance [171]. For this reason, Wrześniewska and Skublewska-Paszkowska [171] analyze the performance of video games on selected operating systems. All tested games were visually compared to determine if their appearance and available graphics options were the same on the different systems.

In relation the identification on the performance problem the only approach that aims a study this issues is RELINE, defined by Tufano *et al.* [158]. RELINE is the first technique to automatically detect game areas in which the frame-rate drops (*i.e.,* areas that might trigger stuttering events). To do this, the authors trained a RL-based agent able to play a given game with the aim of (i) achieving the best results in the game, like a player would do, and (ii) minimizing the frame-rate. While RELINE is meant to be executed by developers before the release, there is no approach that supports them in beta-testing and after release, when gameplay videos from human players are available.

Balance Issues

Balancing issues in video games are very difficult to identify through the traditional testing and presents challenges with automated testing [118, 132, 115]. Pfau et al. [118] applied Deep Player Behavior Modeling to generate models that could reproduce the playtesting strategy. The main goal of the tests was automatic balancing of game difficulty. However, trained AI agents, which emulate the actions of individual players, can also be used for game exploration and detection of bugs and problems within the game. Roohi et al. [132] propose an automated playtesting method to predict the behavior and experiences of human players. This approach leverages the integration of Deep Reinforcement Learning and Monte Carlo Tree Search to enhance automated

playtesting effectiveness. Balancing video games is a highly debated topic, especially among online game players. Whether a game is balanced greatly influences player satisfaction and abandonment rates and players. In particular, different ideologies of balance can lead to worse gaming experiences than actual imbalances. Pfau and Seif El-Nasr [115] present a fine-grained study of Guild Wars 2 community attitudes about balancing factors. In addition, the authors introduce a player-driven quantitative tool to approximate the closest balancing configurations that could optimize player experience and satisfaction. Based on previous work, Pfau and Seif El-Nasr [116] claim that conceptions about the definition of balancing often diverge between industry, academia, and gamers, and different balance design can lead to gamers' experiences that are worse than the actual imbalances. In the study the authors collect game balancing concepts from industry and academia and introduce a player-driven approach to optimizing player experience and satisfaction. Politowski et al. [123] define an approach to integrate game testing to balance video games with autonomous agents. They propose a systematic way to assess whether a game is balanced by (i) comparing difficulty levels between game versions and game design problems and (ii) skill or luck demands.

Several studies targeted the assessment of engagement in video games [31, 167, 26, 60, 13, 83, 67, 175, 139]. These studies differ for the methodologies employed in the measurement and for the insights they seek to obtain about the players' experience. Previous work [67, 175] assessed engagement by relying on electrocardiogram (ECG), facial electromyography (fEMG), galvanic skin response (GSR), and breathing frequency. Using these features, researchers managed to predict players' emotions in real time. However, the collection of such signals requires invasive equipment, and does not allow to scale the measurement of engagement to a large players base, since it still requires the direct involvement of humans in playtesting.

The detection of facial expressions recognition is an alternative way of assessing the experience of the player, since it requires a much less invasive

equipment (a camera) and, as previously explained, could be applied to mine information from gameplay videos. Moniaga *et al.* [105] present a dynamic game balancing system that adjusts game difficulty according to the players' facial expressions, enhancing the gaming experience. Differently from our work, their focus is to correlate four emotions/expressions (*i.e.,* angry, frustration, smirk, and smile) to the difficulty level of the game, increasing/decreasing it consequently. Engagement, on which we focus, is a wider concept, since a player may not be engaged while being frustrated for too difficult parts or bored for too easy ones.

Similarly, Kwon *et al.* [91] proposed a framework for automatically assessing the emotions of players during gameplay. They exploit facial expressions to identify users' emotions, including happiness, surprise, sadness, anger, disgust, and fear. Again, while emotions can be correlated to engagement (and, indeed, we use them as some of the features in FFBD), they only tell part of the story, since users may experience some of these emotions (*e.g.,* sadness, anger) both while being engaged or not.

Killedar *et al.* [85] introduced an approach aimed at assessing players' engagement via facial expressions. Such an approach focuses on facial emotions rather than expressions, meaning that the only features used to decide if the player is lowly engaged are seven emotions: neutral, sad, angry, happy, surprise, disgusted, and fearful.

Chen *et al.* [34] introduced FaceEngage, a non-intrusive engagement estimator leveraging user facial recordings during actual gameplay in naturalistic conditions. The authors show the potential of using front-facing videos as training data to build the engagement estimator. They presented FaceEngage system, that captures relevant gamer facial features from front-facing recordings to infer task engagement. FaceEngage was implemented considering two pipelines: an estimator trained on user facial motion features inspired by prior psychological works, and a deep learning-enabled estimator. Based on the dataset previusly introduced by [34], Pan *et al.* [113] propose a multimodal Deep

Learning model that utilizes non-intrusive and non-restrictive multimodal data (facial, pixel, and sound modalities) to automatically estimate the engagement of game streamers. A limitation of the dataset used to train FaceEngage is that it relies on *apparent* engagement, manually assessed by external persons, rather than on the *actual* engagement experienced by the players. Finally, in a related research thread, Schønau-Foghanno *et al.* [139] mapped the causes for players' involvement during the gaming experience into several types, including intellectual, physical, sensory, social, narrative and emotional.

The state-of-the-art in engagement measurement recognizes the importance of affective states, such as personality traits and emotions, as proxies to properly assess engagement [110, 94, 48]. In recent years, Affectiva [9], has created a huge emotion database, consisting of data from 10 million consumer responses to over 53,000 advertisements in 90 countries. This vast repository of emotion data not only facilitates the training and testing of data-driven techniques algorithms, but also provides benchmarks for advertisers. By comparing the performance of their ads with others in the database, advertisers gain valuable insights into how consumers respond to different types of brand content [8]. Affectiva enables to measure the level of engagement through the weighted sum of a set of action units computed by looking at the facial muscles activation. In particular, it provides an engagement score between 0 and 100. Score engagement it is the weighted sum of the following facial expressions: Inner and outer brow raise, Brow furrow, Cheek raise, Nose wrinkle, Lip corner depressor, Chin raise, Lip press, Mouth open, Lip suck, Smile. Although there are other engagement tools built on top of Affectiva (*e.g.,* IMotion [79, 149]), Affectiva is distributed as a free toolkit representative of the state of the art [35].

Replicating Game Issues

In game development, bug detection and reproduction are critical to productivity. Automation of these tasks has long been an industry goal to reduce

quality management costs. Pinto Gomez and Petrillo [119] presented a new detection technique that leverages data available from the game engine and a time-saving bug reproduction tool.

Previous work defined approaches to support developers in testing video games, aiming at identifying functional and nonfunctional unexpected behaviors before the release [78, 112, 104]. Iftikhar *et al.* [78] proposed a model-based testing approach for performing black-box testing of platform games. A crucial challenge of such approaches is that some issues might only occur after executing a specific set of moves, which requires a certain level of intelligence. Therefore, Deep Reinforcement Learning (DRL) has been explored to provide competitive and intelligent "human-like" support. As a more general approach, Paduraru *et al.* [112] introduced a similar tool named RiverGame, used to perform game testing based on artificial intelligence. This tool lets the user automatically test their products from different points of view: the rendered output, the sound played by the game, the animation and movement of the entities, the performance, and various statistical analyses. Mnih *et al.* [104] used gameplay videos to train an AI agent to play Atari games. In this case, the agent was trained by watching gameplay videos and then could play the game by itself. Zheng *et al.* [178] present Wuji, an approach for automatically finding *crash*, *stuck*, *logical*, and *balance* problems by using evolutionary algorithms, DRL and multi-objective optimization. Wu *et al.* [173] defined an approach based on DRL to perform regression testing, while Ariyurek *et al.* [16] defined synthetic and human-like agents, based on a combination of DRL and Monte Carlo Tree Search (MCTS). Ahumada and Bergel [11] introduced an approach based on genetic algorithms to allow developers to reproduce functional bugs by reconstructing the sequence of actions that lead to a specific faulty state of the game. White *et al.* [172] show how video analysis can be used to increase the replicability of bugs experienced in Android apps. The authors present an approach for automating the process of reproducing a bug.

Some recent studies [173, 178, 16, 158] used Deep Reinforcement Learning to support developers in finding issues in video games (*e.g.,* performance-related). Pfau *et al.* [117] introduced ICARUS, a framework for autonomous video game playing, testing, and bug reporting from which it is possible to extract information about the problems identified (*e.g., crash* and *stuck* events).

Intharah and Brostow [80] introduce DeepLogger, an innovative approach that uses a Convolutional Neural Network (CNN) to analyze gameplay videos and automatically extract user input data. DeepLogger takes into account discrete inputs (*i.e.,* keys pressed or not pressed) and leaves the extraction of continuous inputs (*e.g.,* joysticks or mouse) as an open problem. Thus, this network can only predict user input logs for a game where training data is available. On the other hand, we highlight the lack of an approach that considers both discrete and continuous inputs. This comes at the cost of relying on input overlays, which are only available on a subset of game videos.

”

What is bravery, without a dash of recklessness?

— *Dark Souls*

CHAPTER 3

Detecting Video Game Issues

Contents

3.1 Introduction

In video game development, the limited availability of automated quality control methods often leads to the release of games with undiscovered problems that are only revealed when players start playing. For example, a gameplay video on the game Cyberpunk 2077[1] shows that the game crashes as soon as the player performs a specific action. These videos are a valuable resource for identifying and fixing issues after release.

Lin et al. [97] defined an approach able to automatically identify videos containing bug reports. However, such an approach mostly relies on the video metadata (*e.g.,* its length) and it is not able to pinpoint the specific parts of the video in which the bug is reported. This makes it unsuitable as a reporting tool for game developers, especially when long videos, which are not uncommon, are spot as bug-reporting.

In this chapter, we introduce GELID (GamEpLay Issue Detector), an automated approach that aims at complementing the approach by Lin et al. [97] by (i) automatically extracting meaningful segments of gameplay videos in which streamers report issues, and (ii) hierarchically organize them. Given some

[1] https://youtu.be/ybvXzSLy9Ew?t=1448

gameplay videos as input, GELID (i) partitions them into meaningful segments that might contain bug reports, (ii) automatically distinguishes informative segments from non-informative ones by also determining the type of reported issue (*e.g.,* bug, performance-related), (iii) groups them based on the "context" in which they appear (*i.e.,* whether the issue manifests itself in a specific game area), and (iv) clusters fragments related to the same specific issue (*e.g.,* the game crashes when a specific item is collected).

We evaluate the four components of GELID in isolation. We first extract training data for the machine learning model we use to categorize segments (step 2 of GELID). To this end, we used the approach by Lin et al. [97] to identify candidate videos from which we can manually label segments in which the streamer is reporting an issue. Then, we ran GELID on a set of real gameplay videos and validate its components. First, we manually determine to what extent the extracted segments are usable, by annotating their *interpretability* (*i.e.,* they can be used as standalone videos) and *atomicity* (*i.e.,* they can not be further split). Second, we validate the categorization capabilities of GELID, both when trying to distinguish non-informative segments from informative ones (binary classification) and when trying to pinpoint the specific issue among *logic*, *performance*, *presentation*, *balance*, and *non-informative* through a multi-class classifier. Then, we compute typical metrics used to evaluate ML models (*i.e.,* accuracy and AUC). Finally, we evaluate to what extent the clusters identified in terms of context and specific issues are similar to the manually determined ones using the MoJoFM metric [164].

The remainder of this Chapter is organized as follows. In Section 3.2, we present GELID and its four components in details. In Section 5.4 we describe the empirical study design, while in Section 3.4 we report the obtained results. In Section 3.5 we discuss the results, while in Section 3.6 we report the threats to validity. Section 3.7 concludes the chapter.

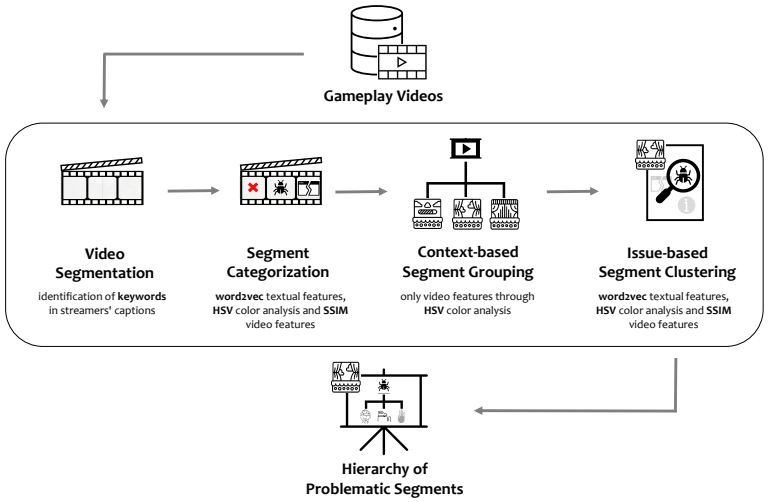


Figure 3.1: The workflow of GELID.

3.2 A Methodology for Gameplay Videos Issues Detection

In this section we describe GELID, an approach that takes as input a set of gameplay videos related to a specific video game and returns a hierarchy of segments of gameplay videos organized on three levels: (i) context (*e.g.,* level or game area), (ii) issue type (*e.g.,* bug or glitch), and (iii) specific issue (*e.g.,* game crashes when talking to a specific non-player character).

Fig. 3.1 shows an overview of the GELID workflow. We describe below in more detail the main steps of GELID.

3.2.1 Video games Issues Taxonomy

We use the taxonomy introduced by Truelove et al. [156] as a base to define the labels we want to assign to the video segments. However, all such labels might be counterproductive since it is likely to observe a long-tail distribution

Table 3.1: Mapping between types of issues identified by GELID and categories from the taxonomy by Truelove *et al.* [156].

Issue Type	Description	Categories [156]
Logic	Issues related to the game logic, regardless of how information is presented to the player.	Object Persistence Collision of Objects Inter. btw. Obj. Prop. Position of Object Context State Crash Event Occurrence Interrupted Event Triggered Event Action Value
Presentation	Issues related to the game interface (graphical- or audio- related).	Game Graphics Information Bounds Camera Audio User Interface
Balance	Detrimental aspects in terms of "fun".	Artificial Intelligence Exploit
Performance	Performance-related issues (*e.g.,* FPS drops).	Implem. Response

(*i.e.,* a few types of issues appear in most of the video fragments, while several other issues are quite rare or do not even appear). Therefore, starting from such a taxonomy, we define macro-categories by clustering similar fine-grained categories. We identified four labels, as reported in Table 3.1: *Logic*, *Presentation*, *Balance*, and *Performance*.

3.2.2 Video Segmentation

The first step of GELID consists in partitioning the video into meaningful segments that can be later analyzed as standalone shorter videos. In the computer vision literature, a similar problem is referred to as "shot transitions

detection" [145]. The aim is to detect sudden changes in the video content. An example of approaches defined to solve such a problem is the one introduced in [152]. Video-related information, however, might not be sufficient to find cuts in gameplay contents. In the context of video segmentation, relying only on scene changes to identify meaningful segments may not be sufficient. Scene changes may be due to various minor factors, *e.g.,* rapid zoom into the viewfinder of a weapon and then back to the general framing of the scene. Such situations do not provide significant information for identifying potential issues. Furthermore, in some contexts, scene changes may not be evident, leading to the creation of very large segments that are difficult to analyse. Let us consider, for example, the gameplay video `https://www.youtube.com/watch?v=_kQIJ2Omy9w`: From 14:10 to 15:56 (mm:ss) there is no shot transition, even though various separate events and actions occur. Moreover, for example, if the game crashes and a shot transition detection approach is used to cut the video, the second in which the crash happens would probably be selected for segmentation. The streamer, however, might need a few seconds to react to such an event by commenting what happened providing useful information for the game developers. Thus, by using shot transitions as cut points, the spoken content related to the issue might be erroneously put in the subsequent segment. To solve this problem, we decided to mainly rely on the spoken content to decide the cut points in the video: The core idea is to get the points in which each subtitle entry (*i.e.,* units of text shown on the screen) begins and ends, slightly shifted by t seconds (where t is a parameter of the approach) to take into account the reaction time of the streamer, and thus consider the video in-between as a segment. As for the shifting operation, given a subtitle entry that starts at second s and ends at second $s + d$ (where d is the duration of the subtitle entry), our approach will extract the video segment between $\max(s-t, 0)$ and $\min(s+d+t, \mathit{video\ length})$. For example, consider the case where we set $t = 5$ and we detect a subtitle entry that starts at 13:45 (mm:ss) and lasts 3 seconds. Our approach will cut the video between 13:40 and 13:52.

We report in Section 5.4 how we tune the t parameter.

As a result, our segmentation approach will implicitly discard some parts of the input video (*i.e.,* the ones in which the streamer is not speaking) and it might put some parts of the video in many segments when $t > 0$ (*e.g.,* for contiguous subtitle entries). Also, it is worth noting that using this strategy might result in a very high number of extracted segments for each video since subtitle entries generally include only parts of a sentence: In subtitles, a given sentence is broken into several entries to allow the watcher to comfortably read each of them. To preliminarily exclude segments that most likely do not contain any piece of useful information and, thus, to reduce the effort for the next step, we use a keyword-matching approach. If at least a relevant keyword is found in the subtitle entry related to a given segment, we consider the segment, while we exclude it otherwise.

To define the list of keywords, we relied on (i) the 12,122 change notes of video games used by Truelove et al. [156] to define the taxonomy of the most frequently encountered problems in video games and (ii) the 996 titles and descriptions of the gamplay videos in the dataset defined by Lin et al. [97]. One evaluator manually extracted, from each instance, a first set of keywords (also composed by more than a word) which were related to issues in video games (*e.g.,* "glitch" or "bug"). As a result of this process, 161 basic keywords were identified the file containing the selected keywords is reported in the replication pacakage [71]. From such keywords, we automatically generated new semantically equivalent keywords to have a broader dictionary. To do that, we first tokenized the keywords and automatically tagged the Part-of-Speech (PoS) by using the spaCy Python package [128]. Then, for each token with its PoS tag, we used both WordNet [103] and SEWordSim [154] to generate both general-purpose and domain-specific synonyms of each word. At this point, for each keyword composed by the sequence of words $\langle w_1, \ldots, w_n \rangle$, we combined all the synonyms of each word and generated the new set of candidate keywords by using the Cartesian product: $\{syn(w_1) \times \cdots \times syn(w_n)\}$.

For example, given the initial keyword “lag”, we generated the candidate alternative keywords “stuttering”, “FPS drop”. From the initial 161 identified keywords, we obtained a total of 207 candidate keywords. Then, two evaluators independently validated the new keywords to discard the ones that were not related to issues in video games. In case of disagreement, they discussed to reach consensus. In the end, we added 96 new keywords, while 111 were discarded. In our analysis, we assessed the inter-rater reliability between the annotators involved in identifying keywords by calculating Cohen’s Kappa coefficient. The obtained results indicate an agreement level of $k = 0.74$. The coefficient value of 0.74 indicates a good level of agreement between the annotators in terms of identifying the keywords. For example, the keyword “crash” generated from “break up” was discarded. Thus, our final list of keywords is composed of 257 keywords.

3.2.3 Segment Categorization

In this second step, GELID aims at categorizing segments based on their content. GELID considers five labels: One for *non-informative* segments (*i.e.,* the ones not reporting issues), and four for *informative* segments (*i.e.,* the ones reported in Table 3.1). Non-informative segments are discarded and not considered in the next steps.

Previous work successfully used machine-learning to solve similar classification problems in the context of mobile app reviews [33, 138]. Such approaches mainly rely on textual features. In our context, we can extract information that could also help to correctly classify segments from video analysis. For example, segments without video might be more likely to be *non-informative*, even if a reader comment is present. Therefore, we include in GELID also video-based features.

More specifically, we extract five sets of features: Three of them only based on the subtitles (*i.e.,* what the streamer says), one of them based on the video

(*i.e.,* what happens in the game), and one of them including the best set of textual features and the set of video-based features.

Textual Features. As for the textual features, we consider Bag of Words (BoW) [177], doc2vec (d2v) [84] and word2vec (w2v) [131]. BoW consists in detecting the dictionary of the training set and using each word of the dictionary as a feature. The value of each feature for a given instance corresponds to the number of times the related word appears in an instance. The number of features directly depends on the training set. In our case, given the training set described in Section 3.3.1, we extracted 2,253 features. The d2v model [84] allows to automatically extract a vector of features for an entire instance (document). Such a model allows to automatically represent a document (sequence of words) as a vector. Specifically, we represent each subtitle string for each identified segment as a vector composed of 40 features since this is the default number of features extracted by such a model [84]. Finally, the w2v model [131] allows to represent a single word as a set of features. Thus, differently from doc2vec, it does not directly work at document-level. To define the features based on w2v, given all the words in a given instance, we extract the vectors through the w2v model and we compute the average of each feature. In this case, we represent each word as a vector of 300 features, again, because the w2v model extracts by default such a number of features [131].

Video-based Features. With video-based features, instead, we mainly wanted to represent to what extent the video contains unexpected frames that could possibly be related to issues. To this aim, given each pair of subsequent frames f_i and f_{i+1}: (i) we compute their structural similarity through SSIM [162], *i.e.,* $s_i = SSIM(f_i, f_{i+1})$; (ii) we extract their HSV histograms using the HISTCMP CORREL function of OpenCV [126], thus obtaining $h(f_i)$ and $h(f_{i+1})$; (iii) we then compute their Pearson correlation coefficient $hsv_i = cor(h(f_i), h(f_{i+1}))$. We use SSIM instead of other image similarity measures because it has been shown that such a metric best captures the similarity of images as perceived by humans [162]. Since such a metric ignores colors but

considers, by default, a black-and-white version of the image, we also use HSV histograms to detect differences in the colors. Finally, given the vectors of values hsv and s for all the frames between 0 and n (number of frames in the video), we aggregate their values and define 12 video based features by computing the mean, median, minimum, maximum, first quartile, and third quartile of both of them. Such features allow us to inform the model about the distribution of such vectors. For example, let us imagine that the game crashed: the frame f_i before the crash is very similar to the previous ones, while the next frame, f_{i+1} is different from f_i. As a result, both hsv_i and s_i will be very high. Two of our features (*i.e.,* the max of both the vectors) will reflect this information.

Given a training set of labeled video segments, we extract the features and train a ML classifier. Given an input (unknown) video segment, we extract the same features used to train the model, given the resulting vector as input to the trained ML model, and obtain the predicted label. We describe in Section 5.4 how we built the training set and how we select the best ML algorithm for this task among Random Forest [77], Logistic Regression, SMO [76], Multilayer Perceptron [130] and IBk [39].

3.2.4 Context-based Segment Grouping

After having collected and categorized segments that contain anomalies (*i.e.,* the ones classified as *informative*, *i.e.,* as *logic*, *presentation*, *performance*, or *balance*), we group them according to their *context*. With "context" we refer to the part of the game (*e.g.,* a specific game level or area) in which the anomaly occurred. This may be helpful to provide the videos to the team in charge of the development of that specific part of the game. Such a step is important for two reasons: (i) Developers analyzing hundreds of videos related to a specific game may experience information overload and this, in turn, would reduce the effectiveness of the video segments filtering step; (ii) Knowing the context in which more anomalies occur allows the developer to identify where attention

needs to be focused to improve the gaming experience.

To achieve this goal, we rely on video information: The assumption is that videos with similar frames regard, most likely, the same context. First, we extract the key frames from each segment by using the Video-kf Python package [129]. Then, we define a summary frame of the whole segment by computing a pixel-by-pixel average of the previously identified key frames. Such a frame will roughly represent the content of the segment and, ideally, it can allow to visually represent the game area. We use a clustering algorithm to group summary frames (and, thus, the associated segments). More specifically, given a distance function between two images (summary frames, in our case), we define a distance matrix which contains the distances between each couple of summary frames and use it to cluster them.

We test two similarity metrics (which are also used for computing the video-based features in the previous step): Structural similarity (SSIM) [162], computed on each pair of summary frames, and the correlation between the HSV histograms extracted from each pair of summary frames. Note that both of them are *similarity* metrics, while clustering algorithms require to indicate the *distances* between instances. Since both of them are bounded in the range $[0, 1]$, we simply transform them in distance metrics by computing $1 - s$ (where s is the value of the similarity metric). Since the number of scenes is not necessarily known *a priori*, we use a non-parametric clustering technique. We describe in Section 5.4 how we select the best clustering algorithm between the two we tested, *i.e.,* DBSCAN [55] and OPTICS [14], and the best distance metric between SSIM and HSV histogram correlation.

3.2.5 Issue-based Segment Clustering

A set of video segments of the same kind (*e.g.,* bugs) and reported in the same context might still be hard to manually analyze for developers. For example, if 100 segments report bugs for a given level, developers need to

manually analyze all of them. It might be the case, however, that most of them report the same specific bug (*e.g.,* a game object disappears). To reduce the effort required to analyze such information, we cluster segments reporting the same specific issue. This would allow developers to analyze a single segment for each cluster to have an overview of the problems affecting the specific area of the game.

To achieve this goal, we represent the instances (*i.e.,* video segments) by using both textual and image-based features and, as in the previous step, we use non-parametric clustering to create homogeneous groups. Textual features can help grasping the broad context (*e.g.,* objects disappearing or anomalous dialogues). Image-based features can help finding visually similar problems (*e.g.,* in the case of glitches). To this aim, we represent each instance (video segment) using the set of features from the categorization step that allows to obtain the best results for that task (as we report in Section 3.4). Differently from the previous step, indeed, we do not pre-compute the distance matrix. This allows us to test this task not only with DBSCAN [55] and OPTICS [14], but also with Mean Shift [61], which, differently from the previously-mentioned algorithms, does not allow to directly use a distance matrix.

Also in this case, we describe in Section 5.4 how we select the best clustering algorithm among them.

3.3 Empirical Study Design

The goal of our study is to evaluate the effectiveness of the four steps of GELID, *i.e.,* (i) extraction of meaningful video segments from gameplay videos (ii) accuracy in categorizing extracted video segments, (iii) capability of clustering video segments about the same gameplay area, and (iv) ability to correctly cluster segments reporting the same specific issue. The context of the study consists of a total of 275 gameplay videos.

Our study is steered by the following research questions (RQs).

> RQ_1: *How meaningful are the gameplay video segments extracted by GELID?*

The first RQ aims at evaluating the quality of the segments extracted by GELID from gameplay videos in terms of their *interpretability* and *atomicity*. It aims at evaluating the "video segmentation" step described in Section 3.2.2.

> RQ_2: *To what extent is GELID able to categorize gameplay video segments?*

With this second RQ we want to understand which features and which classification algorithm allow to train the best model for categorizing gameplay video segments both in two classes (*informative* and *non-informative*, like previous work [97]) and five classes (*logic*, *presentation*, *performance*, *balance*, and *non-informative*). We also want to understand to what extent the best models for the two categorization problems would allow to achieve useful results in practice. RQ_2 evaluates the "segment categorization" step described in Section 3.2.3.

> RQ_3: *What is the effectiveness of GELID in grouping gameplay video segments by context?*

In the third RQ, we aim to understand what the best clustering algorithm is for grouping segments based on the game context, and how effective such an algorithm is in absolute terms. This RQ evaluates the clustering step described in Section 3.2.4.

> RQ_4: *What is the effectiveness of GELID in clustering gameplay video segments based on the specific issue?*

Similarly to RQ_3, RQ_4 aims at understanding which features and clustering algorithm allow to achieve the best results for clustering segments based on the specific issue, and how effective such an algorithm is in absolute terms. This RQ evaluates the clustering step described in Section 3.2.5.

3.3.1 Context Selection

To the best of our knowledge, there are no large-scale, publicly available databases of gameplay videos that provide meaningful information on the classification of problems in video games through subtitle analysis. To answer our RQs and validate the defined approach, we rely on gameplay videos from YouTube. While other platforms, even more video game-oriented, could be used (*e.g.,* Twitch), YouTube provides APIs for searching videos of interest and it also allows to download videos including subtitles, which are required by GELID. While subtitles can be automatically generated when the video lacks them, the results could be noisy and, in this phase, we evaluate GELID assuming high-quality input data. In our study, we collect three datasets, and the criteria used to search for gameplay videos of interest depend on the dataset at hand (explicited in the subsections below).

The first dataset is composed by video segments, and we use it used for training the supervised model used in step 2 of GELID (*i.e.,* segment categorization). We also use this dataset to select the best model for answering RQ_2. The second one is composed of complete videos, and we use it for evaluating the single components of GELID and answer RQ_{1-4}. The third one is a smaller dataset used to evaluate the parameters to be used in the different feature extraction and machine-learning techniques. We publicly release all datasets in our replication package [71].

Training Data

Our goal is to build a training set of labeled segments containing at least 1,000 instances and covering all the issue types GELID is able to identify.

To select videos possibly useful to build our training set, we used the YouTube Search APIs[2]. Specifically, we ran a query using the same keywords used by

[2] `https://developers.google.com/youtube/v3`

Table 3.2: Number of videos retrieved for each keyword.

Keyword	#Videos Retrieved	#Filtered Videos	#Segments
bug	594	514	691
glitch	509	487	282
hack	514	155	64
hacker	502	115	66
cheat	528	145	112
cheater	549	118	40

Lin et al. [97], *i.e.,* "bug", "hack", "glitch", "hacker", "cheat", and "cheater". For each keyword, we retrieved a list of videos matching it. We also added a filter to exclude videos without subtitles or with subtitles in languages different from English since GELID relies on NLP-based features computed on them. Some YouTube videos have manually-defined subtitles, while others have automatically generated ones. We include both of them. Indeed, while it is possible that the second category contains errors, this risk also exists in manually generated ones. Also, the quality of the subtitles generated by YouTube is generally quite high for the English language. As a result, we obtained 3,540 videos. Since some videos were present in more than a list (*i.e.,* they matched different keywords), we removed duplicates and obtained 3,196 videos. We report in Table 3.2, for each keyword, the number of videos retrieved and filtered, along with the number of extracted segments. Note that the number of segments might be lower than the number of filtered videos because a video might not contain valid keywords in the subtitles even though it contains them in other metadata, such as the title.

Our premise is that several gameplay videos report issues. However, issue-reporting videos represent a minority of the entire gameplay videos population (thus the relevance of our research). Therefore, to support the construction of the dataset containing training data for the categorization step, we relied on the approach defined by Lin et al. [97] and consider only videos identified as

issue-reporting. Specifically, we re-implemented their approach (since it is not publicly available) and, for each video retrieved as previously described, we ran the approach and discarded the videos classified as non-issue-reporting. As a result, we kept 1,534 videos. We shuffled such videos and manually analyzed them one by one to extract and label segments. One evaluator manually split each video into meaningful segments, and two evaluators manually labeled each segment as **logic**, **presentation**, **balance**, **performance**, or **non-informative** (when the segment does not report any issue). Specifically, in order to manually split the video into segments, one evaluator carefully watched each gameplay video, covering its entire duration. During this process, the author noted down the specific starting and ending times (in seconds) for each segment that they identified within the video. The identification of significant segments was guided by a specific criterion based on the classification outlined in Table 3.1, which can be found in Section 3.2.1. With the phrase "meaningful segments" we mean video segments that can be analyzed independently as shorter videos and contain enough information that can help achieve the objectives of GELID. To determine whether a segment is "meaningful", as we report later, we use the principles of *interpretability* (to what extent humans can get information from the segment) and *atomicity* (to what extent the segment contains only the information related to a single issue). At this stage, we discarded segments reporting more than an issue at a time. Given the large quantity of videos available compared to the target number of segments we had in mind, we decided to make sure that the training set was diverse in terms of video games considered. Thus, if we noticed that a video game was already taken into account in several videos previously analyzed, we avoided to analyze more videos of it. In total, we manually analyzed 170 gameplay videos, totaling about 17 hours of gameplay. As a result, we identified and labeled 1,255 video segments. Specifically, we obtained 693 non-informative video segments ($\sim$55.2%), 305 video segments reporting presentation-related problems ($\sim$24.3%), 169 video segments reporting logic problems ($\sim$13.5%), 47 video segments with balance

problems (∼3.7%), and 41 video segments highlighting performance problems (∼3.3%). Given the nature of the problem at hand, as we expected, the dataset is imbalanced, with a great majority of segments being non-informative and a very small percentage of them reporting balance- and performance-related issues.

Components Validation Data (Test Set)

To select videos on which we validate the single components of GELID, we focused on a small set of video games. We did this because the third and fourth steps of GELID are reasonable only when segments from the same video game are considered. To select the video games to use, we rely on the information available on Steam, one of the largest video game marketplaces [155]. Based on information obtained from Steam we select three video games that are both popular (*e.g.,* for which many gameplay videos exist) and that had several reported issues (*e.g.,* for which GELID gives the best advantage). More specifically, we select video games with many downloads and low review scores. To do this, we first retrieved the list of the top 100 most downloaded games on Steam, as reported in Table 3.3. Then, we excluded the games with *very positive* or better reviews (*i.e.,* we kept the ones with "mostly positive" reviews or lower). We preliminarily analyzed a random sample of 10 gameplay videos for each video game after this filter using the YouTube search feature. If we found no gameplay videos reporting issues, we discarded the video game. Then, for all the remaining video games, we used the YouTube Search APIs to search for "*video-game-name* gameplay video". We applied filters to select only videos with English subtitles (either manually added or automatically generated) and with medium (4-20min) and long (+20min) duration, with the aim of excluding non-informative videos representing game trailers or identifying a compilation of issues (which, instead, were useful to build the training set). Finally, we selected the three video games with the highest number of gameplay videos retrieved,

i.e., Conan Exiles [146], *DayZ* [147] and *New World* [148]. In total, we obtained 80 gameplay videos, totaling about 45 hours of gameplay.

Since manually splitting the entire videos would have been very demanding, we decided to partially rely on the first step of GELID. More specifically, we identified in the subtitles the keywords selected for the segmentation step. Then, one evaluator manually segmented the video near those points to select a first set of possibly relevant segments, and two evaluators independently manually categorized and clustered them both based on the context and on the specific issue (only for informative videos). The two annotators discussed conflicts to reach consensus. In total, we identified 604 video segments, distributed as depicted in Table 3.4. It is worth noting that we were able to identify only a few balance-related segments (4 in total, with DayZ having none of them).

3.3.2 Experimental Procedure

We summarize in Fig. 3.2 our plan for answering the four research questions, and we provide the details below.

Research Method for RQ_1: Meaningfulness of Extracted Segments

To answer RQ_1, we evaluate the technique we defined with different values of t (streamer reaction times). Specifically, we instantiate our approach with t in the set $\{0, 5, 10\}$ seconds.

We ran the first step of *Video Segmentation* on selected gameplay videos for each video game in the test set, collecting a total of 101 video segments. Note that the number of extracted segments is lower than the number of videos because some videos might not contain any keyword we use in the *Video Segmentation* step to retrieve candidate relevant segments (see Section 3.2.2). Consequently, if a video does not contain any of these keywords, no segment is extracted from it.

Table 3.3: Top 100 most popular games on Steam and related summary review scores ("Overwhelmingly Positive" ✔✔✔, "Very Positive" ✔✔, "Mostly Positive" ✔, "Mixed" ∼, and "Mostly Negative" ✖)

Video game	Review	Video game	Review
CS:GO	✔✔	BeamNG drive	✔✔✔
Pubg	∼	Counter strike	✔✔✔
Dota 2	✔✔	RimWorld	✔✔✔
GTA V	✔✔	World of Tanks Blitz	✔✔
Tom Clancy's Rainbow Six® Siege	✔✔	The Elder Scrolls V: Skyrim Special Edition	✔✔
Team fortress 2	✔✔	NARAKA Bladepoint	✔
Terraria	✔✔✔	Hunt: Showdown	✔✔
Garry's Mod	✔✔✔	Civilization V	✔✔✔
Rust	✔✔	Project Zomboid	✔✔
Apex	✔✔	Factorio	✔✔✔
Wallpaper Engine	✔✔✔	Smite	✔
The Witcher® 3: Wild Hunt	✔✔✔	The elder scrolls online	✔✔
Warframe	✔✔	theHunter: Call of the Wild™	✔✔
Destiny 2	✔✔	Age of Empires II: Definitive Edition	✔✔
Cyberpunk 2077	✔	Satisfactory	✔✔✔
Dead by Daylight	✔✔	Stellaris	✔✔
ARK	✔✔	Fifa 22	✔✔
Elden ring	✔✔	Forza Horizon 5	✔✔
Stardew Valley	✔✔✔	Squad	✔✔
Euro track simulator 2	✔✔✔	The sims 4	✔✔
Rocket League	✔✔	Europa Universalis IV	✔✔
Phasmophobia	✔✔✔	Scum	✔
Payday 2	✔✔	Stumble Guys	✔✔
The forest	✔✔✔	Assetto Corsa	✔✔
War Thunder	✔	Conan Exiles	✔
Valheim	✔✔✔	FINAL FANTASY XIV ONLINE	✔✔
Brawlhalla	✔✔	Crusader Kings III	✔✔
Red dead redemption 2	✔✔	Yugioh Master Duel	✔
DayZ	✔	Left for dead	✔✔✔
Don't Starve together	✔✔✔	eFootball 2023	✖
Sea of thieves	✔✔	Black desert	✔
New World	∼	Soundpad	✔✔✔
Geometry Dash	✔✔	Total War: Warhammer 3	✔
Bloons TD 6	✔✔✔	Fallout 76	✔
The binding of Isaac: Rebirth	✔✔✔	Warhammer 40,000: Darktide	∼
Path of exile	✔✔	Moster Hunter Rise	✔✔
Hades	✔✔✔	Coockie clicker	✔✔✔
Fallout 4	✔✔	EA SPORTS™ FIFA 23	∼
VR Chat	✔	Farming Simulator 22	✔✔
Lost Ark	✔	Victoria 3	∼
Civilization VI	✔✔	Goose Goose Duck	✔✔
7 days to die	✔✔	Undecember	∼
Mount Blade II: Bannerlord	✔✔	Mir4	∼
Vampire Survivors	✔✔✔	Footbal Manager 2022	✔✔
Cities: Skylines	✔✔	Dwarf fortress	✔✔✔
TmodLoader	✔✔✔	Nba 2K23	∼
Arma 3	✔✔	Project: Playtime	∼
Deep rock Galactic	✔✔✔	Divinity: Original Sin 2 - Definitive Edition	✔✔✔
Hearth of Iron IV	✔✔	Paragon the Overprime	∼
Call of Duty®: Modern Warfare® II	∼	Football Manager 2023	✔✔

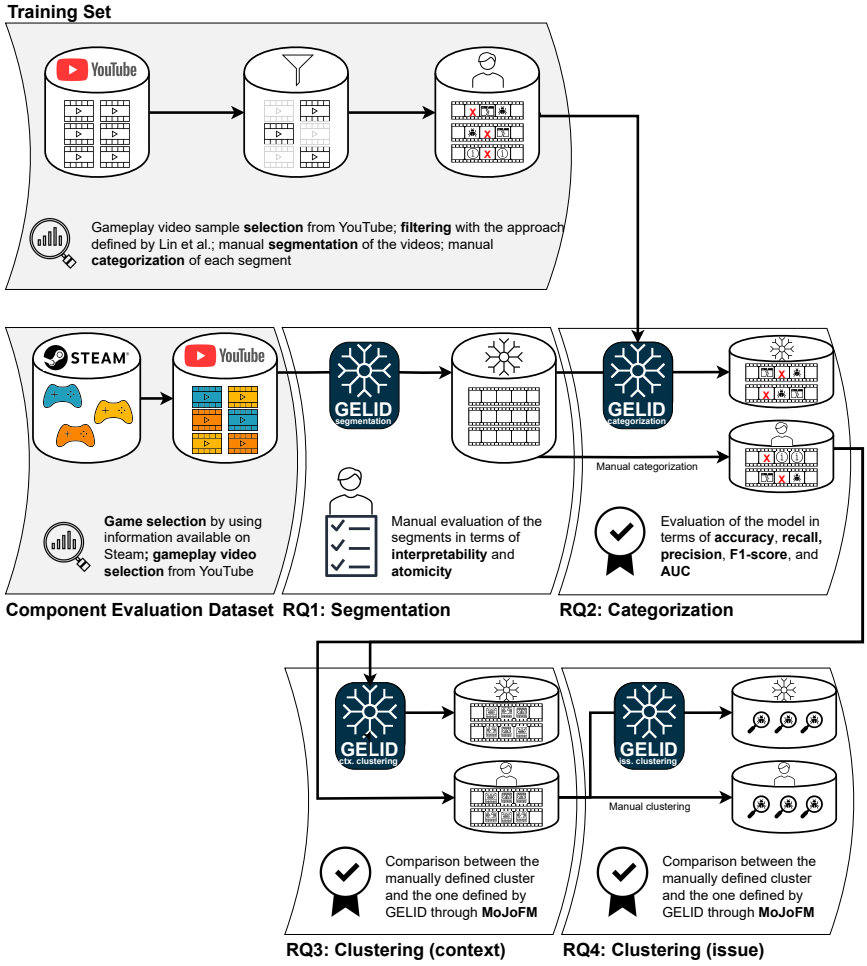


Figure 3.2: Summary of the study design.

Table 3.4: Distribution of issue types (logic, presentation, performance, balance, and non-informative) for each video game considered in the test set.

Video Game						**Total**
Conan Exiles	37	109	10	1	157	314
DayZ	7	67	16	0	90	180
New World	2	44	6	3	55	110
Total	46	220	32	4	302	604

We evaluated the segments detected by each variant of our approach in terms of their (i) *interpretability* (*i.e.,* it is possible to watch the segment and acquire all the information needed to understand what has been experienced by the streamer) (ii) the *atomicity* (*i.e.,* it is not possible to further split the segments). Such aspects are complementary: It would be possible to maximize the *interpretability* by creating few segments (*e.g.,* just one for the whole video); this, however, would result in lower *atomicity* since the segments could be further divided into parts. While we would have ideally wanted to capture the "quality" of segments as a whole, it is quite hard to define a precise metric for such a complex aspect. Thus, we preferred to use two specific and easy-to-evaluate aspects instead. Concerning the relationship between such aspects and quality as a whole, we can say that, given two segments A and B, if interpretability(A) $>$ interpretability(B) and atomicity(A) $>$ atomicity(B), then quality(A) $>$ quality(B). On the other hand, if we have conflicting situations (*e.g.,* interpretability(A) $<$ interpretability(B) and atomicity(A) $>$ atomicity(B)), we can not say whether the quality of A is greater or lower than the quality of B.

Two evaluators watched the segments generated by each variant, for a total of 303 evaluations, and manually annotated each segment in terms of its *interpretability* and *atomicity* on a 5-point Likert scale. As for the first metric, we evaluated to what extent we could fully understand what is happening based only on the segment itself. As for atomicity, instead, we assessed whether the

segment can be further divided in additional standalone (fully interpretable) segments. The final score was computed as 5 minus the number of additional standalone segments that could be further extracted, or 1 if more than four standalone segments were found. Each of the 303 manually analysed slices was independently inspected. We report the inter-rater reliability between the annotators by using the Cohen's kappa coefficient [45, 161]. Then, for each segment, we compute the mean *interpretability* and *atomicity*. Finally, we compare the tested techniques in terms of such metrics using a Mann-Whitney U test [102, 100], and adjusting the p-values resulting for multiple comparisons using the Benjamini and Hochberg procedure [21]. We also report the effect size, using the Cliff's delta [44], to understand the magnitude of differences observed.

Research Method for RQ_2: Segment Categorization Effectiveness

To answer RQ_2, we use all the three datasets previously described. We aimed at evaluating not only the complete approach on a multi-class categorization problem (the four informative classes reported in Table 3.1, plus the *non-informative* class), but also its version on a simplified version of the same problem, *i.e.,* a binary classifier (*informative*, *non-informative*) like the one defined by Lin et al. [97]. It is worth noting, however, that we could not compare our results with the ones obtained with such an approach because it is designed to work only on entire videos, not on segments.

As a first step, we aimed at selecting (i) the best machine learning algorithm, (ii) the best set of features, and (iii) the best preprocessing pipeline for categorizing gameplay video segments in both scenarios. As candidate machine learning algorithms, we selected Random Forest [77], Logistic Regression, SMO [76], Multilayer Perceptron [130] and IBk[39]. We used the implementations available in the Weka toolkit.[3] At this stage, we used the default hyperparameters avail-

[3] `http://www.cs.waikato.ac.nz/ml/weka/`

able in Weka for each of them. As candidate set of features, as explained in Section 3.2, we considered three textual-based sets of features (Bag of Words, word2vec, and doc2vec), a video-based set of feature, and a mixed set of features (including both the best set of textual features and the video-based set of features). As candidate preprocessing pipelines, we considered the use of SMOTE [32], which allows to generate synthetic instances for balancing the training set, and a two-step attribute selection approach: We first rank the features based on their respective information gain and we discard the ones with score 0; then, we run a wrapper attribute evaluator [66] to select the best subset of features in terms of AUC achieved by a simple kNN model with $k = 3$. More specifically, we considered four options: the use of SMOTE alone, the use of our two-step attribute selection alone, the use of both of them, and the use of none of them. At this stage, we relied on the training set, and we performed a 10-fold cross validation for all the combinations of ML algorithms, feature sets, and preprocessing pipelines for both the problems (binary and multi-class). For each of them, we compute and report the achieved AUC (Area Under the ROC curve [24]) [59]. An AUC of 0.5 indicates a model having the same prediction accuracy of a random classifier. A perfect model (*i.e.,* zero false positives and zero false negatives) has instead AUC = 1.0. Thus, the closer the AUC to 1.0, the higher the model performances. In the end, we select the combination that allows achieving the highest score both for the binary and the multi-class model.

Finally, as a third step, we ran the best models on the test set to understand to what extent the models would be useful in practice. In this case we report not the AUC, but also the *precision*, *recall*, and *F-measure* scores. *Precision* is computed as $\frac{TP}{TP+FP}$ and *recall* is computed as $\frac{TP}{TP+FN}$, where *TP*, *FP*, and *FN* indicate the number of *true positives*, *false positives*, and *false negatives*, respectively. *F-measure* is computed as the harmonic mean of *precision* and *recall*.

Research Method for RQ_3: Contextual Clustering Effectiveness

To address RQ_3, we tested the two non-parametric clustering techniques described in Section 3.2, *i.e.,* DBSCAN [55], OPTICS [14] with two distance metric, *i.e.,* HSV and SSIM.

Both DBSCAN and OPTICS require to set an ϵ parameter, which indicates the minimum distance to be used to consider two instances belonging to the same cluster. However, determining the input parameter values can be very difficult. For both non-parametric clustering techniques, we decide the value of ϵ by using a well-known procedure [111]. Specifically, we (i) calculate the distance between each point and its nearest neighbour, (ii) sort the distances in ascending order, (iii) compute, for each pair of consecutive distances, their difference Δ_i = d_{i+1} - d_i), and (iv) set $\epsilon = \max(\Delta_i)$. We used this procedure independently for each clustering operation we run (*i.e.,* each combination of video game and similarity metric).

We compare the results of the algorithms with the ground-truth partition produced in the manual clustering of the test set to evaluate this step of GELID. To do this, we use the MoJo eFfectiveness Measure (*MoJoFM*) [164], a normalized variant of the MoJo distance. *MoJoFM* is computed using the following formula:

$$MoJoFM(A,B) = 100 - (\frac{mno(A,B)}{max(mno(\forall E_A, B))} \times 100)$$

where $mno(A,B)$ is the minimum number of *Move* or *Join* operations one needs to perform in order to transform a partition A into a different partition B, and $max(mno(\forall\ E_A, B)$ is the maximum possible distance of any partition A from any partition B. *MoJoFM* returns 0 if partition A is the farthest partition away from B; it returns 100 if A is equal to B.

We report the MoJoFM obtained for each combination of game and metric considered.

Research Method for RQ_4: Specific Issue Clustering Effectiveness

To answer RQ_4, we tested the same clustering techniques considered in RQ_3 (DBSCAN [55] and OPTICS [14]) plus a third (*i.e.,* Mean Shift [61]) which we could not use in RQ_3 because it can not use custom distance metrics. We start from the ground-truth clusters manually defined in the test set. For each of them, we run the issue-based clustering approach defined in Section 3.2 on the instances belonging to them. We use the same procedure described in RQ_3 to define the ϵ hyperparameters for DBSCAN and OPTICS for each clustering operation. This time, we do not report the ϵ values used for space reasons (given the higher number of clustering operations). We report, like for RQ_3, the MoJoFM score achieved for each video game. We publicly release in our replication package [71] the datasets used in each research question, the ARFF files used to train and test the machine learning techniques, the raw data of our manual analyses for each research question, and additional data that did not fit in our chapter. We also publicly provide the implementation of each step of GELID.

3.4 Empirical Study Results

This section reports the results of the four research questions formulated in Section 5.4.

3.4.1 RQ_1: Interpretability and Atomicity of Gameplay Video Segments

The IRR between the two raters when they evaluated the *interpretability* of the segments extracted with GELID is $k = 0.84$, while it is $k = 0.85$ when evaluating them in terms of *atomicity*. Thus, in both the cases, the agreement was *almost perfect*.

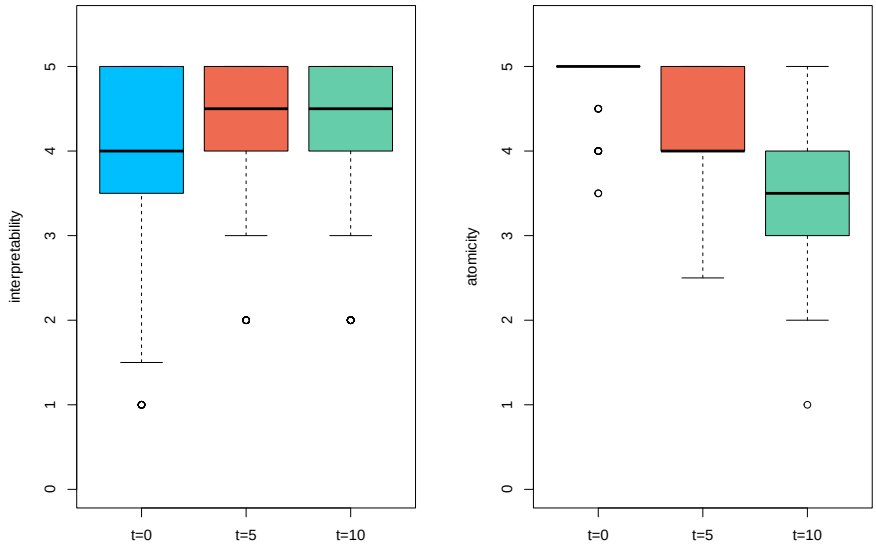


Figure 3.3: Distribution of interpretability (left) and atomicity (right) evaluation of gameplay video segments with the three different thresholds ($t = 0$, $t = 5$, $t = 10$)

When comparing $t = 0$ with $t = 5$ in terms of *interpretability* of gameplay video segments generated by GELID, we obtain an adjusted *p-value* < 0.001, with a *negligible* effect size (δ = -0.146). We obtain an analogous result when comparing $t = 5$ with $t = 10$ (p < 0.001, δ = -0.092, *negligible* magnitude). We observed also a difference between $t = 0$ with $t = 10$: In this case, the adjusted *p-value* is the same (p < 0.001), while, this time, the effect size is *small* (δ = -0.227). The boxplot in Fig. 3.3 (left part) visually confirms the difference we numerically observed.

In terms of *atomicity*, when comparing $t = 0$ with $t = 5$ we obtain an adjusted *p-value* < 0.001, with a *large* effect size (δ = 0.610). We obtain an analogous result when comparing $t = 5$ with $t = 10$ (p < 0.001, δ = 0.577, *large* magnitude). As expected, again, the difference between $t = 0$ with $t = 10$ is *large* as well (*p-value* < 0.001, δ = 0.869). The boxplot in Fig. 3.3 (right part) visually confirms the difference we numerically observed. A case in which two annotators disagreed on the evaluation of the atomicity of a segment is related to segment in a gameplay video on Conan Exiles. One author rated the atomicity of the segment as 4, while the second author rated it as 5. The disagreement arose from the presence of a gameplay setting screen that appeared during the video segment, lasting about 3 seconds. This setting screen interrupted the ongoing game phase and then resumed it. The first evaluator considered this interruption significant enough to be considered as a point in which two segments could be detected, while the second annotator considered the screen appearance time negligible, given its short duration.

Considering overall the results, we can conclude that by increasing the t value we obtain negligible advantages in terms of interpretability and substantial disadvantages in terms of atomicity. More specifically, while increasing t from 0 to 5 allows to obtain an observable difference in terms of interpretability, having a t value higher than 5 practically brings no advantage at all (see Fig. 3.3 — left part). We conclude that t values higher than 5 are most likely not worth considering, while there is a trade-off that users might want to consider between

$t = 0$ (which allows having substantially more atomic videos) and $t = 5$ which allows having more interpretable videos, even if slightly). Indeed, we obtain for $t = 0$ an average interpretability of 3.97 and an average atomicity of 4.88, while $t = 5$ provides an average value of 4.27 for both interpretability and atomicity.

Answer to RQ_1. The proposed segmentation approach achieves satisfactory results. The best results are obtained when using $t = 0$ (privileging atomicity) and $t = 5$ (privileging interpretability).

3.4.2 RQ_2: Gameplay Video Segments Categorization

ML Pipeline Selection and Training. We report in Table 3.5 and Table 3.6 the results of the 10-fold cross-validation comparison performed on the training set to select the best algorithm both for binary and multi-class categorization, respectively. For deciding which sets of textual features we would include in the combination of image-based features and textual features, we compared the average results obtained with textual features alone and we picked the features that generally allow to achieve the best results (*i.e.,* Word2Vec).

The machine learning algorithm that provides the best results for binary classification is Random Forest, while the best set of features is the combination of image-based and textual features. Using both SMOTE and attribute selection, we obtained 0.79 AUC (71.8% accuracy). The best results could be achieved with Random Forest and a combination of image-based and textual features for multi-class categorization as well. This time, however, the best model was the one trained by only running attribute selection (*i.e.,* without balancing the training set with SMOTE). In this case, the obtained AUC is slightly lower (0.75 AUC, 62.0% accuracy), most likely due to the inherently more difficult problem (categorizing in five classes instead of two).

Table 3.5: RQ_2: Comparison, in terms of unweighted average AUC, of different sets of features (**B**ag **o**f **W**ords, **W**ord**2V**ec, **D**oc**2V**ec and **I**mage-based features), preprocessing techniques (SMOTE and **A**ttribute **S**election), and ML algorithms for binary classification (*non-informative* or *informative*).

	Model	Plain	AS	SMOTE	SMOTE + AS
BoW	RandomForest	0.72	0.73	0.72	0.73
	Logistic	0.63	0.74	0.62	0.73
	SMO	0.68	0.60	0.68	0.66
	MultilayerPerceptron	0.52	0.73	0.68	0.73
	IBk	0.58	0.73	0.60	0.73
W2V	RandomForest	0.72	0.72	0.74	0.71
	Logistic	0.68	0.70	0.69	0.70
	SMO	0.65	0.65	0.65	0.65
	MultilayerPerceptron	0.73	0.68	0.72	0.69
	IBk	0.62	0.62	0.62	0.63
D2V	RandomForest	0.52	0.50	0.52	0.50
	Logistic	0.50	0.50	0.49	0.50
	SMO	0.51	0.50	0.48	0.50
	MultilayerPerceptron	0.52	0.50	0.56	0.50
	IBk	0.50	0.50	0.52	0.50
I	RandomForest	0.74	0.69	0.74	0.68
	Logistic	0.69	0.66	0.69	0.66
	SMO	0.61	0.58	0.58	0.57
	MultilayerPerceptron	0.67	0.66	0.69	0.66
	IBk	0.62	0.61	0.62	0.60
W2V + I	RandomForest	0.78	0.78	0.79	**0.79**
	Logistic	0.70	0.72	0.70	0.72
	SMO	0.68	0.65	0.67	0.63
	MultilayerPerceptron	0.74	0.73	0.76	0.71
	IBk	0.65	0.66	0.64	0.66

Table 3.6: RQ_2: Comparison, in terms of unweighted average AUC, of different sets of features (**B**ag **o**f **W**ords, **W**ord**2V**ec, **D**oc**2V**ec, **I**mage-based features), preprocessing techniques (SMOTE and **A**ttribute **S**election), and ML algorithms for multi-class classification (*logic*, *presentation*, *performance*, *balance*, *non-informative*).

	Model	Plain	AS	SMOTE	SMOTE + AS
BoW	RandomForest	0.72	0.70	0.71	0.69
	Logistic	0.70	0.69	0.70	0.70
	SMO	0.67	0.62	0.67	0.69
	MultilayerPerceptron	0.70	0.71	0.70	0.70
	IBk	0.60	0.69	0.60	0.69
W2V	RandomForest	0.73	0.72	0.70	0.71
	Logistic	0.59	0.71	0.63	0.69
	SMO	0.67	0.60	0.70	0.67
	MultilayerPerceptron	0.67	0.64	0.68	0.65
	IBk	0.58	0.59	0.61	0.60
D2V	RandomForest	0.52	0.49	0.51	0.49
	Logistic	0.48	0.49	0.48	0.49
	SMO	0.49	0.50	0.49	0.49
	MultilayerPerceptron	0.52	0.49	0.50	0.49
	IBk	0.49	0.49	0.51	0.49
I	RandomForest	0.69	0.62	0.67	0.62
	Logistic	0.66	0.64	0.65	0.64
	SMO	0.54	0.53	0.62	0.61
	MultilayerPerceptron	0.52	0.66	0.63	0.64
	IBk	0.56	0.58	0.56	0.56
W2V + I	RandomForest	0.74	**0.75**	0.74	0.71
	Logistic	0.61	0.71	0.65	0.70
	SMO	0.70	0.57	0.71	0.68
	MultilayerPerceptron	0.71	0.67	0.71	0.66
	IBk	0.60	0.59	0.62	0.58

Testing the Models. Table 3.7 and Table 3.8 report the recall, precision, F-Measure and AUC scores achieved by the best model for binary and multi-class categorization, respectively. In detail, we report the results achieved both for individual games and for all the instances together.

Overall, the binary classification model exhibits slightly worse results compared to the ones obtained on the training set with 10-fold cross validation (0.61 AUC vs. 0.79). The model has an acceptable recall (72%) and a relatively low precision (56%) on the *informative* class. This means that a developer would be able to get most of the potentially interesting segments, but they also have to manually discard many non-informative ones in the process. The results, however, depend much on the video game at hand: For Conan Exiles, for example, the model always achieves acceptable results both in terms of overall precision (66%) and recall (64%). This might depend on many factors. First, on the quality of the streaming videos taken into account: Streamers might be more verbose for some video game genres, thus allowing the classifier to better identify the segments. Second, on the similarity with video games included in the training set: Some genre- or game-specific terms might be indicative of an issue for some games, while not for others. For example, the phrase “loot hack” might appear in online multiplayer role play games and indicate a *logic* issue, but it might not be pronounced at all by streamers playing racing games.

Analogous conclusions can be drawn from the results achieved with the multi-class model. In this case, it is interesting to observe that some classes the classifier never categorizes instances as *performance* and *balance* (“–” for precision in Table 3.8). This is possibly due to the fact that such issue types are generally less prevalent than others[4] and, thus, the model fails to learn how to recognize them. It is also worth noting that we were not able to find *balance* issues in one of the games taken into account (*i.e.,* DayZ).

[4] 33 and 48 in the training set, 31 and 4 in the test set for *performance* and *balance*, respectively.

Table 3.7: RQ_2: Performance of the best binary categorization model on the test set. We use the icon ℹ to indicate the *informative* class and the icon 🗑 to indicate the *non-informative* class, while 🌐 indicates their weighted mean.

Game	Precision			Recall			F-Measure			AUC		
	ℹ	🗑	🌐	ℹ	🗑	🌐	ℹ	🗑	🌐	ℹ	🗑	🌐
Conan Exiles	61%	70%	66%	77%	52%	64%	68%	60%	64%	0.73	0.73	0.73
DayZ	52%	54%	53%	71%	34%	53%	60%	42%	51%	0.55	0.55	0.55
New World	65%	47%	58%	71%	40%	59%	68%	43%	59%	0.61	0.61	0.61
Overall	56%	62%	60%	72%	45%	58%	63%	52%	58%	0.58	0.64	0.61

Overall, the model achieves better results on the *presentation* class. This is probably due to the fact that, for this category, the model also relies on image-based features, which are less relevant for the other classes.

Answer to RQ_2. The categorization models defined are not able achieve satisfactory results both for binary and multi-class categorization.

3.4.3 RQ_3: Clustering Gameplay Video Segments by Context

Table 3.9 shows the MoJoFM score achieved by the two tested algorithms when comparing their output with the manually defined clusters. First, it can be observed that OPTICS allows to achieve the best results for all games taken into account, between 46.0% (New World) and 21.9% (Conan Exiles). It is worth noting that the variability among video games is, in this case, quite high. This is expected: Some games have areas and levels very similar one to another, thus making the task of visually distinguishing the areas quite challenging even for a human who never played the game. For example, the frames presented in Fig. 3.4 represent two visually similar areas in Conan Exiles that, however, are different. Overall, however, we can conclude that the clustering approach we defined in GELID is only partially able to achieve its goal.

Table 3.8: RQ_2: Performance of the best multi-class categorization model on the test set. We use the icons 🗑, 👻, 🐞, ⏱, and ⚖ to indicate the *logic*, *presentation*, *performance*, *balance*, and *non-informative* classes, respectively, while 🌐 indicates their weighted mean.

Game	Precision						Recall					
	🗑	👻	🐞	⚖	⏱	🌐	🗑	👻	🐞	⚖	⏱	🌐
Conan Exiles	58%	48%	25%	-	-	48%	81%	39%	3%	0%	0%	55%
DayZ	51%	35%	0%	-	-	39%	61%	34%	0%	-	0%	43%
New World	56%	49%	25%	-	-	48%	71%	40%	50%	0%	0%	52%
Overall	56%	44%	13%	-	-	45%	73%	38%	10%	0%	0%	51%
Game	**F-Measure**						**AUC**					
	🗑	👻	🐞	⚖	⏱	🌐	🗑	👻	🐞	⚖	⏱	🌐
Conan Exiles	68%	43%	5%	-	-	49%	0.69	0.65	0.58	0.33	0.57	65%
DayZ	55%	35%	0%	-	-	43%	0.52	0.52	0.53	-	0.47	51%
New World	63%	44%	33%	-	-	49%	0.64	0.60	0.93	0.72	0.10	62%
Overall	63%	40%	10%	-	-	47%	0.63	0.60	0.54	0.61	0.48	60%

Table 3.9: RQ_3: MoJoFM achived for clustering by context with HSV

	DBSCAN	OPTICS
Conan Exiles	17.8%	21.9%
DayZ	23.2%	36.6%
New World	28.0%	46.0%
Average	23.0%	34.8%

Answer to RQ_3. We obtained mixed results for the clustering by context step because its performance strongly depends on the video game at hand.

3.4.4 RQ_4: Clustering Gameplay Video Segments by Specific Issues

Table 3.11 shows the MoJoFM score achieved by the three tested algorithms when comparing their output with the manually defined clusters. In this case, the results are definitely better than the ones obtained in the previous experiment,

Figure 3.4: Different game scenes in Conan Exiles grouped in the same context cluster

Table 3.10: RQ_3: MoJoFM achived for clustering by context with SSIM

	DBSCAN	OPTICS
Conan Exiles	4.8%	21.9%
DayZ	0.0%	2.5%
New World	0.0%	18.5%
Average	1.6%	14.3%

with the best-performing algorithm (DBSCAN) achieving 72.7% MoJoFM score. This is due to the fact that, in this case, there were less instances to cluster for two of the games taken into account (DayZ and New World). As a result, the task was inherently easier. It is worth noting, however, that for Conan Exiles the number of instances to cluster was quite large, in some cases, up to 18 and DBSCAN still achieves very good results (71.2% MoJoFM).

Differently from what observed for RQ_3, we have a much less marked variance among the games (between 69.1% and 77.8%). OPTICS, in this case, achieved slightly worse results than DBSCAN, while MeanShift is clearly less effective than the others.

Answer to RQ_4. DBSCAN allows to cluster the segments very similarly to how human annotators clustered them, with a low variability among video games.

Table 3.11: RQ_4: MoJoFM achived for clustering on the specific-issue.

	DBSCAN	OPTICS	MeanShift
Conan Exiles	71.2%	62.5%	52.9%
DayZ	69.1%	69.1%	58.2%
New World	77.8%	77.8%	55.6%
Average	72.7%	69.8%	55.5%

3.5 Discussion

The main problems we encountered are in the automated categorization of issues in gameplay video segments and in the context-based segment clustering (steps 2 and 3 of GELID).

First, it is worth noting that our results partially contrast the ones obtained by Lin et al. [97], who defined a classifier able to correctly distinguish *informative* from *non-informative* gameplay videos. Segment-level categorization is a much harder problem than video-level categorization. This is confirmed by the fact that even simplifying our five-class categorization problem in binary categorization problem (similarly to the one addressed by Lin et al. [97], but on segments), we still obtain negative results (58% F-Measure, with 0.61 AUC). We have some hypothesis on why this is the case. First, videos have metadata (such as tags, descriptions, and so on) that segments lack. Lin et al. [97] used such metadata, but we could not use them in our context. If a video is specifically aimed at reporting issues (*i.e.,* it contains a compilation of game errors), it is very likely that the evaluators explicitly mention this in the description. Gameplay video subtitles, instead, are much more noisy.

We observed that, often, the subtitle sentences are incomplete and ambiguous (*e.g.,* “logics, bro. Well, I talk all” used for a *logic* problem, “they are lower

than that" used for a *presentation* problem, and "less well-known logic that's arguably one" used for a *performance* problem). To some extent, this happens because the comment corresponding to the portion of the video in which the issue appears might not be in sync with the issue itself: Gamers might talk about the issues even several minutes after it appears. It is worth noting that this problem is not related to the automated segmentation, because in evaluating step 2 with RQ_2 we used manually-defined segments. The problem is in the lack of (logical) sync between what streamers say and when what they say happens on screen. Future work could consider a larger context for extracting the features (*e.g.,* the surrounding n seconds, with even large values of n) instead of only considering the subtitles related to the specific segment. The idea based on the possibility of using a larger context stems from the assumption that expanding the context of observation allows for a broader view of what is happening in the specific gameplay video, thus in the game, and allows more features to be extracted.

Lesson Learned 1. Considering a larger context for extracting textual features might allow obtaining better results.
Future Research Idea 1. To overcome this limitation, future research could aim to consider a larger portion of video both before and after the given identified segment.

Using keywords to detect possibly useful segments of the gemeplay videos might be detrimental. Indeed, there may be segments without streamer comments, that would be completely ignored. These are blind spots for GELID. To address this limitation, it may be necessary to develop new and specialized approaches to detect specific problems, such as glitches or stuttering events.

Related to this, another problem we noticed by analyzing some examples is that streamers sometimes comment on their gaming experience in an irregular manner, often even through simple exclamations (*e.g.,* "the glitch myself?" for *performance*, "BAM!" for *logic*, "and there!" for *presentation*). Catching those

issues is probably infeasible by only relying on textual information. Similarly, we can observe a performance problem found in a gameplay video of New World:[5] The game temporarily freezes while the player is running, but they say *"here can see one right now okay stop doing that let's start running they're nasty big aren't they"*, referring to what is happening in the game. Automatically categorizing this kind of issue is, again, extremely challenging, and a more specific approach would be needed. Another limitation of GELID is related to the fact that it only relies on gameplay videos in English. Future work is needed to generalize it to other languages. In CLAP [138], an attempt has been made to deal with this issue. The evaluators tried to translate the input textual information (in the context of GELID, subtitles) from foreign languages into English and then use the normal approach (which works on English) to deal with them. However, this solution proved to be unsuccessful. In this chapter, we use word2vec: It would be possible to test the effectiveness of word2vec models trained on other languages. Based on the negative results obtained for English, which is quite widespread, we believe that the implementation of such an approach cannot be successful at present.

Lesson Learned 2. Sometimes, textual features are not useful at all since the streamers use generic exclamations to report issues.
Future Research Idea 2. Future research could aim at taking into account the slang used by streamers and to define a vocabulary of the terms most commonly used to describe different kinds of issues or to define specialized approaches to detect issues mostly based on the videos rather than on the captioned spoken content.

When looking at the multi-class categorization, the problem is even more evident in terms of general effectiveness of the model. We report in Table 3.12 the confusion matrix for the multi-class categorization model. While the model

[5] `https://youtu.be/1duizy5DS0g?t=1540`

correctly identifies 81 *presentation* issues, it correctly detects only 2 *logic*-related issues and, again, no *performance*- and *balance*-related issue. More interestingly, the model often categorizes *presentation*-related issues as *logic* issues, while the opposite happens relatively less frequently. In general, instead, the model tends to confuse the specific categories of instances as *presentation*-related, probably because it is the most frequent informative type of issue.

We analyzed some misclassified instances, aiming at getting some insights on why the model tends to confuse some *presentation* issues for *logic* issues and why it is not able to correctly identify *performance* and *balance* instances. We found an interesting example in DayZ. The streamer says *"my doesn't seem to be archived it back back is so annoying"*,[6] but the model probably confuses the indication of an "annoying" circumstance for something related to a functional issue (*logic*), while, in this case, it was referred to a *presentation* issue.

Lesson Learned 3. Given the strong class unbalance, categorization does not work well for detecting *performance* and *balance* problems. Approaches specifically designed for finding such categories of issues might be needed.
Future Research Idea 3. To increase the number of *balance* and *performance* instances, it could be useful to look for and specifically take into account video games that are or have been notorious for such problems.

Another possible reason behind the failure in categorization could be related to the procedure used to define the training set: To collect an adequate number of instances, we considered videos that explicitly report issues (*i.e.,* that contain keywords such as "bug" in their title or description). It is possible that these videos are intrinsically different from the long gameplay videos we used for testing the models. To check if this is the case, we trained/tested two classifiers (both for binary and multi-class categorization) based on the best configurations found in RQ_2 by using 10-fold cross validation on the test set alone, both globally

[6] `https://youtu.be/eDQIdqDC-sc?t=239`

Table 3.12: RQ_2: Confusion Matrix for multi-class categorization on all the instances (Conan Exiles, DayZ, and New World). The columns indicate the categories assigned by the classifier, while the rows indicate actual ones.

	218	75	5	0	0
	126	**81**	8	0	0
	27	16	**2**	0	0
	18	13	0	**0**	0
	3	1	0	0	**0**

and by considering the instances of single games. We report the results in Table 3.13. We observed a clear increase in the effectiveness of both the models, with the binary classification model achieving $\sim$82% accuracy on two games. While more data would be necessary, the results of this analysis suggest that videos explicitly reporting issues are too different from long gameplay videos (that we aim to target) in which issues sometimes appear. Thus, it would be more appropriate to build the training set using the same procedure used to build the test set, even if this require a much bigger effort (it would not be possible, for example, to use the approach by Lin et al. [97] as a filter). Also, using a training set composed of only game-specific instances might allow to achieve better results (even if we observed this only for two games out of three). In detail, again, a training set defined on a specific game allows for more precise information in relation to the game area/level. For example, open world games have very similar game areas, so a large amount of data would allow a more precise distinction to be made between the different game areas in which users find themselves.

Table 3.13: Accuracy and AUC achieved by training/testing the best models for binary and multi-class categorization on the test set alone using 10-fold cross validation.

Game	Binary		Multi-class	
	Accuracy	AUC	Accuracy	AUC
Conan Exiles	81.7%	0.89	71.7%	0.73
DayZ	64.7%	0.75	59.0%	0.56
New World	81.7%	0.89	67.9%	0.72
Combined	72.7%	0.79	59.9%	0.63

Lesson Learned 4. A training set built on long gameplay videos not specifically aimed at reporting issues might help achieving better results. Also, game-specific training might help increasing the model accuracy.
Future Research Idea 4. Future research should verify what is the impact of the type of video, *i.e.,* long and generic gameplay videos or short and focused gameplay videos reporting issues, on the performance of the four steps of GELID.

As for the context-based segment categorization (step 3 of GELID), as we previously mentioned while analyzing the results, the poor performance can be due to the fact that some games have visually similar, but logically different game areas/levels. Some video games might suffer from this issue more than others. In our case, we observed that our approach (specifically, the variant based on HSV histogram correlation, which achieves the best results) works reasonably well on New World, but remarkably bad on Conan Exiles. For the video games on which our approach does not work well, a more sophisticated (and game-specific) approach might be used, which should be specialized on the game at hand so that, for example, it is able to distinguish the specific game areas by recognizing specific game elements.

Lesson Learned 5. A game-specific approach for recognizing the game area/level might be needed for some video games.
Future Research Idea 5. Researchers should test the impact of introducing game- or game-genre-specific features on the effectiveness of the context-based clustering.

3.6 Threats to Validity

Threats to construct validity mainly pertain the possible imprecisions made while defining the test set used to evaluate GELID and to answer all our research questions. As explained in Section 5.4, to reduce this threat, two evaluators independently tagged each instance and discussed conflicts aiming at reaching consensus. This occurred in 1.2% of the cases.

Threats to internal validity concern factors internal to our study that could have affected the results. A first threat regarding RQ_2 is related to the specific set of ML techniques we decided to use and to the preprocessing pipelines we tested. As for the first, we took into account the main categories of classic ML approaches. It is possible that Deep Learning-based approach achieve better results, but we avoided using such approaches because even a small Neural Network (Multilayer Perceptron) achieves very poor results given the small size of our training set. Another limitation related to RQ_2 is the choice not to tune the hyperparameters and to use the predefined hyperparameters provided by Weka. To understand the impact of this decision, we tried to replicate the results of binary classification of segments as *informative* and *non-informative* while varying the main hyperparameter for Random Forest (*i.e.,* the maximum number of features). We report in Table 3.14 the results of such an analysis on the Conan Exiles dataset (note that the results differ from the ones reported in Table 3.7 because we did not run any preprocessing step here).

Table 3.14: RQ_2: Hyperparameter Tuning of the Random Forest categorization model on Conan Exiles. We use the icon ℹ to indicate the *informative* class and the icon 🗑 to indicate the *non-informative* class, while 🌐 indicates their weighted mean.

NumFeatures	Precision			Recall			F-Measure			AUC		
	ℹ	🗑	🌐	ℹ	🗑	🌐	ℹ	🗑	🌐	ℹ	🗑	🌐
Unlimited (default)	55%	72%	63%	88%	29%	59%	68%	42%	55%	0.68	0.68	0.68
1	55%	65%	60%	80%	37%	58%	65%	47%	56%	0.61	0.61	0.61
2	56%	74%	65%	87%	34%	60%	68%	47%	58%	0.63	0.63	0.63
3	57%	80%	69%	92%	32%	62%	70%	45%	57%	0.66	0.66	0.66
4	54%	66%	60%	85%	27%	56%	66%	39%	52%	0.62	0.62	0.62
5	56%	79%	68%	92%	31%	61%	70%	45%	57%	0.65	0.65	0.65
6	55%	80%	68%	93%	26%	59%	70%	39%	54%	0.69	0.69	0.69
7	56%	77%	69%	70%	45%	57%	69%	45%	57%	0.67	0.67	0.67
8	65%	47%	58%	56%	79%	68%	92%	31%	61%	0.69	0.69	0.69
9	55%	72%	63%	88%	30%	59%	68%	42%	55%	0.69	0.69	0.69
10	57%	84%	70%	93%	32%	63%	71%	47%	59%	0.68	0.68	0.68

Although this analysis revealed some improvements in model performance while varying such a parameter, we found that the impact of not tuning it was rather small (+4 percentage points for F-Measure and +0.01 for AUC). Thus, we believe this is not the cause of the negative results we obtained.

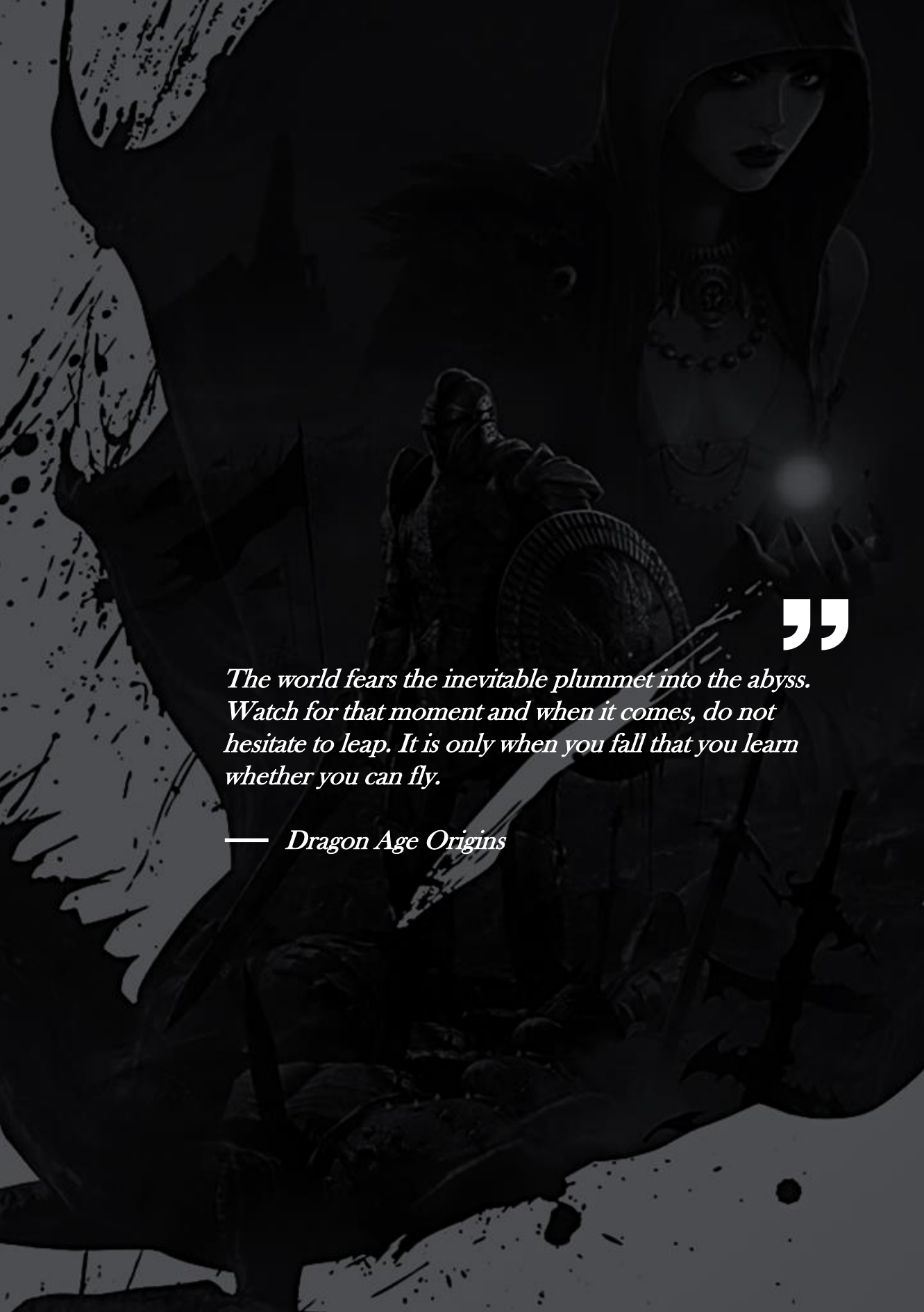
The classes we consider for the multi-class categorization problem (RQ_2 might be incomplete: It is possible that we do not consider some relevant categories of issues. To mitigate this threat, we avoided defining such categories based on our personal experience, but we relied on a state-of-the-art taxonomy [156]. A key threat regards the features considered for step 2 (and, thus, to answer RQ_2). It is worth noting that we relied on features that proved to be useful in other contexts (*e.g.,* categorization and clustering of mobile app reviews [33, 138]), and we also augmented them with video-based features. Still, it is possible that a different set of features leads to better results. As for clustering (both RQ_3 and RQ_4), it is possible that we chose sub-optimal parameters (*i.e.,* ϵ values). To reduce this threat, we used a rigorous procedure [111] to set these values for each tested video game.

Finally, **threats to external validity** concern the generalizability of our findings. Our test set is composed of gameplay videos related to only 3 video games. We could not select videos from a more diverse set of video games because we needed multiple segments related to the same game areas to address RQ_3 and RQ_4. However, it is worth noting that we also report in Table 3.5 and Table 3.6 the results of a 10-fold cross validation performed on the training set, which, differently from the test set, is composed of videos from many video games 110, specifically. Nevertheless, we acknowledge that most of our results are not necessarily generalizable to the vast quantity of video game genres and video games available in the market.

We believe that the variety of video games is not as relevant as the variety and type of streamers involved. GELID heavily relies on spoken content for segmentation and categorization. To this end, having verbose streamers could benefit GELID. On the other hand, the video game selection might mostly impact the two clustering-related steps: For example, games with many graphically similar levels or areas might deceive GELID while it cluster segments.

3.7 Final Remarks

In this chapter we present GELID, a novel approach for detecting anomalies in video games from gameplay videos to support developers by providing them with useful information on how to improve their games. We validated the single steps of GELID in an empirical study involving 604 segments extracted from 80 hours of gameplay videos related to 3 video games (Conan Exiles, DayZ, and New World). We obtained mixed results: The effectiveness of both segmentation (step 1) and issue-based clustering (step 4) are satisfactory, while we observed that categorization (step 2) and context-based clustering (step 3) of segments still do not work sufficiently well to be used in practice. To foster research in this field, we publicly release all the datasets in our replication package [71].



The world fears the inevitable plummet into the abyss. Watch for that moment and when it comes, do not hesitate to leap. It is only when you fall that you learn whether you can fly.

— *Dragon Age Origins*

CHAPTER 4

Identifing Stuttering Events

Contents

4.1 Introduction

One of problems more relevant for video games is constituted by drops in the frame rate (also referred to as “stuttering”). Such an issue makes the user experience not only less entertaining, but even frustrating when it creates impediments for properly playing the game.

The relevance of stuttering events for game developers is well-known. Politowski *et al.* [120] reported that the developers partially automated game

performance testing for two out of the five games considered in their study. Naughty Dog reportedly employed specialized profiling tools [153] while developing and testing *The Last of Us* for detecting stuttering events. Finally, Truelove *et al.* [156] report that game developers agree that *Implementation response* problems may severely impact the game experience.

Researchers recently introduced techniques to detect areas of the game affected by stuttering (see *e.g.,* [158]). While these techniques could help in game testing, it is practically impossible to thoroughly test all the possible interactions between the player(s) and the environment. Therefore, stuttering events might still occur in specific conditions (*e.g.,* when many players are in the same area of the game) that are not detected before the release and can only be reported by the end users. This could happen through classic bug-reports possibly accompanied by videos documenting the observed behavior. Video-based bug reports become fundamental to report issues such as stuttering events. Besides bug reports explicitly opened by end users, stuttering events are implicitly documented in the million of hours of gameplay videos available in platforms such as Twitch [1] or YouTube [2]: Twitch content creators stream, on average, 2.2M hours of videos every day [159], most of which are gameplay videos. Consider, for example, the gameplay video at `https://youtu.be/1LHHLaSRW8Y?t=79`. At minute 01:22, a stuttering event starts and lasts for a few seconds. Developers could use the information provided in the video to localize the problem and fix it. Previous work explored detecting different types of issues [70, 157], but no previous work specifically focused on stuttering events.

In this Chapter, we aim at tackling the problem of automatically finding candidate stuttering events in gameplay videos. We preliminarily surveyed 26 professional game developers, aiming at understanding to what extent solving such a problem would be relevant in the first place. Our results suggest that practitioners would be interested in such an approach and that it would be complementary to tools that they already use (*e.g.,* telemetry [179]).

It might be argued that finding stuttering events is conceptually easy: By

Figure 4.1: Gameplay video with extraneous elements

definition, stuttering occurs when two or more subsequent frames in a gameplay video are identical. However, there are several problems to address. First, there are game scenes in which identical consecutive frames do not indicate stuttering (*e.g.,* game settings). Thus, a naive approach that checks the equality of consecutive frames might falsely report stuttering events. Second, there might be parts of gameplay videos that do not show the video game itself but other elements, such as the streamer's face.

Fig. 4.1 depicts a concrete example of a gameplay video showing, besides the game itself, external elements such as the player recording the video. In this case, checking the equality of subsequent frames is also not enough, since while the "game-related pixels" might not change (due to stuttering), the ones depicting the player may change, thus missing the stuttering identification.

To solve those problems, we introduce HASTE (**H**inter for g**A**me **ST**uttering **E**vents). HASTE works in four steps. First, it splits a gameplay video into visually coherent slices, based on major changes in the color schema of the frames shown on screen. Second, HASTE automatically classifies each extracted slice

as *gameplay* or *non-gameplay* (*e.g.,* game settings, advertisement) using a machine-learning model. Third, it further analyzes each video slice classified as *gameplay* to identify the parts of the frame actually showing the game (*i.e.,* excluding unrelated elements such as the player or an on-screen chat). Finally, HASTE checks each pair of consecutive frames and it identifies stuttering events in which the relevant pixels (*i.e.,* the ones depicting the game) are (almost) identical.

We validate the steps behind HASTE on a total of 105 videos, of which about 10 hours of contents manually analyzed by the evaluators. We found that HASTE is accurate both in determining slicing points for the videos (76% precision) and detecting *non-gameplay* events to filter them out (90% precision and 88% recall). In terms of stuttering event identification, HASTE is largely more effective than three simpler baselines with which we compared it, achieving, overall, 71% recall and 89% precision. Finally, we interviewed two senior game developers aiming at understanding the applicability of HASTE in an industrial context. They generally provided positive feedback on the usefulness and applicability of HASTE in a developer's workflow due to the possibility of accessing a large amount of data available online. In addition, they highlight a major challenge related to the use of gameplay videos in the first place: HASTE might find false positives, *i.e.,* stuttering events that are not related to the gameplay video but to other incidental circumstances (*e.g.,* screen recording software, specific hardware configurations). This is a problem shared with several approaches and tools typically used by developers (*e.g.,* static code analysis tools). Still, HASTE could provide some guidance to let developers understand what they should focus on.

The remainder of this Chapter is organized as follows. In Section 4.2, we provide a motivational study to understand the relevance on detects stuttering events. In Section 4.3 we present HASTE and its components in details. In Section 4.4 we describe the empirical study design, while in Section 4.5 we report the obtained results. In Section 4.6 we report the design and the results

related to a semi-structured interviews conduct with game developers in order to evaluated the level of interest in HASTE. In Section 4.7 we provide an overview of “HASTE-Web: Gameplay Video Analysis”, a web application developed based on the HASTE approach. In Section 4.8 we report the threats to validity, while in Section 4.9 concludes the chapter.

4.2 Motivating Study

We ran a survey with game development practitioners to understand the possible relevance of an automated approach that detects stuttering events in gameplay videos.

4.2.1 Survey Design

The *goal* of this study is to answer the following research question:

> RQ_0: *To what extent is it useful to identify stuttering events in video games through the use of gameplay videos?*

The *context* is represented by objects (*i.e.,* a survey) and subjects (*i.e.,* 26 practitioners from the game development industry).

Participants Selection. We recruited participants through Prolific [1], a platform that helps to select participants for research studies. We looked for candidate participants who (i) were located in the United States and Europe, (ii) had experience in the field of “Computer Science” and “Information Technology (IT)”, and (iii) declared to be video game enthusiasts. We initially decided to also have as filter an active job in the video game industry. However, this resulted in only two candidates available on Prolific. Thus, we removed such a filter and relied on the initial questions of the survey to discard participants without game

[1] https://www.prolific.co/

development experience. Based on the filters applied, we invited 248 candidate participants. We payed 5$ to each participant who completed the survey in all its parts (*i.e.,* by also providing relevant answers to the open questions).

Survey Questions. Participants were initially presented with a welcome page, which explained the goal of the study, reported its expected duration ($\sim$15 minutes) and other basic information. If they agree to participate, the main survey started, which consisted of four phases. In the first one we collected *demographic information*. As a preliminary question, we asked whether the participant had experience with game development or testing. If the answer to the latter question was no, the survey stopped and the participant was excluded from the study. Otherwise, we asked additional information (*i.e.,* job position and number of years of experience, examples of video games they worked on).

In the second phase, we asked questions aimed at understanding *whether and how developers identify stuttering events*. Specifically, we asked (i) if they actively try to identify stuttering events during beta testing of a video game; (ii) the percentage of bug reports that regards stuttering events, in their experience; and (iii) if they use telemetry to detect stuttering events. Based on the last question, the participants who declared to use telemetry were asked to indicate on a Likert scale from 1 to 5 (the higher the better): (i) what accuracy level do the telemetry-based tool achieve in detecting stuttering events, and how complete the information telemetry provides is to (ii) reproduce the issues and (iii) fix them. The participants had to motivate all their answers in open questions. Finally, we asked if they use other tools to identify stuttering events.

In the third phase, we asked questions aimed at understanding *whether developers use data available online to identify stuttering events in video games*. Specifically, we asked which platforms they usually get data from (YouTube, Twitch, Steam, Discord, or others) and to add details on how they do that (open answer).

In the fourth phase, we asked question to *assess the possible usefulness of an automated approach for identifying issues in gameplay videos*. Specifically,

we asked participants to rate, on a Likert scale from 1 to 5 (the higher the better), the extent to which parts of the gameplay video documenting stuttering events would be useful for (i) replicating the problems and (ii) fixing them. We also explicitly asked what other pieces of information would be useful to replicate and fix errors that cause stuttering events. Finally, we asked (iii) to what extent they would be interested in using an automated approach for detecting stuttering events from gameplay videos on a video game they are developing/testing, and (iv) whether they would use this approach in combination with other testing approaches they already use or to replace them. Also in this case, we asked to motivate the answers. We implemented our survey in Google Forms.

Data Collection and Analysis. We kept the survey available to the invited practitioners for eight days because we got immediate feedback from most candidate participants. We collected a total of 53 responses: 3 have been automatically rejected by the platform because the participants exceeded the allowed time limit; 6 have been excluded since the participants declared no experience in game development/testing.

We manually analyzed the remaining 44 to assess the quality of the responses provided by the participants. We discarded the responses for which the open questions were not filled with relevant content, which we used as proxy for the commitment of the participant.

To evaluate the relevant content, we analyzed the individual responses of the participants. In detail, we analyzed the completeness and consistency of open-ended responses. Content was assessed as relevant when the response was complete and directly related to the question. We report some examples of responses that were not considered consistent with the question asked and, thus, excluded from the assessment. As for the question regarding the use of telemetry, a participant provided the definition of telemetry: “telemetry is the term used to denote any source of data [...] There are many popular applications of telemetry in games [...].” However, no motivation was provided. In addition, another participant provided for two different questions the same

generic answer. Specifically, when we asked what other pieces of information is needed to reproduce the stuttering event and to fix a problem, such a participant responded “Disable game DVR. Disable dynamic ticks and HPET. Disable visual effects. Disable Windows features you don’t need. Disable full-screen optimizations. [...] Increase virtual memory. Delete unnecessary programs. Update your RAM. Reset your BIOS to optimized defaults.”

We ended up with 26 valid responses. We summarize the quantitative data through boxplots and barplots, and report examples of interesting open answers we collected.

4.2.2 Results

Most of the participants (73%) reported that they have (or had in the past) experience as developers/testers in the game development industry, while the remaining 27% still had game development experience, but not in industry (*e.g.,* open-source projects). Out of the ones with industrial experiences, 26% are/were testers, 42% are/were developers, and the remaining 32% have/had other roles (*e.g.,* AI specialist). The average experience in game development of the participants is 2 years, with 19% of them having more than 3 years of experience. Fig. 4.2 summarizes the data collected for the subsequent parts of the survey we discuss below.

Identification of Stuttering Events. Most of the participants (76%) reported that they are interested in identifying stuttering events during the beta-testing of a video game. The average reported percentage of bug reports related to stuttering is $\sim$15%, with a median $\sim$10% (top-left part of Fig. 4.2). As expected, most of the participants use telemetry for detecting stuttering events (62%). While they mostly agree that telemetry is useful for detecting stuttering events (median/mode of 4), not all of them think that it can be useful for reproducing and fixing the issue (median for reproduction: 3.5; median for fixing: 4). Some participants reported that telemetry does not provide sufficient information

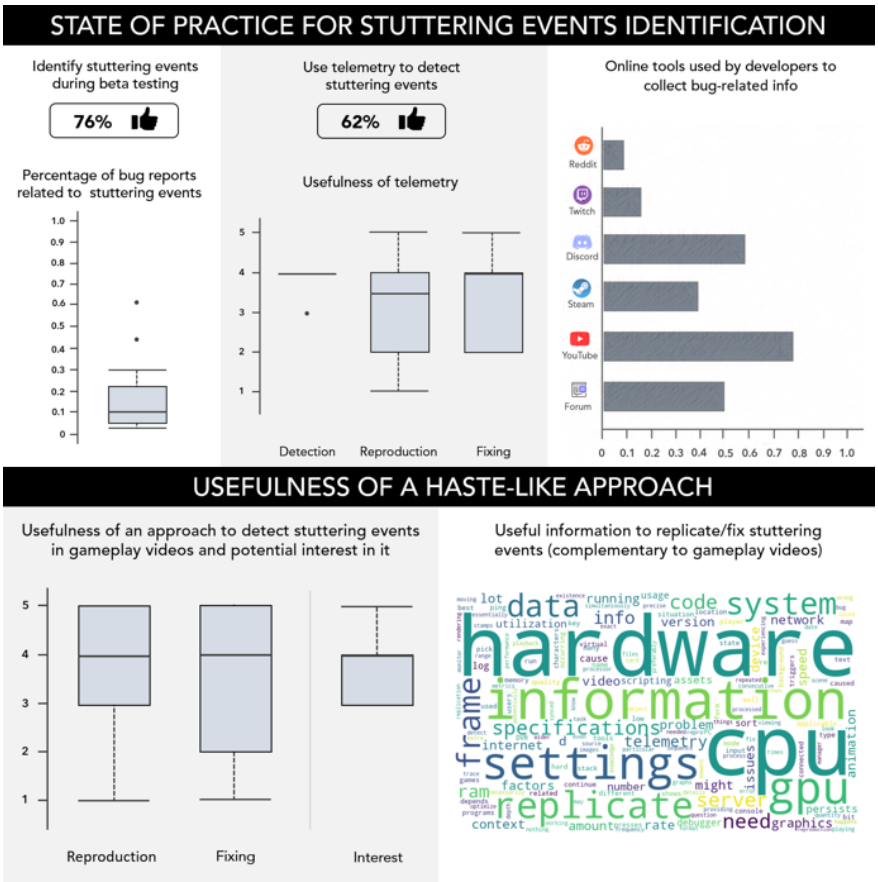


Figure 4.2: Survey results.

to pinpoint the cause of the issue. For example, one practitioner reported that, sometimes, stuttering events are hardware-related and, thus, difficult to reproduce without the exact setup of the end user.

Usage of Online Platforms. The majority of participants (77%) use YouTube to collect bug-related information about games, with several of them also focusing on more communication-oriented platforms such as Discord (58%) and forums (50%). Some of the participants claim to use YouTube to collect information about game issues since "... following a video that specifically addresses the problem is often the best approach. On the other hand another participant states that "Sometimes I search the forums for common problems in games and then try to find them in the game I am testing." In addition, some participants who use more than one platform say that "If something is public enough, then it should go back to the developers and testers [...] using social media is a great way to get information about what is causing any number of problems." We observed that the large majority of participants use YouTube, while a few of them use Twitch ($\sim$5%). While no participant explicitly reported this, it might be the case that they generally prefer the former over the latter because it contains higher-quality videos (often edited before being posted), while Twitch mostly features live/past-live videos.

Usefulness of an Approach for Identifying Stuttering Events in Gameplay Videos. The bottom part of Fig. 4.2 summarizes participants' feelings about the usefulness of an approach for automatically detecting stuttering events in gameplay videos. Most of them believe such an approach would be more useful to reproduce and fix stuttering-related issues than telemetry (even if marginally, given that the median, in this case, is 4 for both the questions). The word cloud in the lower-right part of Fig. 4.2 reports information developers might need to replicate/fix stuttering-related issues (besides gameplay videos). Most of them report *hardware* information (*CPU*- and *GPU*-related information). Finally, 86% of participants would like to use such an approach in combination with existing tools (*e.g.,* telemetry), while the remainder said that it could replace

them. None of the participants reported a lack of interest in testing such an approach (with a median of 4 and no answer below 3).

4.3 A Methodology for Stuttering Events Identification

In this section we present HASTE, an approach able to identify stuttering events through gameplay videos analysis. Fig. 4.3 overviews the steps behind HASTE. The process starts with a list of gameplay videos which developers want to analyze. These are supposed to be gameplay videos related to video games they maintain. The list of videos is provided to the *Gameplay Videos Crawler*, which is in charge of downloading them. Our current implementation supports the download of gameplay videos from YouTube and Twitch. More often than not, the downloaded videos include, besides the gameplay itself, also additional content which is not interesting to identify stuttering. For example, the player (*i.e.,* the person recording the video) might interrupt the gameplay to talk to the viewers. For this reason, HASTE implements a mechanism to identify, within a given video, the fragments representing actual gameplay. This is accomplished through two components: (i) the *Video Slicer* splits the video into different slices, based on drastic changes in the video frames (*e.g.,* the gameplay is interrupted to show an advertisement, thus causing most of the pixels on screen to change); (ii) a *Video Slice Classifier*, which is a machine learning model trained to discriminate between video slices showing and not showing gameplay. Video slices not classified as gameplay are discarded, while the "gameplay slices" are further analyzed. Indeed, it is not possible to just compare subsequent frames to check whether they are identical or not to identify stuttering events. This is due to the presence on screen of additional elements such as the camera recording the player (see *e.g.,* Fig. 4.1). For example, it is possible that despite a game stuttering and, thus, a perfect match between the pixels representing the game in two subsequent frames, the latter

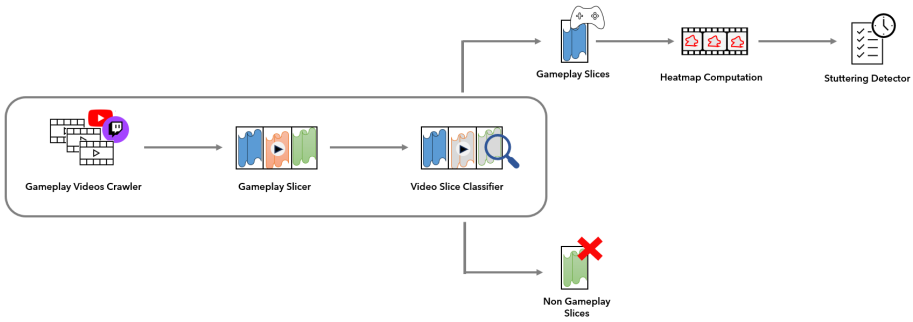


Figure 4.3: HASTE overview.

are different due to the player moving or to other elements appearing on the screen (*e.g.,* subscription notifications). For this reason, HASTE features a *Heatmap Computation* step aimed at identifying, in a given gameplay slice, the pixels of its frames which change more frequently.

These are supposed to be the pixels in which the game is displayed and allow to crop the frames to only focus on this part when computing the similarity between subsequent frames. Cropped subsequent frames being almost identical are identified as candidate stuttering. In the following we detail each of the HASTE's steps.

4.3.1 Video Slicer

Given a gameplay video as input, the *Video Slicer* extracts and stores all its frames. This is done by using the OpenCV library [126]. To make the following steps of HASTE faster, all frames are resized to a resolution of 320×180. Each pair of subsequent frames is then compared using their color histogram representation. Color histograms allow to summarize an image as the number of its pixels having a pixel falling in specific color ranges. Using color histograms

it is possible to compare images by checking whether their "distribution of colors" is similar. Histograms are a typical method used for extracting features from images and one of the most basic methodologies for computing image similarity [144]. Instead of using the classic RGB (Red, Green, and Blue) color space, we use HSL (Hue, Saturation, and Level). Such a color space allows to better detect the similarity between frames even when some effects are employed (*e.g.,* fade-in, fade-out, transitions). For example, if a fade-out effect is used, the hue and the saturation of all the pixels will likely remain the same, while the level (*i.e.,* the brightness) will uniformly reduce for all of them. A drawback of this representation is that the similarity computation ignores the shapes and texture of the compared images. However, given the goal of the *Video Slicer* (*i.e.,* splitting a video when drastic changes in the video frames are observed), a similarity computation based on color distributions is sufficient. For example, if a gameplay is interrupted by an advertisement, we expect substantial changes in the color distribution of the frames.

We compute the similarity between each pair of subsequent frames in the video using the `compareHist` function from OpenCV [126], which takes as input the two histograms to compare (*i.e.,* the two images) and the method to use for the similarity computation. For the latter, we use `HISTCMP_CORREL`, computing a correlation between the two histograms and thus returning a value between -1 and 1, with 1 indicating a high similarity between images. Following well-known guidelines used to interpret correlation coefficients [46], we split the video if two subsequent frames have a correlation lower than 0.3 (*i.e.,* no positive correlation). Such a process results in a set of video slices extracted from the input video. We discard slices shorter than 150 frames (5 seconds) since, as it will be clear later, HASTE exploits information extracted from video slices to automatically classify them as *gameplay* or *non-gameplay* slices. A slice shorter than 5 seconds does not contain enough information to allow for a reliable classification.

It is worth highlighting that the *Video Slicer* is not meant for extracting slices representing parts of the game that humans would find *semantically coherent* but rather at detecting *visually coherent* slices. For example, let us imagine a cutscene in which two visually different characters discuss in a shot-reverse shot in which they are alternately shown on screen. We are not interested in having a single semantically coherent slice for the whole dialogue: Having a slice for each (visually coherent) camera cut is good as well, as long as gameplay and non-gameplay parts (*e.g.,* advertisements) are not in the same slice since the next step assumes this for detecting and discarding non-gameplay slices.

4.3.2 Video Slice Classifier

Once extracted the slices from a video under analysis, they are provided as input to a machine learner in charge of classifying them as *gameplay* or *non-gameplay* slices. HASTE exploits the Weka [163] implementation of the Random Forest algorithm [25] for this task. The Random Forest builds a collection of decision trees with the aim of solving classification-type problems, where the goal is to predict values of a categorical variable from one or more continuous and/or categorical predictor variables. In our work, the categorical dependent variable is the type of video slice (*i.e., gameplay* or *non-gameplay*), and we use features extracted from the video slice as predictor variables. Two families of features are exploited: heatmap-based and similarity-based. We detail in the following these features, while in our study design (Section 4.4) we explain how we trained and evaluated the classifier.

Heatmap-based features. Given the set of frames $S = \{F_1, F_2, \ldots, F_n\}$ in a given slice, we start by creating a 320 $\times$ 180 HM matrix in which each entry represents a pixel of the frames in S (*e.g.,* the entry in position $[1, 1]$ represents the pixel in the top-left corner of each frame). At the beginning, all HM entries are initialized to 0. Then, we perform a pixel-by-pixel comparison between each subsequent pair of frames in S (*e.g.,* F_1 *vs* F_2) and, if a pixel in position $[i, j]$ has

a different color in the two frames, we increase by one the value of $HM[i, j]$. At the end of this process, HM will represent a heatmap showing how frequently each pixel in the S's frames changed. Once created HM, we compute its minimum, maximum, median, average, first and third quartile, and use those six statistics as features for the Random Forest. Our assumption is that the frequency with which the pixels change could help in discriminating *gameplay* slices from *non-gameplay* slices. For example, we assume that *non-gameplay* slices in which the gamer is talking in full screen foreground tend to have less pixel-level changes as compared to *gameplay* slices featuring the video game.

Similarity-based features. We start from the same set of frames S in the slice and compute the correlation between the color histograms of each subsequent pair of frames using `HISTCMP_CORREL`.

This process results in a distribution of correlations having $n - 1$ values, where n is the cardinality of S. Indeed, if $S = \{F_1, F_2, F_3\}$, we compute the following correlations (Cr): $Cr(F_1, F_2)$ and $Cr(F_2, F_3)$. For such a distribution we compute the same six descriptive statistics previously described and provide them as additional features to the Random Forest. The rationale here is that the color histogram similarity between frames could be different between *gameplay* and *non-gameplay* slices. For example, an advertisement on screen is likely to exploit a more variegate and dynamic set of colors as compared to a gameplay, which is usually characterized by a quite stable color scheme for a given game scene.

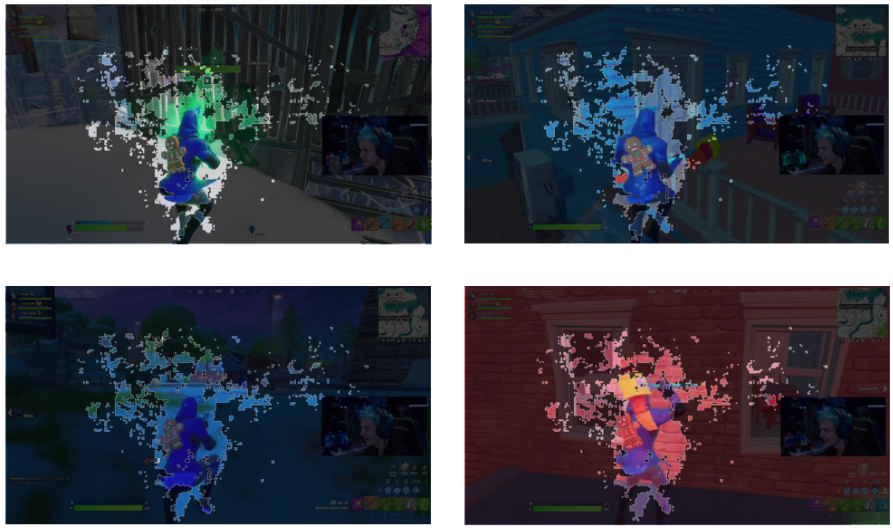


Figure 4.4: Example of heatmap applied to several frames of the same gameplay video slice.

4.3.3 Cropping Frames Using the Heatmap

Video slices classified by the Random Forest as *gameplay* are further analyzed to identify in them the pixels representing the actual game. Indeed, as previously explained, before looking for stuttering events we must be able to focus the attention on the pixels relevant for the game.

For a given *gameplay* video slice, we exploit the previously built heatmap HM and compute its 90% percentile. This means identifying the top-10% of pixels which more frequently change in the slice. It is indeed reasonable to think that the portion (*i.e.,* subset of pixels) showing the actual game on screen is the one changing more frequently, since it represents the core action of the game. Instead, parts of the screen showing ads, chat messages, or the streamer, are likely to change less frequently.

All pixels falling below the 90% percentile are removed from all frames in S (*i.e.,* the frames of a given video slice). In Fig. 4.4 we show an example of how such a process allows to focus the attention only on the part of the frames featuring the game in action. In this example we use a transparency to still make it visible the part that would be cut from the frames (*i.e.,* the one in black). Since all frames belong to the same video slice, they are all cropped using the same heatmap. The output of this step are cropped frames belonging to each *gameplay* slice (*i.e.,* frames only including the top-10% of changing pixels within the slice).

4.3.4 Stuttering Detector

In the last step of HASTE, the cropped frames are provided as input to the *Stuttering Detector* which is in charge of comparing them looking for candidate stuttering events. The stuttering identification is based on a simple idea: If subsequent cropped frames are identical, we can safely assume that a stuttering event is happening, since the cropped frames, as explained, are supposed to

focus on the game action. Given the set of cropped frames in a video slice, we compute the similarity between each pair of subsequent frames by using the Structural Similarity Index (SSIM) [162]. SSIM has been proposed by Wang *et al.* as a way to overcome the shortcomings of classic image similarity techniques (such as the previously described color histogram similarity) and it is based on the idea of comparing the images' textures. The evaluators showed that the SSIM is the metric better capturing image similarity as perceived by humans [162], which is what we need to identify stuttering events (*i.e.,* cases in which humans perceive a high similarity between subsequent frames, giving the impression of lagging). We used the SSIM implementation available in the *scikit* image library [6], which returns a value between 0.0 and 1.0, with the latter identifying identical images.

We opted for a quite conservative detection of stuttering events since our goal is not to flood developers with recommendations about possible stuttering events, but rather to provide precise recommendations (even at the cost of losing some valid data points). In particular, the *Stuttering Detector* reports a candidate stuttering if the SSIM is at least $1 - \epsilon$ for two consecutive pairs of cropped frames. ϵ is a small number that developers can choose to increase or decrease the number of candidate stuttering events returned by HASTE. If ϵ is 0, even a small imprecision in the previous phases (*e.g.,* in cropping) would result in a missed detection of a stuttering event. Therefore, we set $\epsilon = 0.0001$, so that, given the resolution of our frames (320 $\times$ 180 = 57,600 pixels), HASTE is able to tolerate a 5-pixel noise. We say that a stuttering event is there if *two* consecutive pairs of cropped frames meet the similarity threshold to further deal with possible imprecisions of the frames-cropping stage: When we find two subsequent frames meeting our threshold, we ask for a "confirmation" to the next pair of frames as well.

The number of consecutive pairs that should meet the threshold can be further increased in case developers want to reduce the chances to obtain false-positive recommendations. Also, cases in which more than 30 subsequent pairs

of frames meet the similarity threshold are not treated as stuttering, since they are more likely to be due to events such as the player pausing the game. The final output is a set of candidate stuttering events, reporting, for each analyzed video, the frames at which the issues start (if any).

4.4 Empirical Study Design

The *goal* of our study is to evaluate the effectiveness of HASTE in automatically identifying stuttering events in gameplay videos. In particular, we want to assess HASTE in terms of its: (i) accuracy in automatically splitting gameplay videos into visually coherent slices; (ii) ability to correctly classify actual gameplay video slices; and (iii) ability to automatically identify stuttering events in gameplay videos. We aim at answering the following RQs:

> RQ_1**:** *What is the accuracy of HASTE in automatically splitting gameplay videos into visually coherent slices?*

RQ_1 assesses the accuracy of HASTE in automatically splitting gameplay videos into visually coherent slices (*e.g.,* fragments showing the same game scene, an advertisement). RQ_1 evaluates the *Video Slicer* (Section 4.3.1).

> RQ_2**:** *To what extent is HASTE able to correctly classify video slices relevant to the gameplay?*

RQ_2 evaluates the *Video Slice Classifier* (Section 4.3.2) focusing on its ability to classify the video fragments extracted by the *Video Slicer* as *gameplay* or *non-gameplay*.

> RQ_3**:***To what extent is HASTE able to automatically identify stuttering events in gameplay videos?*

Our last research question evaluates HASTE as a whole, verifying whether the stuttering events it identifies are true positives.

4.4.1 Context Selection

We built three video datasets, each one aimed at answering one of the formulated research questions. All datasets have been extracted from either Twitch or YouTube. For the former, the *Gameplay Videos Crawler* relies on the Twitch command-line interface (CLI) [74] while for the latter it exploits the Pytube library [127]. Both streaming platforms allow to download videos by specifying characteristics of interest such as their resolution and FPS.

RQ_1: Dataset for the evaluation of the Video Slicer

We collected 20 gameplay videos from Twitch starting from the list of most popular videos, which is based on the popularity of the streamer and on the number of visualizations. We targeted English videos having a duration between 10 and 40 minutes and being downloadable at 30 FPS at least. Concerning the length thresholds, these have been defined to focus on videos which are long-enough to justify the usage of the *Video Slicer* (*e.g.,* slicing a 30-second video would probably be useless) while considering a maximum length representative of most videos uploaded on these platforms. Table 4.1 reports the exact duration of each video. Overall, this dataset features over seven hours of gameplay videos. All videos have been dowloaded in *.mp4* format in 1280 $\times$ 720 pixels resolution at 30 FPS.

RQ_2: Dataset for the evaluation of the Video Slice Classifier

To evaluate the *Video Slice Classifier* we need to build a dataset to train and evaluate the Random Forest model. This means creating a dataset of video

slices labeled as *gameplay* or *non-gameplay*. Manually building such a dataset would be quite expensive. Thus, we designed the following automated process.

To collect instances of *non-gameplay* slices, we downloaded the top-5 videos (in terms of visualizations) from each of the top-10 YouTube non-game-related video categories (*i.e.,* basketball, cooking, football, Formula 1, motors sport, movies, music, news, sport, trends). Then, we run on them our *Video Slicer* and collected a total of 2,945 slices that we can safely label as *non-gameplay*. Similarly, as representative of *gameplay* slices, we downloaded 5 gameplay videos from each of the six game-related categories in YouTube (*i.e.,* action, adventure, beat, riders, shooter, sport-game). We made sure that the downloaded videos only contained gameplay scenes, without any sort of interruption due to advertisements, the gamer speaking, *etc.* Again, we run the *Video Slicer* on the 30 gameplay videos, obtaining 1,546 slices labeled as *gameplay*. This is the dataset on which the Random Forest will be trained and tested.

RQ_3: Dataset for the evaluation of HASTE as a whole

The third and last dataset aims at assessing the ability of HASTE in identifying stuttering events in gameplay videos. We selected 10 videos from the list of popular gameplay videos on YouTube. These videos have not been used to build the previously described datasets. Given the goal of RQ_3, we selected videos containing in their meta data (*i.e.,* title, description, comments) the word "stuttering". These videos should provide data points useful to assess the ability of HASTE in identifying stuttering events. Table 4.3 reports the duration of the 10 selected videos, which is $\sim$2 hours, in total.

4.4.2 Data Collection and Analysis

To address RQ_1 we ran the *Video Slicer* on the set of 20 Twitch gameplay videos, collecting a total of 1,909 extracted slices. Out of these, we manually analyzed a sample of 320 video slices, ensuring a margin of error of $\pm 5\%$ with a confidence level of 95%.

The estimation has been performed applying a sample size calculation formula for an unknown population [133]. The goal of the manual validation was to assure that the splitting performed by HASTE actually resulted in the creation of visually coherent slices, representing *e.g.,* a specific game scene, an introductory countdown, *etc.* Each of the 320 manually analyzed slices has been independently inspected by two evaluators who classified them as "visually coherent" or not. Conflicts, arisen for 13 slices (4%), have been solved through an open discussion. We report the precision of the *Video Slicer* as the percentage of reported slices which have been classified as visually coherent. Such an evaluation lacks an assessment of the recall ensured by the *Video Slicer*. In other words, we are not assessing whether points in the video which should have resulted in new slices have been missed by HASTE. To partially address this limitation, when manually inspecting the 320 slices we also verified whether each of them contained additional "valid splitting points" that have been missed by HASTE. We also report this data when answering RQ_1.

To answer RQ_2, we trained and tested the Random Forest classifier on the dataset of 1,546 *gameplay* and 2,945 *non-gameplay* video slices previously described. We used the WEKA's default configuration for the Random Forest classifier, *i.e.,* we set the number of trees to 100, the number of randomly chosen attributes to 0 and the maximum depth of the trees to "unlimited". We used a 10-fold cross validation to assess the performance of the trained model. Since our dataset is slightly unbalanced (66% of the slices are *non-gameplay*), we also experimented with re-balancing our training set in each of the 10-fold iterations using SMOTE [32], an oversampling method which creates synthetic samples

from the minor class. Since we did not observe major improvements (results in our replication package [69]), we discuss the results without re-balancing. In particular, we report the confusion matrix output of the 10-fold validation (and, thus, the true positives, true negatives, false positives, and false negatives) and the corresponding recall and precision values for both the *gameplay* and the *non-gameplay* categories. We compare our approach with a very simple baseline that reports all the video segments as gameplay videos.

To address RQ_3 we started by manually building an oracle reporting stuttering events in the 10 YouTube videos composing the dataset for this RQ. The first evaluator looked at the overall 2 hours of videos with the goal if identifying stuttering events. The latter were documented by writing down the second in the video and the corresponding frame at which the event started. The 44 candidate stuttering events identified have been further validated by a second author, who confirmed 41 of them (7% of conflicts). An open discussion aimed at solving conflicts led to the final creation of our oracle, composed by 42 stuttering events spread across the 10 videos as reported in Table 4.3. HASTE has then been run on the 10 videos, collecting the 62 candidate stuttering events reported by it. Based on these, we compute its recall and precision when considering the built oracle as the ground truth.

We also manually analyzed the reported stuttering events that were outside of the oracle (*i.e.,* that we did not identified by watching the video). Indeed, the perception of a stuttering event is quite subjective and subtle in some cases (micro-stuttering). Thus, even instances not present in our oracle might be true positives. Also this analysis has been performed independently by two evaluators, with conflicts arisen on 1 (1.61%) of the inspected instances. Such an analysis allows to compute an overall precision for HASTE (*i.e.,* how many of the stuttering events it reports are true positives).

To better interpret the performance of HASTE, we compare it with two main baselines. The first, named *SSIM baseline*, is basically the last step of HASTE (*i.e.,* the *Stuttering Detector*) without all previous components in

the pipeline. This means that a stuttering event is identified if at least two pairs of subsequent frames have a SSIM $\geq 1 - \epsilon$ (*i.e.,* 0.9999, in our case). Differently from the complete approach, the SSIM is computed (i) between all pairs of subsequent frames in the video since the *non-gameplay* slices are not discarded by the Random Forest, and (ii) on the entire frame, including external elements shown on screen but unrelated to the gameplay (*e.g.,* the player), since cropping is not applied. The *SSIM baseline* allows to assess the boost in performance provided to HASTE by all steps preceding the similarity computation. The second baseline, *Pixel-sim baseline*, is the most simple and natural approach one could devise to detect stuttering events. It computes a pixel-by-pixel similarity between subsequent frames in a video, and it says that there is a stuttering event when an exact match between them is found. In addition to those two main baselines, we also compare HASTE with a version of HASTE (HASTE ^{NoFiltering}) that does not rely on the Video Slice Classifier component, *i.e.,* which does not filter out non-gameplay. This would further allow to highlight the usefulness of such a step.

We run the baselines on the same dataset of videos used for HASTE computing, also in this case, the precision and recall with respect to the manually built oracle. The *Pixel-sim baseline* and *HASTE* ^{NoFiltering} reported a total of 95 and 213 candidate stuttering events, respectively. We manually classified all instances not matching the ones in the oracle using the same procedure previously described for HASTE and computed the baseline precision. Since the *SSIM baseline* reported a much higher number of stuttering events (388), we performed the same analysis, but on a statistically significant sample (95%$\pm$5% confidence) of those not matching our oracle (186 manually analyzed instances).

We publicly release the implementation of HASTE and of the experimented baselines. We also release in our replication package the datasets used in each research questions, the `arff` files used to train and test the Random Forest, and the raw data of our manual analyses [69].

Table 4.1: RQ_1: Accuracy of the Video Slicer

Video	Length	#Identified	#Analyzed	TP	Precision	#Missed
1	30:46	563	94	58	0.62	8
2	13:41	63	11	7	0.64	2
3	20:28	78	13	10	0.77	1
4	16:15	53	9	9	1.00	1
5	29:17	143	24	19	0.79	4
6	15:42	61	10	7	0.70	3
7	23:53	97	16	16	1.00	2
8	19:33	180	30	20	0.66	2
9	20:39	35	6	6	1.00	0
10	35:17	28	5	4	0.80	2
11	19:45	52	9	9	1.00	1
12	29:11	108	18	13	0.72	3
13	17:56	5	1	1	1.00	0
14	24:13	83	14	12	0.86	0
15	13:10	7	1	1	1.00	3
16	18:42	25	4	4	1.00	1
17	22:36	38	6	6	1.00	0
18	23:48	22	4	3	0.75	1
19	28:49	254	43	37	0.86	4
20	17:27	14	2	2	1.00	0
Overall	**7:21:08**	**1,909**	**320**	**244**	**0.76**	**38**

4.5 Empirical Study Results

We discuss the achieved results by research question.

4.5.1 RQ_1: Evaluation of Video Slicer.

Table 4.1 reports the accuracy of the *Video Slicer* component in identifying valid splitting points in the provided videos.

For each video, we report: (i) its ID, which can be mapped to our replication package [69]; (ii) its length; (iii) the number of slices (*i.e.,* frames in which the video should be split) identified by HASTE; (iv) the size of the sample

we manually analyzed; (v) the true positive (TP) instances we identified in the analyzed sample (*i.e.,* points in which the splitting was valid); (vi) the corresponding precision, computed as the number of TPs divided by the size of the analyzed sample; and (vii) the number of missed slices (*i.e.,* additional splitting points we identified that were missed by HASTE). Worth commenting is the number of slices identified by HASTE (1,909), an average of 95 per video. Such a number may look surprisingly high. However, it is worth remembering that the goal of the *Video Slicer* is not to extract slices that are meant to be visualized by humans, but to make sure that each slice embeds a set of frames having a very similar graphical layout.

Indeed, this is crucial for the subsequent steps of HASTE. For example, computing the HM heatmap on a set of frames visualizing completely different content would not make sense, since it would mean focusing on the top-10% of changing pixels which, however, may represent different objects (*e.g.,* in a frame, the game action, in another frame, an advertisement). Thus, the slicing must cluster together similar subsequent frames into one slice, creating a new slice when the pixels on screen substantially change. Based on our analysis, 76% of the identified splitting points are correct, representing major changes of the content depicted on screen. One may argue what false positives are in this context. In other words, how is it possible that an approach based on image similarity may identify wrong split points (*i.e.,* subsequent frames that basically represent the same scene but that are perceived as different by HASTE and thus split). This happens, for example, when a special effect is shown on screen for a few seconds (*e.g.,* some smoke appears in the game for a few frames), leading the color histogram similarity to low values between the last frame without the special effect and the first frame showing it. While HASTE slices the video at this point, this is a wrong decision, since both the frames before and after the special effect represent the exact same scene and, thus, could in theory share the same heatmap.

Figure 4.5: Example of gradient effect resulting in the loss of a valid splitting point

Concerning the "#Missed Slices" column, in the 320 slices we manually inspected, we found 38 valid splitting points missed by HASTE. These are mostly due to the application of a gradient effect when the video moves from one scene (*e.g.,* the player introducing the following video content) to another (*e.g.,* the actual gameplay). The gradient effect applied over a set of frames $\{F_1, F_2, \ldots, F_n\}$ makes HASTE failing since there is no drastic change from a frame F_i to a frame F_{i+1}, with the image on screen slowly changing. However, such a process results in an overall drastic change between F_1 and F_n, which is missed by HASTE. Fig. 4.5 shows a concrete example of the gradient effect resulting in the loss of a valid splitting point.

The issue of missed slices, while minimal in our current dataset, poses a potential challenge for datasets with a prevalent use of gradient transitions between scenes. his effect results in a gradual change that HASTE currently fails to detect due to its reliance on identifying significant frame-to-frame changes. A possible basic solution is to extend the comparison between frames, in addition to the immediately following frames, to a larger time window, which could help identify the cumulative effect of gradual changes. By analyzing the difference between frames in a specific interval, HASTE could detect significant transitions occurring after a series of smaller changes. Alternatively, a more expensive method might include using machine learning techniques on a dataset that includes a significant number of gradient transitions. A model could allow the recognition of gradual change patterns typical of gradient effects, enabling the identification of cutoff points that would otherwise go undetected.

4.5.2 RQ_2: Evaluation of Video Slice Classifier.

Table 4.2 reports the confusion matrix resulting from the classification performed by the *Video Slice Classifier*.

The rows report the *gameplay* (GP) and the *non-gameplay* (NGP) slices in the oracle, which have been classified by HASTE as reported in the columns. For example, out of the 1,537 GP slices in the oracle (1,174 + 363), 1,174 have been correctly identified by HASTE, while 363 have been misclassified as NGP. This results in a recall of 0.77 for the GP slices. The precision for GP slices is 0.80, since HASTE wrongly classifies 296 NGP slices as GP. Thus, eight out of ten slices identified as GP by HASTE are actual gameplay slices.

The precision and recall values are even better for the identification of NGP slices. While the latter are not at the core of HASTE, the excellent results achieved demonstrate that the features used for training the *Video Slice Classifier* are able to filter out slices that are irrelevant for the identification of stuttering events.

The Random Forest does also pair each prediction with a "confidence level", a value between 0.50 and 1.00 that indicates how confident the model is in the provided classification. Such a confidence level is computed as the number of classification trees in the forest that "voted" for a specific output (*e.g., gameplay*). For example, an output prediction $\langle$*gameplay*, 0.90$\rangle$ indicates that the model is 90% confident about the *gameplay* classification. We studied how the confidence of the classifications impacts their quality. In particular, we verified whether by setting a threshold on the confidence of the classification it is possible to effectively exclude false positives (*i.e.,* NGP classified as GP).

Table 4.2: RQ_2: Automatic classification of video slices as *gameplay* and *non-gameplay* by Video Slice Classifier and the baseline.

Video Slice Classifier	**GP**	**NGP**	**Precision**	**Recall**
GP	1,174	363	0.80	0.77
NGP	296	2,657	0.90	0.88
Baseline	**GP**	**NGP**	**Precision**	**Recall**
GP	1,573	0	0.35	1.00
NGP	2,953	0	0.00	0.00

Such a scenario could be interesting for developers who want to receive less "stuttering reports" from HASTE which, however, are more likely to belong to actual gameplay slices. We found that by only considering classifications having a confidence of at least 0.70 (other slices are just discarded as if they are NGP), the precision in the identification of GP slices raises to 0.89 and it further increases to 0.98 by only considering classifications having a confidence of at least 0.90. This has a price to pay in terms of recall which drops to 0.56 (for the 0.70 confidence) and 0.27 (0.90). Still, our analysis shows that using a high threshold on the classification confidence can be a viable solution for developers interested in receiving less, but more likely to be correct, recommendations.

The baseline naturally achieves a perfect recall of 1, which is higher than the one achieved by Video Slice Classifier (0.77) on gameplay segments. However, it achieves a very low precision on such a class (0.35). The practical implication of such low precision is an unacceptably high rate of false positives, which could undermine the usefulness of the classifier in real-world applications.

4.5.3 RQ_3: HASTE for Identifying Stuttering Events.

Table 4.3 compares HASTE and the two baselines in terms of recall and precision achieved with respect to the oracle, *i.e.,* the set of 42 stuttering events we manually identified in the inspected $\sim$2 hours of videos. For each approach

Table 4.3: RQ_3: Recall and precision in the identification of stuttering events we manually identified (oracle).

			HASTE				SSIM baseline				Pixel-sim baseline			
Video	**Length**	**#Oracle**	**#Found**	**TP**	**Recall**	**Prec.**	**#Found**	**TP**	**Recall**	**Prec.**	**#Found**	**TP**	**Recall**	**Prec.**
1	15:15	1	3	1	1.00	0.33	29	1	1.00	0.03	1	1	1.00	1.00
2	8:49	4	8	1	0.25	0.13	29	3	0.75	0.10	7	2	0.50	0.29
3	7:27	1	4	1	1.00	0.25	19	1	1.00	0.05	14	1	1.00	0.07
4	17:21	10	11	7	0.70	0.64	194	5	0.50	0.03	57	3	0.30	0.05
5	11:59	1	2	0	0.00	0.00	1	0	0.00	0.00	1	0	0.00	0.00
6	12:04	1	3	1	1.00	0.33	10	0	0.00	0.00	5	0	0.00	0.00
7	6:14	22	29	18	0.81	0.62	74	17	0.77	0.23	10	5	0.23	0.50
8	21:16	0	0	0	-	-	25	0	-	0.00	0	0	-	-
9	5:10	2	1	1	0.50	1.00	0	0	0.00	-	0	0	0.00	-
10	14:37	0	1	0	-	0.00	7	0	-	0.00	0	0	-	-
	2:00:12	42	62	30	0.71	0.48	388	27	0.64	0.07	95	12	0.29	0.13

and video, we provide the number of stuttering events identified (#Found), the number of true positives (*i.e.,* correctly identified events) and the corresponding recall and precision.

HASTE is able to correctly identify 30 stuttering events (0.71 recall), and nearly half of the reported stuttering events are true positives accordingly to our oracle (0.48 precision). Therefore, even by considering all candidate stuttering events not matching our oracle as "wrong", HASTE works reasonably well, with one out of two reported stuttering being correct. HASTE was not able to identify 12 stuttering events documented in our oracle. This happened mostly due to imprecisions for the Random Forest classifier that wrongly labels some slices as *non-gameplay*, thus excluding them from the stuttering analysis. For example, for the video with ID=3 we miss three out of the three stuttering events, since they fall within slices classified as *non-gameplay*. An example of such a scenario is available in the video hosted at `https://youtu.be/ok9TV-lZyJk?t=56`, with the slice starting at second 57 wrongly classified as *non-gameplay* by HASTE, with the consequent miss of the stuttering documented in our oracle (second 63).

In relative terms, it can be noticed that HASTE performs significantly better than both baselines. The *Pixel-sim baseline*, which simply checks whether consecutive frames are equal at pixel level, achieves the worst results in terms of recall (0.29) accompanied by a low precision as well (13%). This shows that

a trivial approach is largely insufficient for the problem at hand. Several of the wrongly identified stuttering events belong to parts of the videos unrelated to gameplay, such as those showing on screen the game settings which, by their nature, are quite static and tend to feature equal pairs of subsequent frames. Also the *SSIM baseline* tends to recommend false positive stuttering events in these cases, since it does not benefit from the exclusion of *non-gameplay* slices performed by the Random Forest. The *SSIM baseline* has the worst precision (0.07), due to the additional tolerance it has compared to the other baseline. Indeed, while the *Pixel-sim baseline* identifies a stuttering only if two subsequent frames are identical, the *SSIM baseline* inherits the design decision of our approach, with the 0.9999 threshold.

Concerning the recall value (0.64), it may look surprising that *SSIM baseline* achieves a lower recall than HASTE. Indeed, the additional steps behind our approach (in particular the *Video Slicer* and the *Video Slice Classifier*) are mostly aimed at removing false positives, *i.e.,* stuttering events detected when there is no gameplay on screen. The lower recall is completely due to the mask employed in HASTE for cropping the 10% most frequently changing pixels in the slice. The top part of Fig. 4.6 shows how HASTE "sees" two subsequent frames before computing their SSIM, which is instead computed on the entire frames (shown in the bottom of Fig. 4.6) by the *SSIM baseline*. The cropped frames allow HASTE to exclude the face of the player from the similarity computation, thus correctly identifying the stuttering, since the core part of the game scene is identical between the two images. Instead, the baseline considers the changes in "irrelevant" parts of the screen which make it missing the stuttering since the similarity falls below the set threshold.

It is worth noting that on the video with ID=2 both baselines achieved a higher recall as compared to HASTE. This is due to the lack in this video of external elements on screen (*e.g.,* the player), which makes all strategies adopted in HASTE to exclude false positives useless, and just leading to the lost of valid stuttering events.

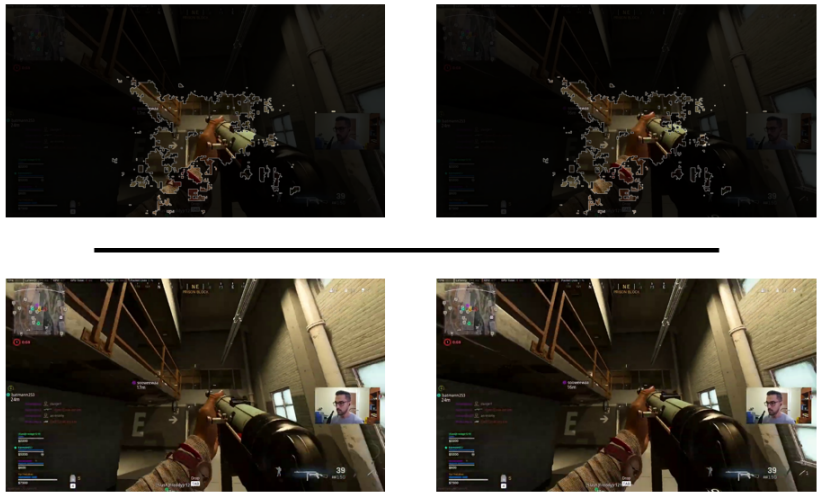


Figure 4.6: Comparison between HASTE and *SSIM baseline*: On top how HASTE "sees" two subsequent frames before computing their SSIM; on the bottom the complete frame how seen by the baseline for the similarity computation

Table 4.4: RQ_3: Precision in identifying micro-stuttering events not in the oracle. † means we analyzed a sample of events.

	HASTE			SSIM baseline			Pixel-sim baseline		
Video	**#Analyzed**	**TP$_{\neg O}$**	**Precision**	**#Analyzed†**	**TP$_{\neg O}$**	**Precision**	**#Analyzed†**	**TP$_{\neg O}$**	**Precision**
1	2	1	0.50	14	0	0.00	0	0	-
2	7	6	0.85	13	4	0.31	5	3	0.60
3	3	2	0.67	9	1	0.11	13	4	0.31
4	4	3	0.75	98	10	0.10	54	4	0.09
5	2	1	0.50	1	0	0.00	1	0	0.00
6	2	1	0.50	5	2	0.40	5	1	0.25
7	11	10	0.91	30	19	0.63	5	1	0.25
8	0	0	-	13	0	0.00	0	0	-
9	0	0	-	0	0	0.00	0	0	-
10	1	1	1.00	3	0	0.00	0	0	-
	32	25	0.78	186	36	0.19 ± 0.05	83	13	0.16

We further analyzed the stuttering events identified by the experimented approaches but not matching our oracle (*i.e.,* correct stuttering we missed while creating the oracle). This could happen especially in the case of micro-stuttering events which are hard to spot for humans, but that might still be relevant for developers [165]. We show the results of such an analysis in Table 4.4, in which we report, for each approach, (i) the number of events analyzed, (ii) the number of true positives found that do not belong to the oracle (TP$_{\neg O}$), and (iii) the computed precision. Remember that for the *SSIM baseline* we analyzed a statistically significant sample (186 instances) of the 388 reported events, while we analyzed all the events for HASTE and the *Pixel-sim baseline*. HASTE confirms its superiority, by achieving a 0.78 precision compared to the 0.19 of the best-performing baseline. Overall, when considering both the true positives falling and not falling within our oracle, 89% of the stuttering events detected by HASTE are actual stuttering or micro-stuttering events. Only 7 out of 62 identified events are false positives.

Table 4.5 reports the results of the comparison between HASTE and HASTE ^{NoFiltering}. HASTE ^{NoFiltering} is able to correctly identify a larger number of valid stuttering events as compared to HASTE (0.73 recall), but it is less precise (0.15 precision). The higher recall can be explained with wrong classifications of segments as non-gameplay, which did not allow HASTE to detect stuttering events

Table 4.5: Comparison between HASTE and HASTE ^{NoFiltering} in terms of recall and precision in the identification of stuttering events we manually identified (oracle).

			HASTE ^{NoFiltering}				**HASTE**			
Video	**Length**	**#Oracle**	**#Found**	**TP**	**Recall**	**Prec.**	**#Found**	**TP**	**Recall**	**Prec.**
1	15:15	1	7	1	1.00	0.14	3	1	1.00	0.33
2	8:49	4	19	**2**	0.50	0.11	8	1	0.25	0.13
3	7:27	1	7	1	1.00	0.14	4	1	1.00	0.25
4	17:21	10	135	7	0.70	0.05	11	7	0.70	0.64
5	11:59	1	2	0	-	-	2	0	0.00	0.00
6	12:04	1	5	1	1.00	0.2	3	1	1.00	0.33
7	6:14	22	32	18	0.81	0.56	29	18	0.81	0.62
8	21:16	0	4	0	-	-	0	0	-	-
9	5:10	2	1	1	0.50	1.00	1	1	0.50	1.00
10	14:37	0	5	0	-	-	1	0	-	0.00
	2:00:12	42	213	31	0.73	0.15	62	30	0.71	0.48

in them. The lower precision, instead, shows that the actual non-gameplay videos that HASTE ^{NoFiltering} contain consecutive identical frames that the last step of HASTE detects as stuttering events. In summary, we can conclude that the Video Slice Classifier is fundamental to significantly reduce the number of false positives.

We further analyzed the stuttering events identified by the experimented approaches but not matching our oracle (*i.e.,*, correct stuttering we missed while creating the oracle). We show the results of such an analysis in Table 4.6, in which we report, for each approach, (i) the number of events analyzed, (ii) the number of true positives that do not belong to the oracle ($TP_{\neg O}$), and (iii) the computed precision.

Such an analysis confirms the superiority of HASTE, which achieves 0.78 precision, compared to 0.15 obtained by HASTE ^{NoFiltering}.

Table 4.6: Comparison between HASTE and HASTE ^{NoFiltering} in terms of precision in identifying micro-stuttering events not in the oracle.s

	HASTE ^{NoFiltering}			HASTE		
Video	**#Analyzed**	**TP$_{\neg O}$**	**Precision**	**#Analyzed**	**TP$_{\neg O}$**	**Precision**
1	6	1	0.17	2	1	0.50
2	17	8	0.47	7	6	0.85
3	6	2	0.33	3	2	0.67
4	128	4	0.03	4	3	0.75
5	2	1	0.50	2	1	0.50
6	4	1	0.25	2	1	0.50
7	14	10	0.71	11	10	0.91
8	4	0	-	0	0	-
9	0	0	-	0	0	-
10	5	1	0.20	1	1	1.00
	182	28	0.15	32	25	0.78

4.5.4 Generalizability of HASTE

To assess the generalizability of HASTE beyond the dataset we used for evaluating it, we run it on a generic gameplay video in which we manually validated the presence of stuttering events even if was not declared in the video metadata. In detail, we started by manually building an oracle reporting stuttering events on a gameplay video. To select the gameplay video, we used Reddit to look for a video game and a specific area notoriously affected by stuttering. Specifically, we selected a video regarding a driving session from the video game Grand Theft Auto V. Such a video was a normal gameplay video for which no stuttering information was reported in the metadata (*e.g.,* title or description). The first author looked at the 18 minutes of the video with the goal of manually identifying stuttering events. As described in the evaluation of RQ_3 stuttering events were documented by noting the second in the video and the corresponding frame at which the event began. We identified 7 stuttering

events. We then ran HASTE on such a video. It reported 8 candidate stuttering events. Based on these, we compute its recall and precision when considering the built oracle as the ground truth. We obtanied 0.86 recall and 0.75 precision. We manually analyzed the reported stuttering events that were outside of the oracle.

4.5.5 Discussion

In this section, we provide practitioners with suggestion on how they could use HASTE in practice.

Using Gameplay Videos from Twitch or YouTube. An interesting point to discuss regards the distinctions between acquiring gameplay videos from the two dominant platforms used for hosting gameplay videos, *i.e.,* YouTube and Twitch. Twitch has become a leader in the live streaming sector, primarily because it was the pioneer in executing the streaming model effectively. This early and successful focus on live streaming helped Twitch carve out a dominant position in this particular niche. On the other hand, YouTube has always been at the forefront of video hosting services, offering a platform where users can upload, share, and watch generic video-on-demand (VOD) content. In our study, we used YouTube videos as the main resource to evaluate our third research question (RQ_3) related to the identification of stuttering events. This choice is based on YouTube's extensive collection of curated shorter videos (between 5 and 20 minutes) showing stuttering events within games. These videos provide a focused lens to analyze and validate stuttering events identified by HASTE. On the other hand, we use videos available on Twitch to evaluate our first research question (RQ_1) related to the evaluation of video slices, where we have the ability to analyze longer videos to identify more potential scene changes within the video that may not occur in shorter videos. The extended duration of streams on Twitch provides a larger dataset for initial analysis and refinement of our approach, ensuring that HASTE can efficiently handle extended gameplay

videos. On Twitch, during live streaming, streamers often pause gameplay to offer commentary or advertising content. This variation of content within a single stream increases the complexity of our analysis, allowing us to evaluate different elements within long-form video content. For this reason, we considered both platforms in our evaluation. It is worth noting that HASTE is not limited to one of them and it can be used on generic videos (from both platforms, or even others). The identification of stuttering events does not depend on the video source on which the analysis is conducted, neither from a technological point-of-view (*e.g.,* reliance on specific APIs) nor from a methodological perspective (since the only information needed is the source video).

Replicating Problems and Debugging. HASTE is mainly intended to provide developers with video bug reports from online platforms. This means that it has two key limitations. First, HASTE does not provide developers with indications on how to replicate the problem observed in the videos. Replicating a bug could be particularly difficult in video games since, in some cases, the input sequence that manifested the failure need to be provided with perfect timing for a successful replication. Besides, even when timing is perfect, the inherent non-deterministic nature of some video games might make it difficult to replicate some problems. There have been some attempts to tackle this problem in the literature [70], but the research in this area is still in its infancy and future work should specifically aim at solving this problem. A second limitation of HASTE is that it does not provide developers with feedback on how to debug and fix those issues. Achieving this goal, however, is beyond the scope of HASTE, which is meant to simply highlight problems, similarly to the several tools daily used by software developers such as linters or test cases.

Graphics Settings: Impact on Stuttering events in Video Games. The presence of stuttering events might be more prevalent when the player chooses settings that provide a better graphic experience since the hardware is more stressed in such a scenario. Nevertheless, stuttering events are *never* something expected from the players, above all during gameplay. Since the hardware

resources are sometimes insufficient to provide the best graphic experience with the highest frame rate available, some video games allow the player to choose between high graphic quality with limited FPS (*e.g.,* 30) or lower graphic quality with higher FPS (*e.g.,* 60 or 120). Still, such a trade-off should never result in the manifestation of stuttering events (indeed, it is provided to avoid them). Stuttering events are clearly more problematic when detected with the game's default graphics settings, which might impact the highest number of players.

Effectively Using HASTE. Through the analysis of gameplay videos HASTE lacks access to hardware information, making it challenging to definitively attribute the identified issues to software, hardware, or configuration-related limitations (*e.g.,* high quality settings on older hardware). Therefore, as any other tool, HASTE might provide false positives. To address this problems, we suggest practitioners to prioritize stuttering events based on the frequency they are reported by HASTE. If the same stuttering event is reported in several videos, it is likely that it is not just an hardware-related or configuration-related problem, but rather a game-related one. On the other hand, isolated stuttering events might indicate false positives (*e.g.,* related to the hardware or the configuration used). Besides, practitioners could use HASTE in combination with other testing tools, as also suggested by the developers we interviewed.

4.6 Industrial Applicability of HASTE

We evaluated the level of interest of game developers in HASTE by conducting semi-structured interviews. In this section, we report the design and the results of such a study. Note that, differently from the survey we preliminarily conducted to verify the relevance of the problem and of our methodology, in this case we aim at receiving feedback on the specific approach we defined (HASTE).

Table 4.7: Interview participants details.

Full Name	Position	Company	Game Development Experience
Lorenzo Valente	Lead Developer	Tiny Bull Studios, Italy	7+ years
Jonathan Simeone	Full Stack Developer	Datasound, Italy	7+ years

Table 4.8: Interview questions.

Question	Type of response
Stuttering Evaluation (repeated for three stuttering events)	
Does the identified stuttering event actually result from a problem within the game?	Yes/No
Please motivate your answers	Open response
Does the provided information about the stuttering event offer enough details to replicate the issue?	Yes/No
Please motivate your answers	Open response
Overall Evaluation	
Would you use this tool before the release of the game?	Yes/No
Please motivate your answers	Open response
Would you use this tool after the release of the game?	Yes/No
Please motivate your answers	Open response
Would you use HASTE in combination with other tools?	Yes/No
How useful is HASTE overall for identifying potential stuttering events?	5-point Likert scale
Please motivate your answers	Open response
Do you have suggestions on how HASTE could be improved?	Open response

4.6.1 Interviews Design

The *goal* of this additional analysis is to assess the practical applicability of HASTE in an industrial context. Specifically, we want to assess whether game developers would consider exploiting HASTE in their testing activities to identify stuttering events in video games. We conducted semi-structured interviews and involved two *game developer* (see Table 4.7) to understand whether they perceive HASTE as a valuable asset that aligns with their needs and objectives in identifying and addressing stuttering events within their game development process. We selected the two participants using convenience sampling (both of them are former students at the University of Molise).

Before each interview, one evaluators explained the objective of the study and described how HASTE works. We showed that the process starts by conducting a search on YouTube for a particular game of interest. In the second step, HASTE can be executed on the given gameplay video to identify stuttering

events in it. As a result, HASTE gives as output a set of times in the video in which potential stuttering events occur.

Both interviews lasted about 30 minutes and were conducted by one evaluators, who recorded and transcribed them for the following analyses. The interviews were based on a reflective strategy. In this way we encourage participants to share their experiences, thoughts and insights in a more introspective way. For example, by asking open-ended and exploratory questions, we encourage participants to reflect on their experiences and provide detailed and nuanced answers.

We preliminarily executed HASTE on one of the gameplay videos we used for our main evaluation of HASTE randomly sampled from the ones we used and identified 3 stuttering events. In detail we use the gameplay video related to the video game *Fortnite*. We asked the participants open the video on YouTube and we told them the time at which HASTE detected each stuttering event (one at a time). We gave them the freedom to navigate the video directly on YouTube to possibly get context regarding the event. After they analyzed each of them, we asked for feedback aimed at understanding whether (i) the identified stuttering event is really an issue with the game and (ii) the information about the stuttering event is sufficient to reproduce the problem (see the top part of Table 4.8).

After the evaluation of the three events, we asked questions aimed at getting feedback on the whole HASTE (see the bottom part of Table 4.8). Specifically, we asked whether they would use this tool to identify stuttering events: (i) in the testing phase before the game release, (ii) in the testing phase after the game release, and (iii) in combination with other tools. Based on the last questions, participants were asked to indicate on a Likert scale from 1 to 5 (the higher the better): the usefulness of HASTE in identifying potential stuttering events. Finally, they were asked to provide possible suggestions on how HASTE could be improved. For all questions, participants were asked to motivate their answers.

4.6.2 Results

Identifying Stuttering Events. Both Lorenzo and Jonathan confirmed that the parts of the video we made them watch (reported by HASTE) contained stuttering events. As for the first question, Lorenzo stated that while the one observed is a stuttering event, it is difficult to determine from the video whether it is a problem due to the game or to the hardware. On the other hand, Jonathan claimed that the observed events are due to a possible rendering problem of the game. Specifically, he points out that, in one of them, stuttering occurs when the player turns the view and a larger game scene with more details is displayed. Similarly, in another case, the player is entering inside a narrower passage: Jonathan states it is very likely that the stuttering event is due to the game since often such parts are used to unload game elements of the previous scene and load the ones that need to be rendered later, thus causing performance issues. In detail he states, "*Regardless of the video card or the hardware, even if it has high power, if the scene is heavy, there is still a drop in fps although not quite as noticeable as it is on a less powerful video card. However, it remains an obvious optimization problem*". In addition, he points out that "*In some games where the graphics are particularly impressive, there are also scenes that, despite all possible optimization, remain heavy as they are rich in detail. Optimizing those scenes would mean having less details and thus risk losing player engagement*".

Reproduction of Stuttering Events. Both Lorenzo and Jonathan reported that HASTE does not provide sufficient information about the identified stuttering events to reproduce the conditions that caused of the issue. Again, Jonathan states "*Based on the observed stuttering events, probably due to a rendering problem, we have some information inherent to what caused the stuttering and thus might allow us to reproduce the conditions that caused the problem. [...] However, there are other aspects not known through the gameplay video that could affect the event*". For example, both point out that some of the reported problems may be hardware-related. The devices on which the video game runs,

or more specifically, the video card used, may have a not negligible impact on the presence of possible stuttering events.

Practical Application of HASTE. In relation to the possibility of using this tool *before the release* of the game, Lorenzo stated that, during beta-testing, he would rather rely on testers: "*I would rely on their experience without going through gameplay videos, gather information through game logs or direct screen recordings*". On the other hand, Jonathan evinces in HASTE a support in beta-testing: "*Maybe the tester can miss some stuttering event because they are thinking about so many other things, and maybe they do not notice the micro lag. So it could be a useful tool to increase the support testing"*. In relation to the use of HASTE *after the release* of the game, both Lorenzo and Jonathan were positive. In particular, they recognized the potential of the enormous amount of information now available through the continuously evolving streaming platforms. Lorenzo states: "*This tool could be used at scale, on a much larger pool of streamers. By analyzing their gameplays, a game developer can get a lot of information. However, the actual causes of the problem remain to be considered*". Again, Jonathan shows strong confidence: "*Absolutely yes, the game continues to be tested by end buyers and I continue to monitor it. This would allow me to find any problems that passed unnoticed in beta testing*". Both participants would use HASTE in combination with other tools. In details, Lorenzo states: "*Yes, I would use it in combination with the tool provided in the engine we use in development. Specifically, I would use the engine tool first and HASTE in the second phase*". Jonathan points out: "*I would use it with any other tool that automatically allows me to identify issues missed in testing. The strength comes based on the fact that gameplay videos are always on the rise because the streamer's profession is now emerging as a real profession. So this tool can be very useful for constant monitoring*".

Usefulness of HASTE tool. In terms of usefulness of HASTE in the identification of stuttering events, Lorenzo gives a rating of 3 out of 5. He states "*I am uncertain still about the same considerations expressed above about possible*

problems due to hardware that could affect stuttering events. [...] However, it might be useful to get this information from streamers". In relation to the possibility of improvement HASTE, he suggests taking information from the logs generated by games during gameplay: "*If HASTE highlights video-side stuttering events and at the same time another tool notices an abnormal sequence of error logs within the log file then there would be more certainty about what caused the stuttering event*". In relation to the same aspect, Jonathan gives a rating of 5 out of 5: "*The proposed events are actually stuttering events. Regardless of what caused them, the tool identified them. Still, more in-depth analysis of these events may be needed*". He also suggests that to improve HASTE it would be interesting to allow developers to give feedback on identified stuttering events in order to recognize false positives. Therefore, a continuous learning model could be useful to improve HASTE through feedback from those who use it.

Summing up. The two participants mostly provided positive feedback on HASTE, but emphasized its limitations. One of them was worried about false positives, while the other one was more enthusiastic about the approach. While we acknowledge that HASTE can provide (even several) false positives, we argue that this is not necessarily an impediment for all developers. The same problem affects most static code analysis tools and still some developers want to used them because they have other advantages (*e.g.,* quick feedback). As the interviews point out, also HASTE has a clear advantage: It is capable of exploiting the very large amount of information available online and reduce the effort of developers interested in analyzing gameplay videos to find and fix performance issues. In addition, the interviews highlight a key point: The usefulness of HASTE strongly depends on its integration into the workflow of game developers. Despite our very limited sample of developers involved, the interviewees reported that they would use HASTE at different stages. Finally, the participants suggested to (i) implement a continuous learning system based on developer feedback to improve the results and (ii) integrate it with information

available from game logs, which however are harder to acquire than gameplay videos and could be more easily available during beta-testing. Finally, the participants highlighted that it may be difficult in some cases to understand whether an observed lag is due to a software (game) issue or to an hardware one. While we agree, game developers may consider an identified stuttering event as relevant only if found in several videos possibly from different streamers, thus increasing the chance that the observed issue is independent from the hardware and is, thus, software-related.

4.7 HASTE-Web: Gameplay Video Analysis

In this section we present “HASTE-Web: Gameplay Video Analysis”, a web application developed from our HASTE approach to help developers in identifying stuttering events in gameplay videos. HASTE-Web leverages this approach to provide an accessible and easy-to-use platform for analyzing performance problems through gameplay videos analysis.

Using HASTE-Web, users can submit links to gameplay videos, which are then processed by the HASTE tool to detect stuttering events. The results of the analysis provide insights into where stuttering events occur in the gamepley videos, allowing users to identify areas that may reveal stuttering events. In addition, HASTE-Web maintains a history of all previously analyzed videos, making it easier for users to review past results and track performance issues over time.

4.7.1 HASTE-Web Architecture

Fig. 4.7 shows the HASTE-Web architecture organized into several key components designed to support the detection and management of stuttering events in gameplay videos. These components work together to analyze videos,

identify stuttering events, and present the results to users. A detailed description of each component and its interactions is provided below.

The **Dashboard** provides the main interface for users, allowing them to interact with the application. Through these component, users can upload gameplay videos, initiate analysis, and view results. The Dashboard interacts with the *StutteringEventDetector* component for identifying stuttering events and to the *StutteringEventsManager* component that allows the user to analyze and evaluate identified stuttering events.

StutteringEventDetector receives the video URL from the Dashboard and is responsible for analyzing the gameplay videos to detect stuttering events. The detector runs the gameplay videos through two subsystems: *VideoSlicer* and *VideoSlicerClassifier*. **VideoSlicer** component takes the gameplay videos as input and slices it into segments. **VideoSlicerClassifier** analyzes each segment produced by the *VideoSlicer* and classifies it as gameplay or non-gameplay. Only segments classified as gameplay are forwarded for further stuttering analysis. The StutteringEventDetector produces a set of annotated gameplay segments, each labeled with details of detected stuttering events. Through the comunication with *StutteringEventsManager* the identified stuttering events are stored for later access and review by users through the Dashboard.

StutteringEventManager receives the classified data from the *StutteringEvent-Detector*, which includes details of detected stuttering events. StutteringEvent-Manager through the interaction with the Dashboard, allowing users to review and confirm or not stuttering events identified in the *VideoSlicerClassifier*. These segments are highlighted for the user, providing a view of areas where the system has detected performance problems. Users can review each highlighted segment through the Dashboard, confirming whether each detected event is indeed a stuttering problem. After the user has confirmed a stuttering event StutteringEventManager sends it to the *StutteringEventsDB*.

StutteringEventsDB component takes in classified stuttering event data from the StutteringEventsManager. It stores all detected stuttering events

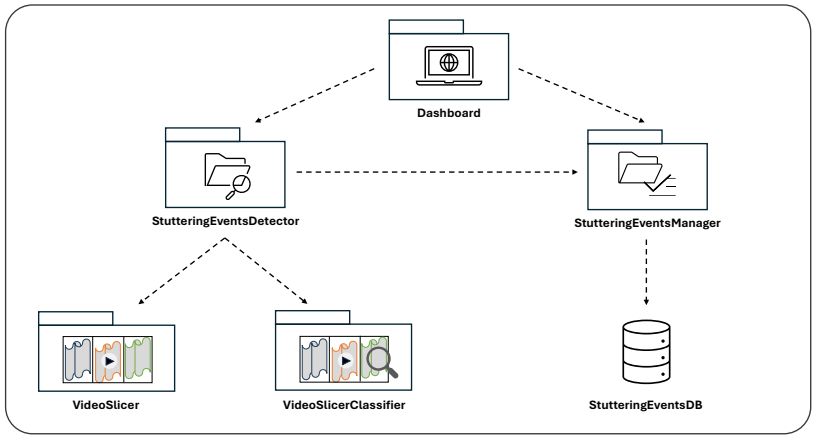


Figure 4.7: HASTE-Web Architecture

allowing users to access an organized history of previous analyses. When a user requests to view previous analyses, the **StutteringEventsDB** provides the necessary data to the Dashboard via the *StutteringEventManager*.

4.7.2 HASTE in Action

Suppose a game developer uses HASTE-Web to detect potential stuttering problems in gameplay videos. After logging into the web-application, the developer selects the *"Stuttering Detector"* function (see Fig. 4.8), where he can insert the *URL* of the gameplay video to be analyzed and start a detailed analysis. The system then starts processing the video by dividing it into segments, identifies the relevant parts of the video containing gameplay, and focuses only on those segments for analysis of stuttering events. During the analysis, the video appears in the *"My Videos"* section, allowing the developer to monitor its progress.

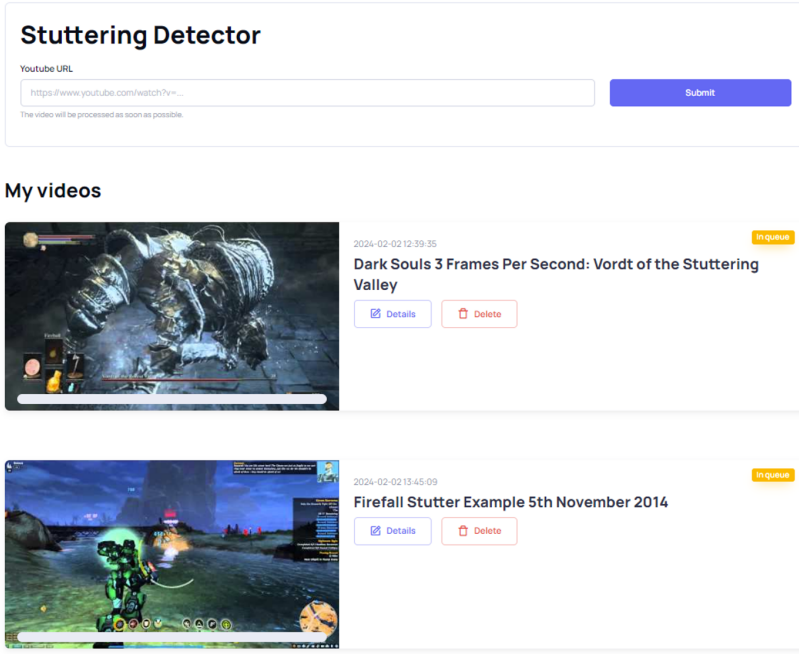


Figure 4.8: Stuttering Detector

The status of the video is displayed as *in queue*, indicating that the analysis is running. Once the analysis is complete, the video appears in the *"My Collections"* section, a personal repository for all analyzed videos (see in Fig. 4.9).

By selecting *"Details"* function for each analyzed video, the developer can view segments where stuttering events were detected. In this section, HASTE-Web shows the developer frames extracted from each segment marked as containing a potential stuttering event. The frames provide a visual context, allowing the developer to evaluate specific parts in gameplay video when performance problems might occur. Each stuttering event is displayed along with a video playback interface, which allows the developer to watch the corresponding segment in real time. This feature helps the developer evaluate whether the

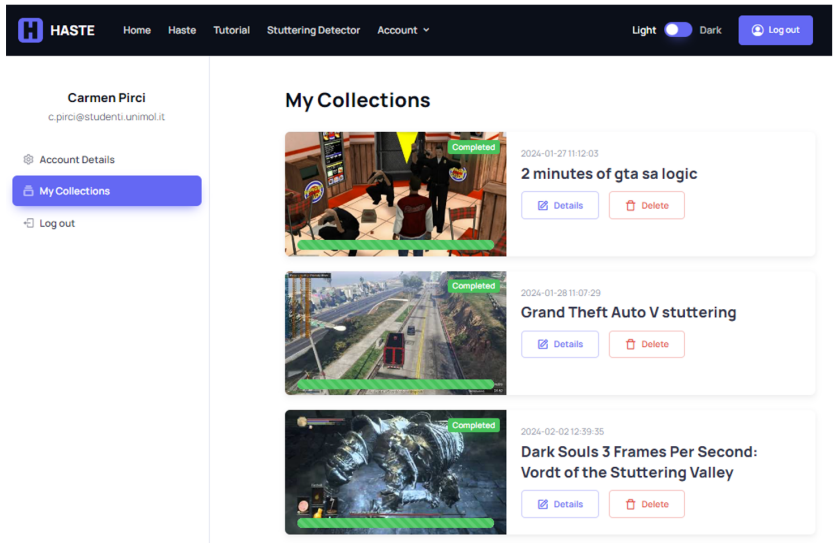


Figure 4.9: Collection Gameplay Videos Analyzed

identified problem shows a stuttering event or not. For each detected event, HASTE-Web offers the option of classifying the segment as “Stuttering” or “Non-Stuttering” (see Fig. 4.10). Once the developer classifies a segment, it is moved to one of the categories—“Classified as Stuttering” or “Classified as Non-Stuttering”—providing an organized structure for review and management of reported events.

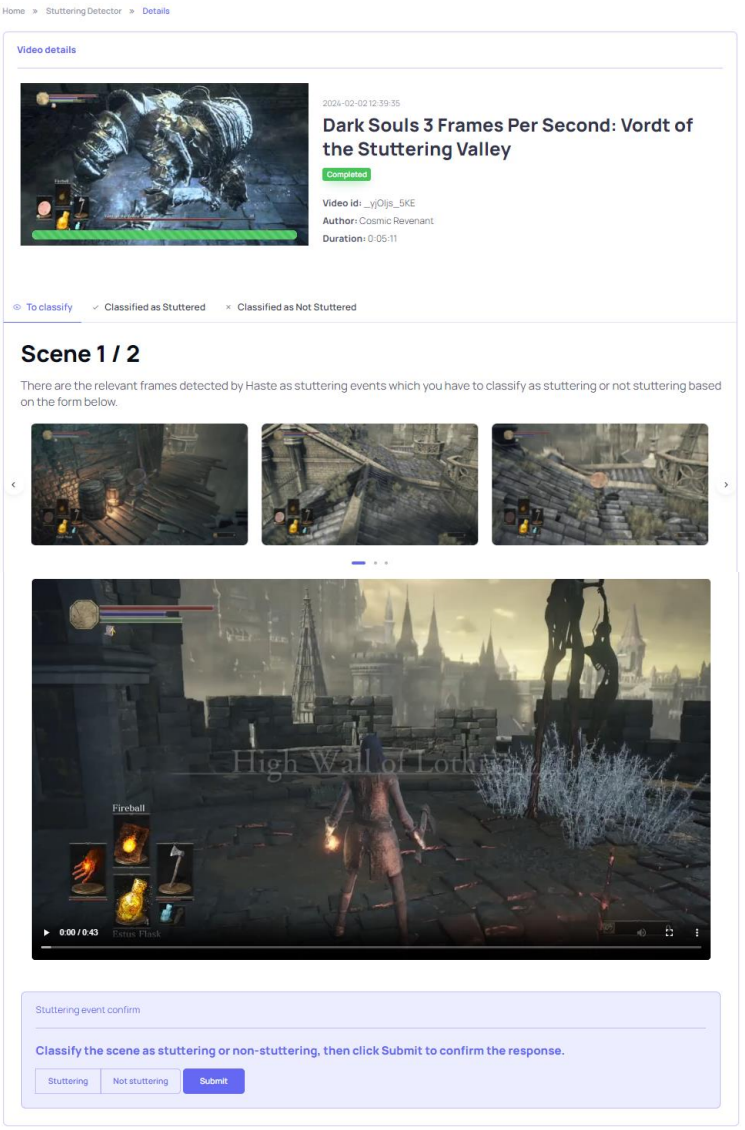


Figure 4.10: Classification of Stuttering Events

4.8 Threats to Validity

Construct validity. Those are mainly related to imprecisions made when building the oracles used in our evaluation and, more in general, when manually inspecting the output of HASTE and of the baselines. The manual analyses always involved at least two evaluators. As for the dataset used to assess the performance of the *Video Slice Classifier*, it has been automatically built starting, however, from manually-classified videos assigned in YouTube to specific categories. Thus, we are confident about the quality of the assigned *gameplay*/*non-gameplay* slices. It is possible that the stuttering events identified in gameplay videos are not due to issues with the game itself but to external factors, such as a heavy computation running on the player's machine or suboptimal hardware configuration. However, gameplay videos are usually recorded by professional players equipped with proper hardware and interested in recording an enjoyable playing session.

Internal validity. We did not tune some of the HASTE's parameters. For instance, we discarded video slices shorter than 5 seconds, since we assumed that any sort of statistical feature we could compute on their frames (*e.g.,* the median of the HM heatmap distribution) would be unreliable. Similarly, we identify candidate stuttering only if at least two subsequent pairs of frames exhibit a SSIM higher than the set threshold (0.9999). The latter is a possible parameter to tune as well. Our decision of avoiding fine-tuning these parameters was driven by the high cost of manually validating different variants of HASTE. However, the lack of parameters fine-tuning does not invalidate our findings, but makes the reported performance a sort of lower-bound for what could be achieved by systematically adjusting the HASTE parameters. To answer RQ_1, we mostly relied on the labeling performed by the evaluators, who might have been biased in the evaluation. To alleviate this threat, we involved two external validators to evaluate the 320 manually-analyzed slices. Such validators were asked to see the labels we provided and classify them either as "correct" or "incorrect." Both

the evaluators reported that all the instances had been correctly validated. This suggest that there was no significant bias, thanks to the methodology we used to label the segments. We involved the same validators to re-assess the labels we assigned to the segments to answer RQ_3, which focused on the identification of stuttering events. Specifically, we asked them to double-check the label we assigned to (i) the 42 stuttering events identified by the evaluators, and (ii) the ones that HASTE or any baseline classified as stuttering event (343 events), totaling 385 evaluations. We used the same methodology adopted for the check of RQ_1. Again, the evaluators reported no disagreement with our evaluations.

We set the threshold for splitting the video in segments in the Video Slicer component of HASTE to 0.3. We chose such a value because we wanted to find clear cut points. We analyzed the distribution of similarities between pairs of consecutive frames in the videos we analyzed to answer RQ_1. Fig. 4.11 reports the results of such an analysis. It is clear that similarity values below 0.8 are always outliers for all the videos. This shows that varying the threshold in this range would result in the inclusion/exclusion of relatively few data points (outliers).

We chose to set the threshold for the slicing at 5 seconds. It might be argued that different thresholds values might allow to obtain better results. We conducted an analysis on the video segments discarded (lower than 5-seconds). In detail, we examined the video segments of each gameplay analyzed in our study. We identified 75 segments shorter than 5 seconds, of which 58 reported a duration of 0 seconds (less than 30 FPS detected). The cumulative duration of the remaining 17 segments was 43 seconds from a total of about two hours of gameplay videos analyzed. Specifically, two segments had a duration of 4 seconds (*e.g.,*, depicting a part of gameplay where the player accesses a game map), while three segments had a duration of 3 seconds, and the remainder mostly lasted only one second. In addition, in 3 out of the 10 videos analyzed, no segments were deleted based on the 5-second threshold. We observed that these short interruptions mostly occur when lighting significantly changes

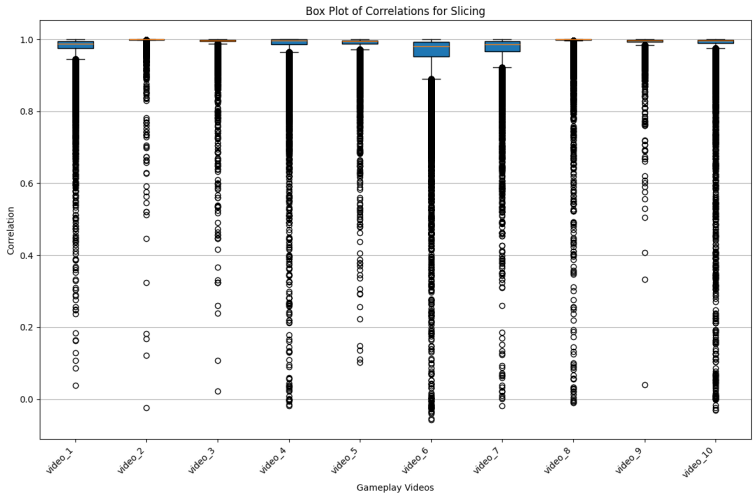


Figure 4.11: Distribution of color histogram similarity values between pairs of consecutive frames.

in the game. For example, these short segments often correspond to bomb explosions, which light up the scene, switches to night mode via infrared, or brief accesses and configurations of game maps. We chose not to combine these short video segments because our goal is to identify scene changes to distinguish between gameplay and non-gameplay segments. Merging segments could increase the risk of classification errors. Therefore, the 5-second threshold was determined as a pragmatic balance between excluding too many short, potentially uninformative segments and maintaining the integrity of significant scene changes within our analysis framework. On the other hand, the impact of the using a slightly larger threshold would have been irrelevant. We analyzed how many segments are in the range between 5 and 10 seconds, and we observed that there are only 13 segments between 5 and 10 seconds.

We chose 3 as a minimum number of consecutive identical frames to trigger the detection of a stuttering event. This, however, could be a sub-optimal choice. To evaluate how many consecutive identical frames are needed to identify stuttering events, we conducted an evaluation with humans to understand to what extent they could detect stuttering in different scenarios. We focused on an approximately one-minute gameplay video. We introduced stuttering events by duplicating frames at three different points in the video, creating three potential stuttering scenarios to assess the visibility and perceptibility of stuttering for users. These scenarios included (i) two consecutive identical frames, (ii) three consecutive identical frames (the validation approach used in HASTE), and (iii) four consecutive identical frames. We asked three external evaluators to watch the videos and mark the seconds in which they perceived stuttering events. The results of this validation process revealed that none of the evaluators identified stuttering with only two duplicate frames. However, in scenarios with three and four consecutive identical frames, the results varied. We report the details of the results in Table 4.9.

These results suggest that duplication of two consecutive frames is probably not sufficient to be perceived as stuttering by viewers, while duplication of three

Table 4.9: Identification of stuttering events with different numbers of identical consecutive frames.

	2 frames	3 frames	4 frames
Evaluator 1	0	3	3
Evaluator 2	0	1	1
Evaluator 3	0	2	2

or more frames leads to consistent detection of stuttering, thus supporting our heuristic choice in HASTE.

External validity. We initially aimed to include in our survey developers from around the world to get a more comprehensive perspective. In the end, we decided not to do this because, in an initial version of the survey, we received several responses from bots. To reduce the manual effort in discarding such responses, we decided to limit the survey and get responses only from the United States and European countries. As a result, however, we might have missed several participants, especially from countries with significant game development industries such as Canada and Japan. To analyze the impact of their exclusion, we simulated a new run of the survey we conducted on Prolific with all the filters we used in our survey, except for the nationality, that we set to Canada and Japan. Prolific warned that fewer than 25 eligible participants would be available, and it did not report the exact number for privacy-related reasons. Thus, we believe that our selection criteria did not significantly alter the results we would have achieved by including such nationalities as well.

In our survey we could only involve 26 participants. It is possible that larger or different samples of practitioners could have resulted in different conclusions. Nevertheless, when analyzing the open answer questions, we noticed recurring themes and a lack of new themes (which might indicate that we reached saturation). For this reason, we believe that, despite the limitations, the sample is sufficient for our purposes. Being the evaluation mostly based on manual

analysis (with the exception of RQ_2), we limited our study to a total of 30 videos (20 for RQ_1 and 10 for RQ_3), excluding the 75 used for RQ_2. Still, this involved multiple evaluators manually analyzing over 10 hours of videos. Two primary concerns revolve around the relatively small sample sizes in both the survey and structured interviews. In the initial survey, only 26 participants were involved. However, it is imperative to underscore that this subset was carefully selected from a broader population, but we focus only on participants connected with the world of video game development. This strategic selection ensured that responses were not only more informative but also aligned with the specific prerequisites outlined for our study. Similarly, the structured interviews were constrained to a mere two participants. This limitation was attributed to the challenges in interfacing with video game developers actively engaged in their professional roles. Despite the small number, it's crucial to acknowledge that these interviews involved two senior developers. This allows us to give robust foundation to the insights gleaned, considering the experience and expertise possessed by the interviewes.

4.9 Final Remarks

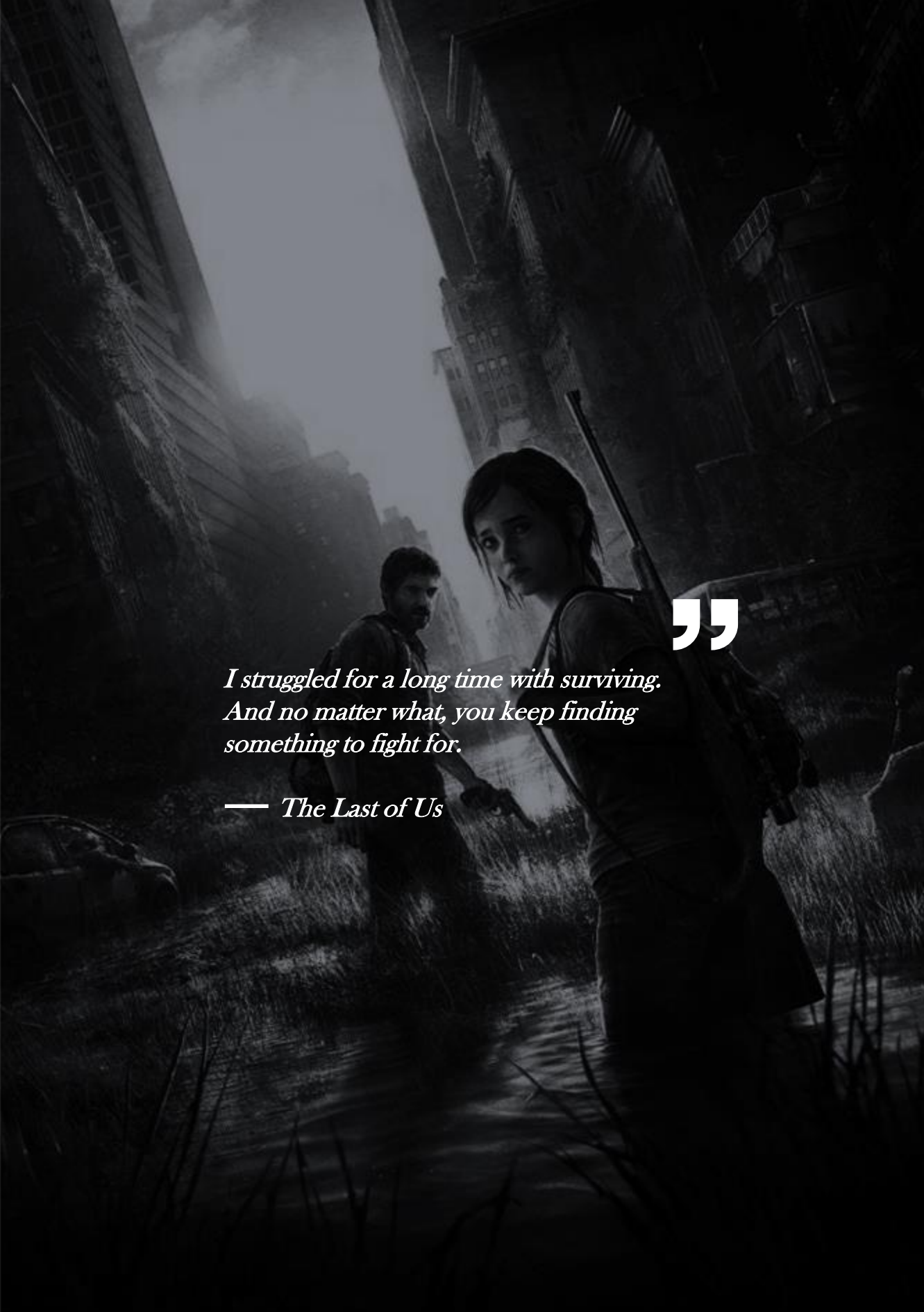
Stuttering is a relevant problem in video game development since it can significantly impact the gaming experience and, thus, lead to poor perceived quality of the product. The identification of stuttering events is, however, quite challenging. Indeed, the "search space" in which they could manifest is huge in modern video games.

We presented HASTE, an approach that allows video game developers to automatically detect stuttering events documented in gameplay videos from Twitch and YouTube so that they can try to reproduce and fix them. We validated the three main steps of HASTE on a total of 105 videos. As for the two preliminary steps, our results show that HASTE (i) is able to accurately split videos in

visually coherent slices, and (ii) is able to distinguish slices containing gameplay from the ones with other contents (*e.g.,* ads). When looking at HASTE as a whole, we found that it is able to achieve significantly better results than the baselines, with an overall 71% recall and 89% precision.

The results of two interviews we conducted with expert video game developers to assess the applicability of HASTE in an industrial context highlight its strengths in identifying potential stuttering events, leveraging the vast amount of data available online. On the other hand, such interviews also highlight the limitations of HASTE: First, it might not provide enough information to reproduce the issue, and second, it might report false positives (*i.e.,* stuttering events not due to the game but to other incidental problems). Both participants provide valuable insights on addressing current limitations.

All code and data used in our study is publicly available in our replication package [69].



I struggled for a long time with surviving. And no matter what, you keep finding something to fight for.

— *The Last of Us*

CHAPTER 5

Detecting Low Engagement Events

Contents

5.1 Introduction

As compared to other types of software, video games have a peculiar non-functional requirement: They need to entertain the users and be fun to play. In other words, they must be engaging. Politowski *et al.* [121] report the "lack of fun" is indeed one of the most common issues leading to the failure of video game projects.

Assessing the extent to which a game is engaging requires playtesting, a specific kind of end-to-end testing [53] in which players are asked to play with different parts (or even variants) of a game and provide their feedback about several aspects, including the game engagement. While other quality attributes of the game can be objectively and (in some cases) automatically assessed (*e.g.,* Frames Per Second as a proxy for performance), engagement is inherently subjective and hard to assess without asking the opinion of players. The high subjectivity implies that what is observed during playtesting in terms of user engagement might not generalize to the greater public. Indeed, playtesting is difficult to scale up in terms of involved players. The difficulties experienced by practitioners in assessing the level of entertainment of their products have been also documented in the literature [134].

Developers could rely on such a goldmine of information from streaming platforms such as YouTube [2] or Twitch [1] to find areas of a video game that are not engaging enough for most of the player, *i.e.,* that are affected from *objective* issues in terms of entertainment. Also, continuously mining the players' engagement in gameplay videos would allow developers to monitor how the engagement is impacted by the game updates introduced after the release or how to plan future releases. The facial expression in the recorded face of the streamers can be exploited to achieve this goal. Indeed, previous work showed that facial expressions can be leveraged to assess engagement [166] and/or emotions triggered by advertisements or movies [9]. Recent studies have shown that these solutions can effectively evaluate engagement in video

games [34, 85]. However, the latter are different from other types of media because they are interactive. Therefore, specific facial expressions related to engagement could be radically different [142, 166].

While such specialized approaches show promising results, their capabilities have only been assessed on *perceived* engagement: Players' facial expressions have been labeled by external evaluators, who defined the ground truth (*i.e.,* "engaged" or "not engaged") on which the approaches have been evaluated. In other words, the such ground truth is based on a subjective interpretation of the video content rather than on the direct feedback from the players themselves. It is still unclear to what extent such approaches are able to capture the *real* engagement of the players, or whether this is possible in the first place. Also, it is unclear whether such approaches would be used by practitioners and how.

In this chapter, we aim to fill this gap. First, we ran an experiment with players aimed a collecting a new dataset that provides the *real* level of engagement experienced by the players during gameplay sessions. We asked 40 participants to play eight video games while we filmed their facial expressions and simultaneously recorded their gaming sessions. We paused the games at regular intervals of a minute and asked them to report their level of engagement. In total, we collected 1,130 labeled pairs of 1-minute videos of facial expressions and engagement labels. Based on such a dataset containing *real* engagement data, we compare two state-of-the-art models (Affectiva [8] and K [85]) and an additional ML-based approach that relies on state-of-the-art features (we conveniently call it FFBD– Facial Features-Based Detection). We show that the best model (*i.e.,* FFBD) achieves a good level of precision in identifying low-engagement events (74.7%). We also test whether using the best model would allow practitioners to find out which games suffer more of low-engagement issues. To this aim, we ranked the eight video games used in our study in terms of number of low-engagement events both by using (i) the predictions performed by the approach, and (ii) the actual low-engagement events declared by the players. The results show that the two rankings are

very similar (Spearman ρ = 0.833). As a second contribution, we explored the feasibility of applying the proposed approach in an industrial context. We conducted semi-structured interviews with two senior video game developers. Overall, they mostly provided positive feedback: They both acknowledged the utility of an approach for detecting engagement and its applicability in the game developement workflow. However, they highlighted that such approaches might not identify low engagement events if, during the gameplay the streamer is influenced by external factors (*e.g.,* chat interaction). Despite this limitation, developers noted that engagement estimation approaches could still offer valuable guidance, helping practitioners identify areas of concern, before and after the game release.

Our findings have important implications for researchers and practitioners. Researchers can rely on our dataset to define new approaches for engagement estimation in video games. Practitioners could integrate such approaches both (i) in playtesting activities to detect low-engagement levels at a finer grain, and (ii) in their after-release monitoring of the entertainment of their product, to timely detect when their products needs updates. We release a prototype tool implementing our approach which can be directly used on online gameplay videos to find low engagement events.

5.2 Predicting Player Engagement

Engagement is a complex phenomenon. There is no agreement in previous research on the precise definition of engagement as a measurable psychological state, and many models for characterizing it have been proposed [52]. Engagement is often linked to the concepts of *immersion* and *involvement* in the context of video games [27, 54, 7]. In our work, we adopt the following definition of engagement: *Engagement is the degree to which the player is involved in the video game and, thus, the extent to which they are willing to continue playing.*

While playing video games, we assume that engagement depends on (i) the players' attitude and preferences in terms of the video game at hand, (ii) incidental factors, such as the tiredness or the willingness of playing at a given moment, and (iii) the engaging nature and quality of the *gameplay* of the video game. While the first two factors are out of the control of video game developers, the latter is what constitutes a non-functional requirement of every video game. The *gameplay* can be defined as the set of "*all actions performed by the player, influencing negatively or positively the outcome of the uncertain game situation in which they are engaged in*" [68]. More specifically, gameplay is usually implemented in a *gameplay loop*, *i.e.,* a cyclic sequence of phases that ultimately rewards the player and aims at increasing their willingness of playing (thus, engagement). A simplified example of *gameplay loop* for an RPG game is the following: The player buys weapons and armors to fight enemies, then they engage in fights and, finally, they are rewarded with more items or gold through which they can buy better weapons and armors, and the cycle restarts.

Engagement in a video game can drop when this cycle breaks. This happens, for example, when the game is too hard and, thus, the reward is never achieved (in the example, the enemies are too hard to defeat). Nevertheless, it is not true that the lower the difficulty the higher the engagement either: If the reward can be obtained without a challenge (in the example, the enemies are too easy to defeat) the player might experience boredom and thus being not engaged. As described by Ermi *et al.* [54], engagement can result in "*challenge-based immersion*", in relation to the mental skills "*such as strategic thinking or logical problem solving*". In short, the game should be aimed at being balanced [62]. Finding the "sweet spot" for obtaining engagement is a matter *game design* and *level design*.

Previous work defined several approaches for automatically measure engagement in video games [34, 85]. While each approach has its own peculiarities (we report them below), they are all based on the same premise.

A player plays a video game and he/she is engaged or not engaged while doing so. As a result, the players feel emotions (*e.g.,* happiness) and reacts accordingly with specific facial expressions (*e.g.,* raises the eyebrows or moves the head in certain ways). Some features are extracted from the recorded facial expression and adopted to train a model to predict engagement. Since the training of such models is supervised, it is required to annotate the facial expressions with the ground truth. There are two ways in which such models are evaluated and, thus, the ground-truth is defined, *i.e.,* through *perceived* engagement and *real* engagement.

In the former, the models are evaluated in assessing engagement *as a human would* [34]. External evaluators (different from the players) observe the facial expression of players and manually assess whether they are engaged or not. This annotation constitutes the ground truth. This type of evaluation allows to understand to what extent the models are as good as humans in assessing the engagement. Achieving the maximum accuracy in this type of evaluation means that the model exactly mimics what a human would do.

In reality, however, human might fail at assessing the engagement of other humans by only relying on their facial expression. For example, a person with furrowed eyebrows and a neutral expression might can be interpreted as symptoms of engagement, while the same expression might indicate tiredness, anger, and thus low engagement. Consider, for example, the gameplay video at `https://youtu.be/iAdcCjrL_6M?t=347` (one of the frames is also reported in Fig. 5.1). The streamer laughs, which could be interpreted as "fun." However, he is actually frustrated as he is blocked in a game mechanic that is not working, as it becomes clear in the continuation of the video. A human who only watches that specific segment would think that the player is engaged.

Thus, it appears clear that evaluating engagement prediction approaches on the *perceived* engagement might provide an over-estimation of their actual capabilities.

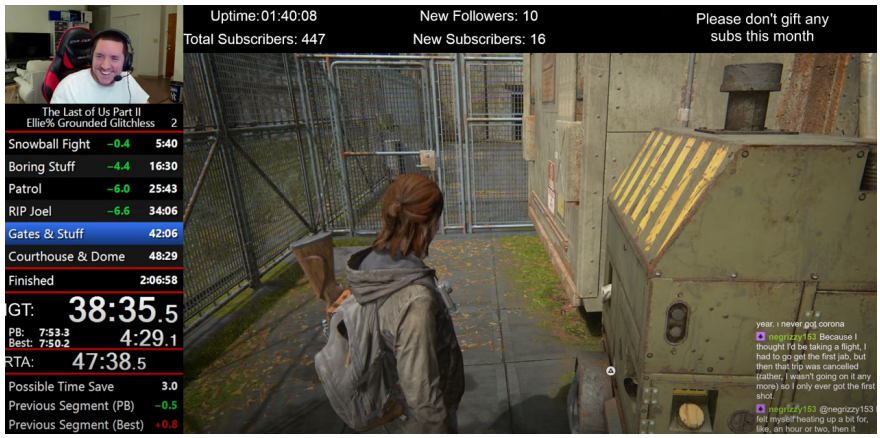


Figure 5.1: Example of low-engagement

Using the *real* engagement as the ground-truth, however, requires a direct feedback by the players, which should be collected in a timely manner (*e.g.,* right after the gaming session).

5.3 A Dataset of Real Engagement

In this section, we describe how we defined a dataset that contains evaluations of the *real* engagement of the players. Specifically, to do this, we conducted a human study with players.

Context. The experimental context consists of *subjects* and *objects*. The subjects are 40 video game players, *i.e.,* people that regularly play video game and that played at least once any video game in the last year. To recruit them, we used convenience sampling. More specifically, we involved (i) students at the University of Molise, (ii) personal contacts of the authors, (iii) other people we reached out by locally publicizing such an activity.

Table 5.1: Video Games used for the empirical evaluation and their characteristics.

Video Game	**Description**
Space Invaders	Destroy the space invaders by firing your laser
Qbert	Change the colour of all pyramid cubes
Amidar	Visit all places on the grid while avoiding enemies
Gopher	Protect a crop of three carrots from a gopher.
Rayman	Platforming adventure
Snake	A modified version of Snake
Lonely	Reckless descent through pristine mountains
Golf Assassin	White pixel minigolf platformer

The objects are 8 video games reported in Table 5.1. We selected them aiming at covering different genres. Specifically, four of them were ATARI games (Space Invaders, Qbert, Amidar, Gopher), *i.e.,* classic 2D games, while four were free-to-play games available on the Steam Platform (Rayman, Snake, Golf Assassin, and Lonely). As for the ATARI games, we used the versions available in the Gym Python library [109]. Gym offers a set of Atari 2600 environment simulated through the Arcade Learning Environment. In our study we use the default environment of each game. As for the games taken from the Steam Platform, we simply downloaded and installed them.

Protocol. Before starting the experiment, we asked all participants to sign an informed consent form which explains the purpose of the study and the treatment of the data acquired during the experiment. We also carefully explained the study protocol and what we would have asked them to report (*i.e.,* their level of involvement, as we explain later).

The experiment consisted in several subsequent *gaming sessions*. In each gaming session, the participants were asked to play a game and report their engagement at different times. More specifically, each gaming session started with a description of the game and the commands to be used. When the participant started the game session, we started acquiring their face through a webcam.

Then we let the player play for 3 minutes. After each minute of gameplay, the game was paused and a small window was shown, asking the participants' level of involvement ("What is your level of involvement?") on a Likert scale from 1 (very low) to 5 (very high). We use this as an operationalization of the psychological state of engagement of the player, as previously done in the literature [168].

We divided the study in two parts: a mandatory first part and an optional second part. In the first part, we ran eight gaming session, one for each of the eight games in Table 5.1. Then, the participants were asked whether they wanted to continue playing. If the answer was yes, the second part started, in which we ran additional gaming sessions (one at a time) with randomly chosen games and then asked whether they wanted to continue playing. The second part continued until the answer to the last question was negative. We did this to make sure that the player was not annoyed by the experiment (which could reflect in their facial expression and affect their engagement). In total, we collected approximately 19 hours of recordings.

Each participant was involved for $\sim$30 minutes, on average. The study was conducted in a controlled setting. Specifically, it has been executed in a laboratory with the same equipment in order to have consistent data acquisition across participants. One of the authors prepared the environment and supervised all the participants with the aim of monitoring the correct execution and intervening if they needed clarifications. To avoid biases due to the interactions among participants, we involved one participant at a time and we made sure that no other person was present during the experiment, except for the supervisor (one of the authors) who, however, did not interact with participants and was out of their sight. This allowed us to check that the change in the participant's emotions is due to the game played at that specific time alone.

The order in which the first 8 games were played was the same for each participant, while, as specified above, the execution of the other games is random. To guide the execution of the experiment and to automatically measure

the variables we were interested in, we implemented a script that automated the execution of the whole games, the acquisition of the participants' face, and their level of involvement.

Before running the experiment, we ran a small pilot study with four additional participants (not involved in the main study), in order to test the framework and the protocol and to spot any possible problem before starting the study.

Dataset Characteristics. After collecting all the raw recordings of the participants' faces during the game sessions, we extracted the frames from each recording (10 frames per second), totaling 600 frames for each 1-minute gaming session. We labeled each of such sequences in a binary way: we used the *low engagement* class when participants reported 1 or 2 as level of involvement, and the *non-low engagement* class for the other evaluations (3 to 5). In the end, we collected a total of 1,130 data points, each one characterized by the frames extracted from the face recording associated with a binary label concerning the engagement (low engagement or non-low engagement).

5.4 Study I: Predicting Real Engagement

The *goal* of our study is to understand if it is possible to use real level of engagement to detect poorly engaging parts in video games, both for individual players and globally.

In particular, we aim to answer the following research questions (RQs):

> RQ_1: *To what extent is it possible to detect the real engagement in a gaming session with state-of-the-art approaches?*

With this first research question, we want to evaluate whether is possibile detect the real level of player engagement based on analysis of the video recording of a given game session with state-of-the-art approaches.

RQ_2: *To what extent can state-of-the-art approaches rank the video game parts similarly to how humans would in terms of real engagement?*

With this second research question, we want to understand whether using the real level of engagement, it is possible for practitioners to spot video games sections in which a high number of players experienced low engagement.

To answer both our RQs, we rely on the dataset collected in the human study presented in Section 5.3.

5.4.1 Engagement Detection Approaches

We compare three engagement detection approaches: *Affectiva*, the approach by Killedar *et al.* [85], and an additional ML-based approach based on state-of-the-art features. We could not include FaceEngage [34] in our experiment because the tool implementing the approach is not publicly available. We describe such approaches below.

Affectiva

Affectiva is a leader in media analytics, in the field of emotion analysis. Their commercial solution relies on a large emotion database, consisting of data from 10 million consumer responses to over 53,000 advertisements in 90 countries. Affectiva measures the level of engagement through the weighted sum of a set of action units computed by looking at the facial muscles activation. In particular, it provides an engagement score between 0 and 100. The engagement score is calculated as the weighted sum of the following facial expressions: Inner and outer brow raise, Brow furrow, Cheek raise, Nose wrinkle, Lip corner depressor, Chin raise, Lip press, Mouth open, Lip suck, Smile.

We used the free version of Affectiva available through its JavaScript APIs [35] in the IMotion platform [79]. Affectiva computes and reports an engagement score for a single frame, with values ranging between 0 and 100. For each game session recording, we aggregated such frame-level scores by computing the average engagement.

K+

Killedar *et al.* [85] introduced an approach aimed at assessing players' engagement via facial expressions. Such an approach focuses on facial emotions rather than expressions, meaning that the only features used to decide if the player is lowly engaged are seven emotions: neutral, sad, angry, happy, surprise, disgusted, and fearful. The authors process the detected emotions and use fuzzy logic to obtain an overall engagement score. This score is derived by evaluating the intensity of facial emotions and combining them to produce an engagement index, which reflects the experience of the player. We could not exactly replicate the approach by Killedar *et al.* since (i) no replication package is available, and (ii) at several points, the description of the approach is ambiguous. However, one rule is very clear: When the player prevalently shows neutrality, the engagement is very low. Thus, we decided to implement an optimistic version of the approach by Killedar *et al.* (K+) based only on this rule. In our implementation, if the precondition of such rule is met (*i.e.,* if the prevalent emotion detected in the majority of frames of the recording is "neutral"), we assigned the label *low engagement*; otherwise, we assumed that the remainder of the approach that we did not replicate is always correct, *i.e.,* we use as predicted label the actual label. Note that, in our implementation, we use the emotion detection provided by Py-Feat instead of the custom one used in the paper since the training dataset was not available and, again, we could not replicate that part of the approach. The set of emotions detected by both techniques is the same.

Table 5.2: Biometric Data Acquired and their Relationship with Engagement.

Name	**Description**	**Relationship with Engagement [10, 36]**
Pitch	Rotation around the side-to-side axis	↓ Frustration or disapproval
Roll	Rotation around the front-to-back axis	↑ Interest in something
Yaw	Rotation around the vertical axis	↓ Distraction
AU01	Inner Brow Raiser	↑ Involvement with the content
AU02	Outer Brow Raiser	↑ Involvement with the content
AU04	Brow Lowerer	↑ Involvement with the content
AU05	Upper Lid Raiser	↓ Frustration
AU06	Cheek Raiser	↑ Involvement with the content
AU07	Lid Tightener	↓ Frustration
AU09	Nose Wrinkler	↑ Involvement with the content
AU10	Upper Lip Raiser	↓ Frustration
AU11	Nasolabial Deepener	↑ Involvement with the content
AU12	Lip Corner Puller	↑ Involvement with the content
AU14	Dimpler	↓ Lack of Involvement
AU15	Lip Corner Depressor	↑ Involvement with the content
AU17	Chin Raiser	↑ Involvement with the content
AU20	Lip Stretcher	↑ Involvement with the content
AU23	Lip Tightener	↓ Frustration
AU24	Lip Pressor	↓ Frustration
AU25	Lip Part	↑ Involvement with the content
AU26	Jaw Drop	↑ Involvement with the content
AU28	Lip Suck	↑ Involvement with the content
AU43	Eyes Closed	↑ Involvement with the content
Anger	Intense emotion of displeasure, frustration, and hostility	↓ Frustration
Disgust	Revulsion and aversion towards something unpleasant or offensive	↑ Involvement with the content
Fear	Perception of possible imminent danger or threat.	↑ Involvement with the content
Happiness	Positive and joyful emotion characterized by contentment and satisfaction.	↑ Involvement with the content
Sadness	Deep and intense emotion of sorrow and unhappiness.	↑ Involvement with the content
Surprise	Sudden and unexpected emotional reaction.	↑ Involvement with the content
Neutrality	Absence of clear emotional response or indifference towards a situation.	↓ Lack of involvement

FFBD

We considered an additional approach — we conveniently call it FFBD, *i.e.,* Facial Features-Based Detection. FFBD is based on a complete set of state-of-the-art facial features and relies on a classic ML algorithm (*i.e.,* Random Forest, in detail we used the implementation with the default configuration provided in Weka [163], which requires to be trained to classify a given instance in a binary way (*low* or *non-low* engagement). We describe in Section 5.4.2 how we trained the model for our experiment. FFBD relies on three categories of features: emotion, expression, and behavior features. In Table 5.2, we describe such categories and explain how they might relate to engagement based on empirical evidence from previous work.

Emotion Features. Positive emotions, such as joy, excitement, or satisfaction, are related with positive engagement in the game [10, 36]. Such emotions can be triggered, for example, by success in the game, achieving goals, overcoming challenges. Whereas, negative emotions, such as frustration, anger, or boredom, may indicate a negative level of engagement. We use Py-Feat [36], a publicly available tool that leverages consolidated approaches for emotion recognition through facial expression analysis to determine emotions and other characteristics from static images that include the face of a human (in our case, the player) [37]. More specifically, given an image, the tool reports a probability that the person is experiencing the emotion reported in Table 5.2. To compute video-level features starting from the emotions measured on each frame composing the video, we calculate for each of them, the first, second, and third quartiles (Q1, Q2, and Q3), and their standard deviation (SD). This means that for a video including 600 frames, we will have 600 values for each emotion (*i.e.,* a probability that the user is experiencing that emotion) that represent a distribution on which the above-listed statistics are computed. In addition, for each emotion e we compute its total duration (TD_e) as the percentage of frames in the video for which it is the prevalent emotion (*i.e.,* the one with the

highest probability). For example, if there are five frames, and in three of them the player has a prevalent *happy* emotion, the $TD_{happiness}$ feature is 60%. We also compute the longest sequence of frames in which an emotion is prevalent (LD_e). To do this, we first assign the prevalent emotion to each frame. Then, for each e, we compute the longest sequence of consecutive frames marked with e and calculate its percentage with respect to the total duration of the video. For example, if there are, in total, ten frames, and there are at most 3 consecutive frames in which the player is *happy*, the $LD_{happiness}$ is 30%. Finally, we compute features related to the emotions of the player shown at the very last frame of the recording. The assumption is that such emotions shown by the player just before the time in which we are interested in capturing the engagement might be important in assessing it. We consider both the probabilities reported by Py-Feat to each emotion (ranging from 0 to 1) and the binary value (0 or 1), where 1 is assigned to the prevalent emotion in the last frame, while 0 to the other ones. Similar measures have be used in previous work [93].

Expression Features. While emotions help determine the level of engagement of a player, a finer-grained level of detail (*i.e.,* the raw action units) might help detecting expressions possibly related to low engagement that are not related to specific emotions. Again, we use Py-Feat [36, 37] to compute the probability that the player is performing all the action units we consider (see Table 5.2). As we do for the emotion features, we aggregate each expression feature previously listed (*i.e.,* the probabilities reported by Py-Feat for each action unit) by computing the first, second, and third quartiles, and their standard deviation, for all frames in the video. Also in this case, we add features related to the last frame (one for each action unit). Such features have as values the probabilities that the player performed the related action unit in the last frame.

Behavior Features. The behavior of a video game player can be used to assess their engagement by determining the player's level of immersion in the game. For example, the physical proximity of the player to the screen can indicate a high level of engagement. Since we assume we only have information

about the face, we only focus on the behavior exhibited through the head. Specifically, we consider as features the pitch, roll and yaw of the head.

We use Py-Feat to compute the three previously-mentioned features for each frame. Specifically, the features range between $\pm$90 degrees (roll), $\pm$75 degrees (yaw) and $\pm$60 degrees (pitch) [37]. As we do for both the previously-reported categories of features, we aggregate each behavior feature by computing the first, second, and third quartiles, and their standard deviation, of the measurements performed for all frames of the video. We also use the values of Pitch, Roll, and Yaw observed in the very last frame, as done for the other categories of features.

5.4.2 Experimental Design

To answer RQ_1, we compare the three previously-described approaches on our dataset containing *real* engagement evaluations. Both Affectiva and K+ do not require training. So, we simply use them to predict the engagement of all the data points in our dataset. Note that Affectiva returns a continuous score rather than a binary classification. To classify the engagement level as *low* or *non-low*, we used two thresholds. The first one is $k = \frac{2}{5} \times 100 = 40$. We did this because, as explained, we discretized engagement as *low* when the engagement level self-reported by the users is lower than $\frac{2}{5}$. The second threshold is the one that allows to achieve the best F1 score. We report the results achieved by using threshold levels between 10 and 100 at steps of 10 to have a complete view of the performance of Affectiva. As for FFBD, instead, we use a 10-fold cross validation to evaluate the model, which consists in dividing the dataset in ten equally-sized folds and using 9 of them for training (1,017 instances) and one for testing (113 instances). Note that, as a result, we obtain the predicted engagement for all the instances, thus making the results comparable to the ones obtained with the other approaches. We noticed that Affectiva fails to compute the engagement scores on some frames (*e.g.,* frames with a light

rotation of the face, whereas with PyFeat we are able to do this). To perform a fair comparison, we re-trained/tested FFBD also on a subset of instances from which excluded such frames.

We computed and report the recall, precision, and F1-score (the harmonic mean of precision and recall) for both the classes (*i.e., low engagement* and *non-low engagement*). We also report the AUC (Area Under the ROC curve [24]). An AUC $\simeq$ 0.5 indicates a model having the same prediction accuracy of a random classifier. A perfect model (*i.e.,* zero false positives and zero false negatives) has AUC = 1.0.

To answer RQ_2, we rank the game areas considered in our study in two different ways. First, we do that based on the best prediction model resulting from RQ_1 and, specifically, on the total number of predicted *low-engagement* events. Second, we rank the areas based on the number of self-reported *low-engagement* events. Ideally, a perfect alignment of the ranking means that the predicted engagement could be used instead of the *real* one to detect areas that are lowly engaging according to several players without explicitly asking them. Note that since in our experimental design each user plays each game for at most 3 minutes and they play at most a level of each game, we can safely say that all the recordings correspond to a single game section.

To compare the two rankings, we computed the Spearman's rank correlation coefficient (ρ) [108], which computes the statistical dependence between the rankings of two variables. A high correlation coefficient indicates that the two rankings are very similar. We also reported the p-value of the correlation, which indicates the probability that the correlation is different from 0.

Our replication package [72] includes the implementation of the script we used to run the experiment, the dataset (not including the original recordings for privacy issues, but the extracted features), and the scripts for analyzing data.

5.4.3 Results

RQ_1**: Individual Level of Engagement**. Table 5.3 reports both the confusion matrix of our Random Forest model and the precision, recall, and F1 score metrics we computed, based on our 10-fold cross validation. As it can be noticed, the precision is quite high for both the classes ($\sim$0.75). On the other hand, FFBD confuses several low engagement events for non-low engagement events (the recall is 0.41 for low engagement). This is probably due to the unbalance between the two classes: 35% of the instances are low engagement, while the other 65% are non-low engagement.

The Random Forest classifier reports, for each instance, a "confidence level", a percentage value that indicates how confident the model is in the provided classification. Such a score is computed as the percentage of classification trees in the forest that "voted" for a specific output (*low engagement*, in our context). The results previously shown are obtained using the default threshold (0.5) for deciding which instance is predicted as *low engagement* and which one as *non-low engagement*. We tested different thresholds to understand its impact on the final result. We assume that practitioners would be more interested in having a *precise* tool that reports only the very likely low engagement events. We report in Table 5.4 precision, recall, and F1 score for the thresholds ranging from 0.5 to 1.0, with a step of 0.05. As it can be noticed, a threshold $k = 0.6$ would allow to achieve a very high precision (0.84, +0.10 percentage points), at the cost 0.2 percentage points of recall. Higher threshold values would have a cost in terms of recall without any relevant benefits in terms of precision, which anyway is at most 0.89 (excluding $k = 0.8$, for which the precision is 1.0, but the recall is close to 0).

Table 5.5 reports the results obtained using the engagement score provided by Affectiva. The best results in terms of F1-score can be achieved with $k = 90$. Thus, we use Affectiva $_{k=90}$ and Affectiva $_{k=40}$ (for the previously-explained reasons) as representatives of a binary classifier based on Affectiva.

Table 5.3: RQ_1: Confusion matrix and evaluation metrics of FFBD.

		Predicted: Low Eng.	Predicted: Non-low Eng.	Precision	Recall	F1
True	**Low Eng.**	159	231	0.74	0.41	0.53
True	**Non-Low Eng.**	55	685	0.75	0.93	0.83
			Weighted Average	0.75	0.75	0.75

Table 5.4: RQ_1: Precision, recall, and F1-score for the *low engagement* class using different threshold values.

k	**Precision**	**Recall**	**F1 score**
0.5	**0.74**	**0.41**	**0.53**
0.55	0.77	0.27	0.41
0.60	0.84	0.20	0.32
0.65	0.89	0.11	0.19
0.70	0.89	0.05	0.10
0.75	0.78	0.02	0.04
0.80	1.0	0.01	0.02

We show in Table 5.6 the comparison among the three approaches. The top part of the table reports the results achieved on the whole dataset, while the bottom part provides the results achieved on the subset of frames that can be treated by Affectiva. The effectiveness of FFBD is consistently higher than the other two approaches. However, such a approach confuses several low engagement events for non-low engagement events (the recall is only 0.41 for the *low engagement* class). This is probably due to the unbalance between the two classes: 35% of the instances are low engagement, while the other 65% are non-low engagement. The AUC of FFBD is 0.79, which shows that the state-of-the-art features have a high capability of distinguishing the two classes.

Both the Affectiva binary classification versions we considered have a low power of distinguishing the two classes. This is clear when looking at the AUC, which is only 0.58 for such a the other approach, while it is 0.79 for our

Table 5.5: RQ_1: Affectiva threshold evaluation. The metrics are referred to the *low engagement* class.

k	TP	FP	TN	FN	Precision	Recall	F1
10	174	239	504	213	0.42	0.45	0.44
20	214	351	392	173	0.38	0.55	0.45
30	261	433	310	126	0.38	0.67	0.48
40	**302**	**522**	**221**	**85**	**0.37**	**0.78**	**0.50**
50	326	583	160	61	0.36	0.84	0.50
60	354	632	111	33	0.36	0.92	0.52
70	367	668	75	20	0.35	0.95	0.52
80	381	705	38	6	0.35	0.99	0.52
90	**386**	**717**	**26**	**1**	**0.35**	**0.99**	**0.52**
100	387	743	0	0	0.34	1.0	0.51

evalaution.[1] It is worth noting that Affectiva achieves much better results in the classification of engagement in other contexts [9]. This suggests that assessing engagement in video games is a different problem. Retraining Affectiva would probably allow to achieve results comparable to the one achieved by FFBD. We could not do that because it is a closed-source tool. K+ achieves slightly better results than Affectiva but, again, it is an optimistic implementation of the real approach, K, that would very unlikely be able to achieve such results in its intended implementation.

In an attempt to characterize engagement in video games, we measured the importance of the state-of-the-art features used in FFBD through the Info Gain algorithm [66] and ranked them based on their prediction power. We report the results in Table 5.7 (we discard features with importance 0). It can be noticed that at least a feature from two of the three categories of state-of-the-art features (expression and emotion) appears in the ranking, while the first one from behavior features (first quartile of Yaw) appears in the 15th position. Most of the important features are emotion-related, which supports the focus given in the literature to such an aspect [52, 65]. More specifically, it appears that

[1] Note that AUC is independent from the threshold, so the AUC for Affectiva is the same for both the thresholds we considered.

Table 5.6: Comparison between FFBD and the other two approaches.

Tools	Precision	Recall	F1	AUC
FFBD $_{bas}$	0.75	0.75	0.75	**0.79**
FFBD $_{comp}$	0.73	0.73	0.73	**0.79**
Affectiva $_{k=40}$	0.37	0.78	0.50	0.58
Affectiva $_{k=90}$	0.35	0.99	0.52	0.58
K+	0.27	1.00	0.53	0.42

the ones related to happiness and surprise are the most important ones (top five positions). Among the several action unit we considered in our expression-related features, only two are among the most important ones, *i.e.,* AU01 (inner brow raiser) and AU17 (chin raiser).

Answer to RQ_1. State-of-the-art approaches are highly effective in predicting the *real* engagement of players (best F1: 0.79). However, the recall for the *low engagement* class is lower than 50% for the best model.

RQ_2: Real Engagement in Practice. Table 5.8 shows the rankings of video games obtained by using the self-reported number of low engagement events and the predicted one. To this aim, we relied on FFBD, which achieves the best results. The number of low engagement events (actual and predicted by the model) at game level are significantly and strongly correlated (Spearman ρ = 0.83, with a p-value of 0.015).

It can be observed that the first three positions in the ranking (*i.e.,* the three games for which the players reported to be less engaged with) are the same. In other words, a prediction model can correctly rank them in terms of engagement. On the other hand, it struggles when the number of low engagement events is generally lower (*i.e.,* in the lower part of the ranking), probably because of its low recall. The error, in general, is at most of two ranking positions.

Table 5.7: Feature importance for emotion- (♥) and expression- 😐) features. LF indicates the features computed in the last frame (either *cont*inuous or *bin*ary).

Rank	Category	Feature	Importance
1	♥	Happiness (Q1)	0.032
2	♥	Happiness (LF-cont)	0.031
3	♥	Surprise (Q2)	0.029
4	♥	Surprise (Q3)	0.027
5	♥	Happiness (SD)	0.026
6	♥	Fear (Q1)	0.022
7	♥	Surprise (Q1)	0.022
8	😐	AU01 (Q2)	0.021
9	😐	AU17 (Q1)	0.020
10	♥	Fear (Q2)	0.020
11	😐	AU17 (Q2)	0.019
12	♥	Disgust (Q2)	0.018
13	♥	Fear (Q3)	0.018
14	😐	AU01 (Q1)	0.018
15	🔄	Yaw (Q1)	0.017
16	♥	Happiness (Q1)	0.017
17	😐	AU17 (Q3)	0.016
18	😐	AU01 (Q3)	0.016
19	♥	Surprise (SD)	0.016
20	♥	Happiness (Q2)	0.015
21	♥	Surprise (LF-cont)	0.015
22	♥	Disgust (Q3)	0.014
23	♥	Fear (TD)	0.011
24	♥	Fear (LD)	0.009
25	♥	Happiness (LF-bin)	0.007

Table 5.8: RQ_2: Video Games rankings (real and predicted), with the number of low engagement events for both the scenarios and the rank difference between the two.

#	Real (SR)		Prediction (SA)		Diff
1	Amidar	87	Amidar	44	0
2	Qbert	68	Qbert	36	0
3	Space Invader	63	Space Invader	32	0
4	Lonely	46	Gopher	31	+1
5	Gopher	39	Snake	24	+2
6	Golf Assassin	33	Lonely	23	-2
7	Snake	30	Rayman	18	+1
8	Rayman	19	Golf Assassin	16	-2

Answer to RQ_2. An approach based on state-of-the-art features — FFBD— allows to rank game sections based on the presence of low-engagement events as players would do.

5.5 Study II: Industrial Applicability

We further studied the applicability of state-of-the-art engagement detection approaches with industrial practitioners. We report below the design and results of such a study.

5.5.1 Study Design

The *goal* of this second study is to evaluate the practical applicability of an engagement detection tool in an industrial context.

This study was steered by the following research question:

RQ_3: *Would an engagement detection tool be industrially relevant?*

Context

The context of the study is composed of two subjects (participants) and an object (gameplay video). As subjects, we involved two *game developer*. We selected the participants using convenience sampling (both of them are former students at the University of Molise). Specifically, such participants are (i) Lorenzo Valente, Lead Developer in Tiny Bull Studios (Italy), and (ii) Jonathan Simeone, Full Stack Developer in Datasound (Italy), both with more than 7 years of experience in game development.

As for the object, we used as a gameplay video of the *Cyberpunk 2077* game. We chose such a video game because it has received several negative reactions when it was released [3, 4, 5], thus making it more likely to find videos in which streamers displayed low engagement. To select the specific gameplay video, we iteratively ran FFBD on the first results related to such a video game from YouTube until we found one for which it detected at least 5 low engagement events. To showcase the capability of our study, we implemented a prototype tool that, given a gameplay video and the location of the streamer's face as input, adds the predicted engagement-related information on the video (*i.e.,* indicates in which parts the player is poorly engaged). More specifically, we ran FFBD (the best-performing one) to predict the streamer's engagement for every minute of the video, and we added a bar at the bottom of the video in which we mark in red the potential low engagement events, while in green we indicate the non-low engagement events (see Fig. 5.2).

Experimental Procedure

To answer RQ_3, we conducted semi-structured interviews. Before each interview, one of the authors explained the objective of the study. Each interview lasted about 30 minutes and was conducted by one of the authors, who recorded and transcribed what the participants said for the following analyses. The interviews were based on a reflective strategy: We encouraged participants to

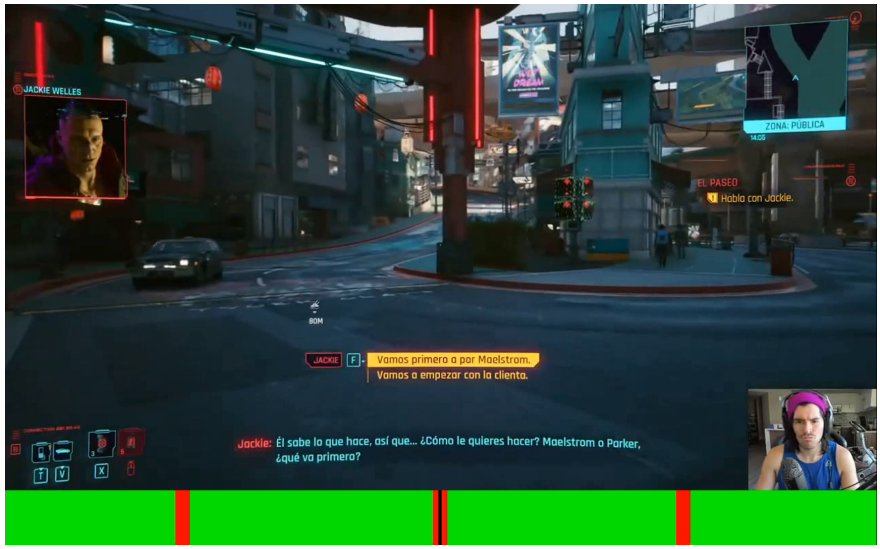


Figure 5.2: An example of what we showed to participants. The bar below shows engagement in time (red → low engagement).

Table 5.9: Interview questions.

Question	Type of response
Low Engagement Evaluation (repeated for five low engagement events)	
Is the identified low engagement event related to an engagement problem in the game?	Yes/No
Please motivate your answer	Open response
Does the provided information about the low engagement event offer enough details to replicate the issue?	Yes/No
Please motivate your answer	Open response
Witch other information might be useful in identifying low engagement events?	Open response
Overall Evaluation	
Would you use the proposed approach before the release of the game?	Yes/No
Please motivate your answer	Open response
Would you use the proposed approach after the release of the game?	Yes/No
Please motivate your answer	Open response
Would you use the proposed approach in combination with other tools?	Yes/No
Overall, how useful is the proposed approach for identifying potential low engagement events?	5-point Likert scale
Please motivate your answer	Open response
Do you have suggestions on how the proposed approach could be improved?	Open response

share their experiences, thoughts and insights in an introspective way. To do this, we mostly asked open-ended and exploratory questions. Specifically, the interview conductor asked the participants to focus on the parts of gameplay video at which the approach detected possible low engagement events (one at a time), but they could freely navigate the video to get more context. After they analyzed each of them, we asked for feedback aimed at understanding whether (i) the trigger for the identified low engagement event is actually due to a problem in the game, (ii) the information about the low engagement event is sufficient to reproduce the problem (*i.e.,* recreate the game conditions that led to the observed low engagement event), and (iii) witch other information allows to identify low engagement events.

After the evaluation of the five events, we asked questions aimed at getting feedback on an hypothetical engagement detection tool that works as the prototype tool we showed them (see Table 5.9). Specifically, we asked whether they would use this tool to identify low engagement events: (i) in the playtesting phase, before a release, and (ii) in the testing phase after the game release. Also, we asked them whether they would use it in combination with other tools (and, if so, which ones). Finally, participants were asked to indicate on a Likert scale from 1 to 5 (the higher the better) the usefulness of an engagement detection tool in identifying potential low engagement events, and whether they had possible suggestions on how the detection accuracy could be improved. For all questions, participants were asked to motivate their answers.

5.5.2 Results

Identifying Low Engagement Events. Lorenzo confirmed that three of the five parts of the video we made them watch contained low engagement events. Lorenzo confirmed that the streamer is not having fun in the first low engagement event observed. He states: “The streamer was at the end of a scripted sequence where the player is in a piloted vehicle and where he shots

the last enemies, and from that point he just had to wait for the mission to end. Scenes like these generally represent parts of the game with a lower level of engagement." In the analysis of another low engagement event, Lorenzo claims that the streamer is in an area where he is looking for a mission or clue from the map that he cannot find. He adds: "It is a problem of the game because it requires the player to find the goal in order to move forward in the game. However, if they receive too little information, players have trouble moving forward, and the engagement is lowered." An additional low engagement event is identified in the cutscenes where the streamer is in a narrative phase of the game where they have to listen to parts of dialogue. On the other hand, the two instances classified by Lorenzo as non-low engagement contain (i) a phase of the game where the streamer is exploring, and (ii) a phase of the game where the player was being very careful (*e.g.,* area full of enemies). In these two events, Lorenzo reported that it is very likely that the player was entertained in those situations.

Jonathan, instead, reported that all of the five parts of the video we made them watch contained low engagement events. However, he claimed that two of the events (the same ones that Lorenzo reported as false positives) were harder to assess. To claim this, Jonathan examined the context related to the part of the gameplay that preceded the reported low-involvement event. He observed an actual change in engagement by the streamer where in both cases there was a shift from a more dynamic to a more static gameplay phase.

Both Lorenzo and Jonathan reported that all events identified by the tool provide sufficient information to reproduce the conditions that caused of the issue. The game parts are clear and provide sufficient context.

Additional Information to identify Low Engagement Events. Lorenzo claims that a piece of information that could be useful in analyzing these events is whether the streamer is interacting with the chat. The level of engagement could be affected by the fact that the streamer is distracted reading or responding to comments. Therefore, it would be useful to integrate information about the

presence of external elements that influence the streamer (*e.g.,* whether the streamer is looking toward the chat or they are happy because they received a donation).

Jonathan claims that it would be interesting to analyze the streamer's game actions (*i.e.,* how they interact with game elements). He states: "When my engagement level drops, I start exploring the game scene and the menus. If they have too much information or contain distracting elements, I do not read the descriptions carefully, so I have to go back and forth and I have even lower engagement. For example, it could be helpful to consider also (i) the time spent in a specific game scene, (ii) the number of times the player dies, and (iii) whether they interact with the available game elements (*e.g.,* weapons). Many elements could be confusing, and if the player does not use them this could make the game more difficult and consequently lower the level of engagement."

Both Lorenzo and Jonathan claim that valuable information could be gained from analyzing the streamer's audio. By combining it with the available video information, one could understand how the streamer interacts with the game or live broadcast. For example, the streamer could express amusement while commenting during the live stream, even when immersed in a gameplay scene with minimal engagement.

Practical Application. In relation to the possibility of using a tool for engagement detection *before the release* of the game, Lorenzo stated that it could be very useful for game designers during beta-testing: "A game designer might find it useful to identify parts of the game that are particularly boring to the player. Such information would give them the ability to identify parts of the game that could be changed (*e.g.,* excluded)." Jonathan finds in the such tools a support in beta-testing as well. "I would use such a tool because it would be a great asset [...]. A low engagement event would allow me to identify a game design problem. When the player is not engaged it means that there is a part of design where I did something wrong." In relation to the use of an engagement detection tool *after the release* of the game, both Lorenzo and Jonathan were positive.

In particular, they recognized the potential of the large and growing amount of information now available through the streaming platforms. Lorenzo states, “This tool could be used at scale, on a much larger pool of streamers. By analyzing their gameplays, a game developer can get a lot of information to combine to get a complete overview. In this way, one could identify a possible part of the game where players have a low level of engagement.” Again, Jonathan shows strong confidence: “Absolutely yes, the game continues to be tested by end buyers and I want to keep monitoring it. Having such information would give me the ability to automatically detect problems even after the game is released.” Both participants would use an engagement detection tool in combination with other tools. In details, Lorenzo states, “Yes, I would use it in combination with other tools (if they are available) which automatically allow me to identify things that were missed during the testing phase.” Jonathan points out, “I do not know of any other tools that allow you to obtain information on the level of engagement, but I would use any tool similar to this one if it supports me in identifying issues missed in testing.”

Usefulness of Detection Approaches and Suggestions. In terms of usefulness, Lorenzo gives a rating of 5 out of 5. In relation to the possibility of improving the detection accuracy, he suggests taking information from the streamer’s audio and any textual information captured from the chat interaction.

In relation to the same aspect, Jonathan gives a rating of 4 out of 5. “I found consistency in what I saw. However, in some cases, I had to force myself looking at context before identifying the issue.” He also suggests that to improve the accuracy it would be interesting to allow developers to give feedback on identified low engagement events in order to recognize false positives. Therefore, a continuous learning model could be useful to improve the detection accuracy through feedback from those who use it. Similarly to Lorenzo, Jonathan highlights the possibility of obtaining information through analyzing the streamer’s audio and textual information from the chat.

Answer to RQ_3. The participants provided positive feedback on using an engagement detection tool, suggesting improvements like incorporating audio, chat, and game context to better detect low engagement events. They emphasized that the tool's usefulness depends on its integration into game developers' workflows, with potential use both before and after a game's release.

5.6 Threats to Validity

Threats to construct validity are related to possible inaccuracies in the *real* engagement assessment we performed to define our dataset. Our interruptions might have influenced the engagement of the participants. To understand if this has been an issue, we asked a subset of participants to re-watch all their eight gameplay sessions, including both their face and the game recording, and re-evaluate their level of engagement (Likert scale from 1 to 5) without knowing their previous evaluation. We asked them to try to remember whether they were enjoying playing the video game or not. We involved 10 randomly selected participants out of the 40 we involved in the study. We used the same methodology we used in our experiment to transform the evaluations from 1 to 5 into the two classes "low engagement"/"non-low engagement"). Finally, we compared the new binary labels with the ones in our dataset. We observed an agreement rate of 88.75%. More specifically, we found only 9 cases of disagreement out of a total of 80 observations. Out of these, 6 observations transitioned from low to non-low engagement, while 3 observations changed from non-low to low engagement. This result suggests that our procedure for labeling the instances is sufficiently accurate and, most importantly, it is unlikely that our procedure artificially increased the number of "low engagement" events.

However, this analysis further reveals the dynamic nature of engagement and the influence of other external and internal factors.

Threats to internal validity concern the design choices that we made that could affect the results of the study. The main threat is related to the implementation of the approaches we compared in the first study. First, we could not fully replicate one of them (K) because of the small amount of details available. To mitigate this threat, we compared our approach with an *optimistic* version of K, K+, which achieves better results than K by construction. Similarly, we did not have access to the commercial version of Affectiva, and thus we used the free implementation available online. We acknowledge that such a version might achieve worse results than the commercial one. Finally, we did not include in the comparison the approach by Chen *et al.* [34] because the replication package was not available, we lacked sufficient details to re-implement the approach from scratch. Another possible limitation is that we did not perform hyperparameter tuning for Random Forest. We decided not to run this step due to the very limited amount of data points we had: We preferred to avoid dedicating some of them for such a step rather than for training/testing. Also, our results are basically a lower bound of what can be obtained with hyperparameters tuning.

Threats to external validity concern the generalizability of our results. Our test set consists of 40 players selected from a population with specific characteristics (*i.e.,* mostly young University students). Such a sample may not be representative of the entire population. Further replications of our study and extensions of our dataset are necessary to improve such an aspect. As for our semi-structured interviews, we only involved two developers that share part of the educational background (Bachelor degree at the same University) and that work in small companies. The results we obtained might not be generalizable to the whole population of game developers, and other developers might find our approach not useful.

5.7 Final Remarks

We presented two studies. In the first, we compared three state-of-the-art approaches on a dataset we collected containing the labels representing the *real* engagement of players. In the second, we interviewed two senior game developers to assess the industrial applicability of an engagement-detection approach. The results show that some models are highly accurate in the identification of *real* low engagement events. Besides, the senior game developers we interviewed showed interest in the adoption of engagement detection approaches.

It's a funny thing, ambition. It can take one to sublime heights or harrowing depths. And sometimes they are one and the same.

— *Dishonored*

CHAPTER 6

Reproducing Game Session

Contents

6.1 Introduction

For being successful in in the games industry market it is not enough to only deliver games ensuring high engagement for the player: Games must exhibit all properties typical of high-quality software, such as good performance and reliability with as few bugs as possible. While the study proposed by Lin et al. [97] and the approaches introduced in Chapter 3, Chapter 4 and Chapter 5 are able to identify parts of the gameplay in which a issues are documented. Identifying the problems is only part of the solution. A crucial aspect for developers is the ability to replicate these problems to solve them effectively. This requires understanding the sequence of player actions (inputs) that triggered the bug.

In this chapter, we propose a technique that given as input a gameplay video and a set of possible actions that the game supports (*e.g.,* the game's supported keyboard keys), produces as output the sequence of actions which reproduce the portion of gameplay given as input. The main assumption is that gameplay videos show on screen the game actions. This would empower developer to reproduce bugs identified in gameplay videos by state-of-the-art techniques [97, 70].

In the literature, Intharah *et al.* [80] introduce DeepLogger in order to reproduce issues through the anlysis of gameplay videos. However, DeepLogger presents some limitations, for this reason we introduced RePlay to overcome these constraints. DeepLogger takes into account discrete inputs (*i.e.,* keys pressed or not pressed) and leaves the extraction of continuous inputs (*e.g.,* joysticks or mouse) as an open problem. Thus, can only predict user input logs for a game where training data is available. On the other hand, RePlay considers both discrete and continuous inputs. This comes at the cost of relying on input overlays, which are available only on a subset of gameplay videos.

We started investigating this problem in the simplest scenario in which the executed actions are shown on screen and for which the main task is to export such controls from publicly available gameplay videos. Also, we developed a technique tailored for one specific game, just as a proof of concept of the problem we want to address in the long run. Our approach uses machine learning models to discriminate between frames showing/not showing the game actions overlayed. The ones not showing them are discarded as non-gaming frames (*e.g.,* an advertisement shown on screen) while the others are further analyzed to extract the executed actions. The sequence of executed actions identified in consecutive frames composing the gameplay can be used to replicate that specific gameplay. While we are able to correctly identify $\sim$80% of the performed actions, $\sim$20% of errors leads the agent to successfully replicate only $\sim$50% of the 40 gameplays on which we tested it. Our approach can be applied to different video games, although the complexity of implementing the approach will increase with the number of possible inputs the game supports.

Our preliminary work shows that even a simple approach like the one we propose can replicate some gameplay videos (for a specific game). We expect more tailored techniques to effectively address the problem with higher precision and generalizability across games and perhaps make the approach independent from the control overlay in the footage, at the cost of increased demand for computing power. The remainder of this chapter is organized as follows. In Section 6.2, we present RePlay and its components in details. In Section 6.3 we describe the empirical study design, while in Section 6.4 we report the obtained results. In Section 6.5 we report the threats to validity. Section 6.6 concludes the chapter.

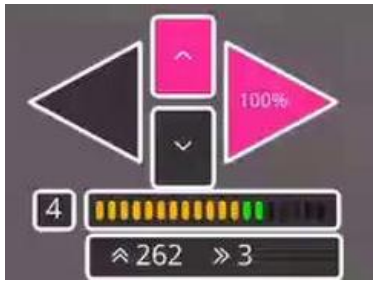
6.2 A Methodology for Game Session Reproduction

We present RePlay, an approach to extract the sequence of actions performed by a player to replicate a given gameplay. The main steps performed by RePlay are: (i) overlay detection, to identify the game controls shown on screen, if any; (ii) the extraction of the actions performed by the player; and (iii) the game reproduction. Some parts of our implementation, meant to be a proof-of-concept, are tailored for a specific video game, namely *Trackmania*, a popular racing game series developed by Nadeo[1] and published by Ubisoft[2]. We chose Trackmania due to the limited number of controls to be identified and consequent player actions to be predicted. Indeed, the possible commands are limited to acceleration, braking, and turning left/right. Also, each game starts with the beginning of the track: It is easier to identify the initial status of the game to replicate, as opposed, *e.g.,* to open-world games.

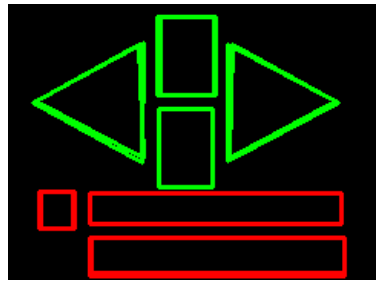
6.2.1 Overlay Detection

As one can see from Fig. 6.1a, overlays have synthesized iconography, which greatly simplifies extraction and parsing. This, given a gameplay video as input we first need to detect the game control overlay (*i.e.,* the part of the screen showing the actions performed by the player). More specifically, we need to (i) filter the frames in which the overlay is present (control overlays might be temporarily obstructed in certain frames and, thus, not reliable for input extraction), and (ii) identify the overlay elements that represent input devices (*e.g.,* buttons). Overlays may vary not only in position but also in type across different videos, depending on the input mechanisms used during gameplay. These overlays can take many different forms, including joystick movement indicators and on-screen prompts for keyboard and mouse inputs.

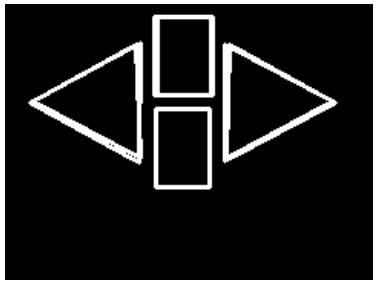
[1] https://www.nadeo.com/
[2] https://www.ubisoft.com/en-us/



(a) Reference Image



(b) Reference Image



(c) Mask extracted

Figure 6.1: Process used to rank and extract mask.

Furthermore, the overlay may vary depending on the game's specific input requirements, such as displaying key bindings alongside corresponding controls or highlighting specific actions. Thus, to achieve our two goals, we first identify the position of the overlay in the specific video, then we filter out frames that do not have the overlay, and we use the shapes identified to distinguish the parts of the overlay.

Identifying the overlay and building the mask. Our first step is to run PyShapes [125] on each frame, with the goal of identifying regular shapes (*e.g.,* rectangles and triangles) shown on screen. Then, we automatically identify a frame in which the overlay featuring the game controls is clearly visible and identifiable. To do this, we manually define a set of rules regarding the number, the types, and the relationships between the (detected) shapes that are supposed to compose the overlay of the game at hand. The mask of the overlay is extracted from one of the frames for which all such rules are met. Note that the rules are highly game- and overlay-dependent. For Trackmania, the mask must include a triangle pointing to the right, a triangle pointing to the left (for analog stick inclination) and two rectangles in the center (*i.e.,* acceleration and reverse). If we find exactly two triangles and two rectangles with the previously-mentioned relationships, we assume we found a good candidate for extracting the overlay mask. The mask is defined on the basis of the edges of the detected shapes. An example of this is shown in Figure Fig. 6.1b, where the green shapes will be considered as relevant, while the red ones will not. The process involves locating a frame with clearly visible controls and exporting a corresponding mask (Fig. 6.1c).

Filtering out frames without the overlay. We first extract from a given frame the pixels that are in the overlay mask. Then, based on the assumption that we do not know the color of the edges of the overlay, we convert such pixels in grayscale. We obtain an linear array of pixels in the order they appear in the image scanning first the rows (left-right) and then the columns (top-bottom).

We extract measures to aggregate such an array of pixels, each of which is represented by a single value ranging between 0 and 255. Such measures are: (i) maximum (*i.e.,* the brightest color), minimum (*i.e.,* the darkest color), mean, variance, contrast (*i.e.,* $\sum_{i=1}^{n}(x_i - \bar{x})^2$), and homogeneity (*i.e.,* $\sum_{i=1}^{n} \frac{x_i}{1+(\bar{x}-\bar{x})^2}$) where x stands for the pixel value of each pixel in the array. We train a Random Forest model [25] using such measures as features to classify whether or not the frame contains the overlay.

6.2.2 Control Extraction

Once detected the controls overlay, we extract the inputs from each frame, creating a sequence that can be played back.

For each frame, we extract a value representing the activation of each control supported by the game. We support two types of input devices: *buttons*, which can be either pressed or not (*i.e.,* they can be represented through boolean values), and mono-dimensional *sticks*, that can have in-between positions (*i.e.,* they can be represented as decimal numbers). We assume that each shape in the overlay corresponds to a specific game input device, which can be either a button or a joystick, and that developers manually map each overlay shape to the input device at hand. For example, the control overlay of Trackmania is composed of (i) top and bottom arrows that map the acceleration and reverse commands (buttons), and (ii) left and right arrows that represent the inclination angle of the analog stick. The top and bottom arrows are mapped to the respective buttons on the pad, while the left and right arrows are mapped to the same device (main analog stick of the pad): the right and left arrow values are mapped to positive and negative x-axis values of the analog stick, respectively. We use two different procedures to extract the input value for overlay shapes representing buttons and sticks.

Buttons. For buttons, the shape in the overlay is completely colored with an overlay-dependent color when the button is pressed, while it might be transpar-

ent (depending on the overlay at hand) when the button is not pressed. Since the color used by the overlay to represent the "button pressed" event strictly depends on the overlay at hand and on its settings, we take such a color (*active color*) as an input for this step. We first extract all the pixels contained in the overlay shape. Then, we check if all the pixels are within a certain distance from the active color: If they are, it means the button is pressed, otherwise it is not. We do this because the active color might not fully cover the game capture below, but it might be semi-transparent. To compute the distance between pixel colors we compute the difference between each color channel in the RGB space (red, green, and blue) and sum them. We say that a pixel is equal to the active color if the distance between them is lower than a threshold t. For our experiments, we use the threshold $t = 50$, which was determined through manual experimentation to optimally suit the purpose of detecting similar colors.

Sticks. For sticks, the shape in the overlay is gradually colored (again, with an overlay-dependent color) to represent the percentage of stick inclination. To predict the percentage in the arrows, we use a regression machine-learning model. The model can estimate the percentage value of inclination of a stick for a given overlay element. To do this, since we assume a shape represents mono-directional sticks (we do not support bi-directional sticks), we first summarize the whole shape content by extracting some pixels from it.

Specifically, for the triangular shapes used in the overlay of Trackmania, we extract the pixel lines beside the upper- and lower-edges of the triangle. In addition, an orthogonal line is derived from the center of the left or right base to the opposite vertex. This extraction process aims to provide the model with information about how colored is the triangle. We use the Bresenham's line algorithm [170] to get the pixel lines from the overlay shape.

For each line of pixels we compute two measures. The first one represents the size of the longest sequence of consecutive pixels that are within a certain distance from the active color, while the second one represents the percentage of pixels that are within a certain distance from the active color (*i.e.,* we also

count pixels that are not consecutive, for example when some other elements go above the overlay). We compute the distance between pixel colors in the same way we do for buttons and filter them based on the same threshold t. We use such measures, extracted for each line, as features for a linear regression model that predict the stick inclination in a range between 0 and 100, based on the overlay behaviour. A training set is needed for this step, which can be easily built for overlays that also report the numeric value of the percentage of inclination.

6.2.3 Gameplay Reproduction

We implement an agent through which we execute the sequence of commands extracted from the gameplay video. We implemented the agent using the emulator for the gamepad available in *vgamepad* [160]. We take as input the output of the previous step to create a simplified list of commands for the gamepad APIs. We set the frame-rate of the game at 30 FPS. Then, to replicate the commands, we make sure that we replicate the input devices status inferred in the previous step each $\frac{1}{30}$ of a seconds. If a given frame i is classified as without overlay during the first step, there will be no input associated to it. In such cases, we assume the input status did not change from the previous frame and, thus, we keep the input devices status extracted for the frame $i - 1$.

6.3 Empirical Study Design

The *goal* of our study is to understand if it is possible to detect command acquisition and replicate the game through RePlay. We aim to answer the following research questions (RQs):

RQ1: *To what extent is RePlay effective in identifying frames that include the controls overlay?*

RQ1 assesses the performance and reliability of the approach in identifying frames that contain the control overlay.

RQ2: *To what extent does RePlay allow to infer the input commands?*

RQ2 evaluate the accuracy of the approach to infer input commands in the gameplay analysis.

RQ3: *To what extent does the agent allow to reproduce the games?*

With this last RQ, the goal is to evaluate the accuracy of the agent to reproduce the gameplay, based on the command gathered with RePlay.

6.3.1 Context Selection

To evaluate the first two steps of RePlay (RQ1-2), we built three annotated datasets. First, we randomly sampled 3 gameplay videos of Trackmania from Twitch ($\sim$10 hours), downloaded them, and extracted the frames after fixing the frame-rate at 30 FPS. We sampled $496,770$ frames and annotated them by indicating whether the overlay was visible or not. This allowed us to define the dataset $D_{overlay}$, which contains pairs $\langle frame, overlay \rangle$. We further sampled 165 frames and manually annotate them with the information regarding the status of the two buttons (pressed or not pressed) based on the information provided in the overlay (*i.e.,* their color). As a result, we defined the dataset $D_{buttons}$, which contains triples $\langle frame, button, status \rangle$, where each frame is repeated twice (one for each $button$) and $status$ is a categorical variable (*pressed* or *not pressed*). Finally, some overlays show inside the triangles the percentage number indicating the stick inclination. We rely on them to build a third dataset of randomly sampled 246 frames. We manually labeled them with the percentage

Figure 6.2: Example of frame and controls portion exported.

indicated in the triangles. Thus, we obtained the dataset D_{stick}, which contains triples $\langle frame, stick, percentage \rangle$, where each frame is repeated twice (one for each $stick$) and $percentage$ is a numerical (integer) variable ranging from 0 to 100. To evaluate the last step of RePlay (RQ3) we built a last dataset. We randomly sampled 9 videos reporting entire game sessions of Trackmania from Twitch, all of them different from the ones used for building the previous datasets. Besides, we recorded 31 gameplay videos using the same maps and overlay shown in the Twitch clips. We use the whole RePlay on such videos to extract commands and define the agents to replicate them. We ran each game with the agent as a player and recorded the videos. As a result, we obtained D_{games}, which contains pairs $\langle video_{original}, video_{replicated} \rangle$, where $video_{original}$ is the video of the game played by a human player and $video_{replicated}$ is the one played by the agent defined with RePlay. Fig. 6.2 shows an example of a frame paired with the portion of the image exported.

6.3.2 Data Collection and Data Analysis

To address **RQ1**, we use $D_{overlay}$. For each instance $\langle frame, overlay \rangle$, we first extracted the features previously defined for the first step of RePlay from $frame$. Then, we trained and tested our model on the dataset containing such features and the $overlay$ value as a label. We used the implementation and default configuration of Random Forest available in Weka [75]. We ran a 10-fold cross validation to assess the performance of the trained model. We compute and report accuracy, precision, and recall.

To answer **RQ2**, we ran two separate evaluations for buttons and analog sticks. As for the former, we use $D_{buttons}$: For each instance $\langle frame, button, status \rangle$, we ran our button pression detection approach on the $frame$ for the shape of the $button$ at hand. We compare the predicted button pression status with the $status$ label. We report accuracy, precision, and recall for such a step. As for the latter, we use D_{sticks}: Again, for each instance $\langle frame, stick, percentage \rangle$, we extracted from the $frame$ the features defined for the second step of RePlay (stick inclination) for the shape of the specific $stick$ at hand. We trained and test a linear regression model on the dataset containing the extracted features and the $percentage$ value as labels, again using the implementation and default configuration available in Weka. We ran a 10-fold cross validation to assess the performance of such a model. We report the obtained Mean Absolute Error (MAE) and Relative Absolute Error (RAE).

To address **RQ3**, we manually compared, for each instance of D_{games}, the $video_{original}$ and $video_{replicated}$ by extracting 1-second clips every 3 seconds of the videos. We started from the beginning of the videos and proceeded until we found a difference in the clips, by synchronising the two videos based on the start of the control sequences. We consider the clips different if any of the control sections are not matching exactly. We measure, for each video, time percentage of correctly replicated game by dividing the time at which the first difference has been found by the video duration.

We publicly release the implementation of RePlay and the script we used to run the experiment and the dataset of results for each RQs in our replication package [30].

6.4 Empirical Study Results

The results of RQ1 show that the model has an accuracy of 98% and precision and recall values of 0.99 in overlay detection. Based on the evaluation results, we can conclude that the model to identify frames showing the overlay, in our context, is highly effective. Regarding the button pression detection (RQ2), RePlay achieves 99.0% of precision, 97% of recall 96% of AUC. As for the analog stick (again, RQ2) inclination prediction, instead, RePlay achieves Mean Absolute Error of 3.16 and a Relative Absolute Error of 12.88%. While the button detection is very promising, we observed that the models makes a small error in predicting the stick inclination which could hamper the perfect replication of a game. Finally, as for RQ3, we found that our agents can reproduce, on average 47.23% of the games we analyzed. While this result might seem slightly underwhelming, it is important to consider the difficulty of the tracks. In particular, the agent finds in tight turns and very rapid changes of direction especially difficult. This also happens when the player performs maneuvers very close to the boundaries of the map, such as walls. On the other hand, if the turn is simple, the agent is generally able to complete the entire turn exactly as in the original video. This observed behavior of the agents may have a strong impact on the accuracy of specific games.

We conducted an additional analysis to evaluate to what extent the agents' actions are synchronized with the players actions. To this end, we continued analyzing the 1-second clips also after the first error and only focused on the overlays. This allowed us to check what would have happened if the agent did not make a mistake. In this case, we found that, on average, 81.21% of the

commands are performed correctly.

The results obtained during the analysis aimed to answer RQ3 remains reliable and enables us to evaluate the quality of the proposed approach, as well as the implementation of the agent involved in the gameplay replication process.

6.5 Threats of Validity

In this section we summarize the threats of validity of our work.

Construct Validity. In our evaluation, we assumed that the gameplay videos, that we want to reproduce, report an overlay that shows user input commands through coloured each key input. However, gameplay videos do not always feature an overlay showing input playback, and when they do, they do not necessarily follow the standards used in our study as a reference.

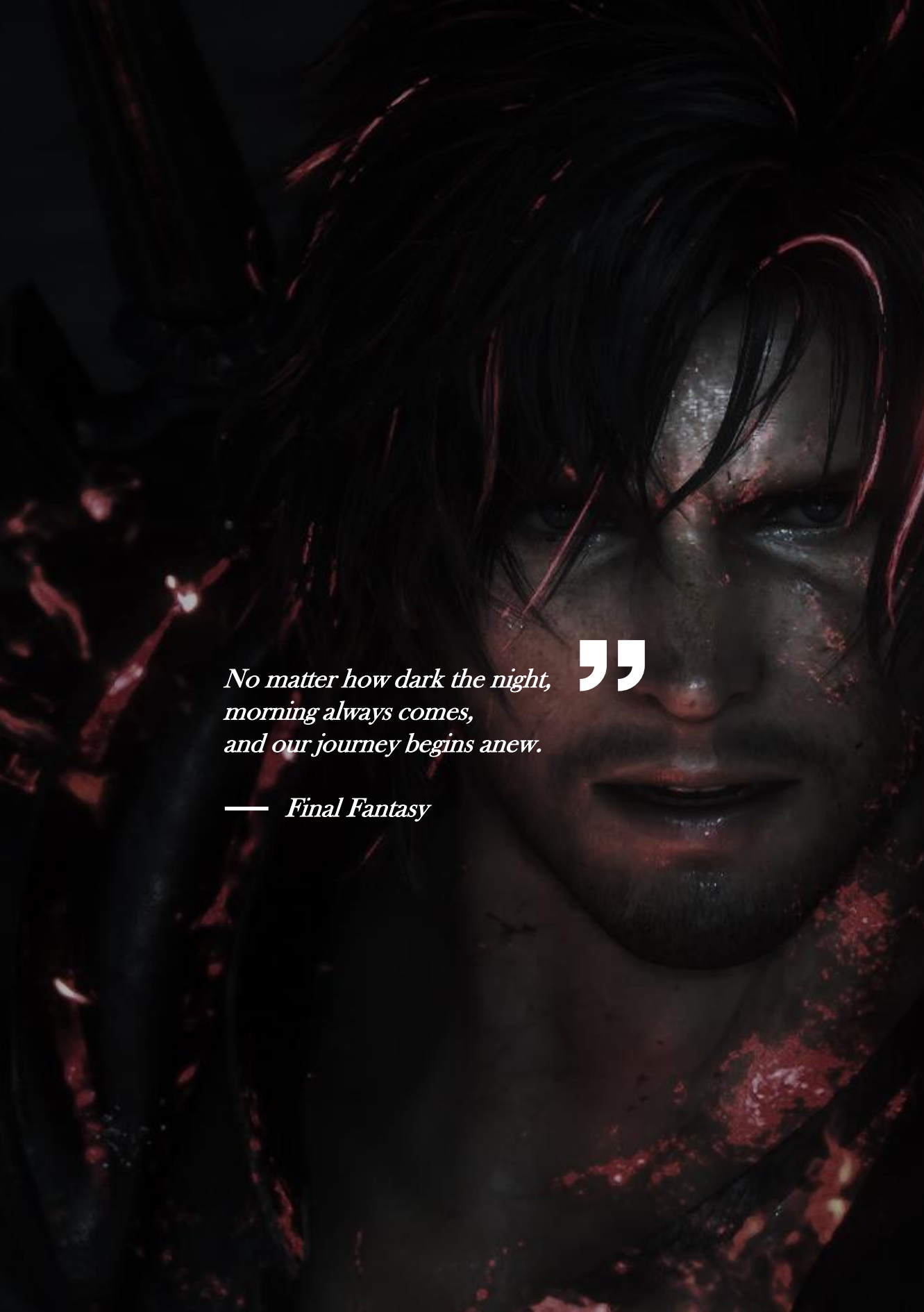
Internal Validity. The selection of the threshold for extracting commands through colour similarity introduces a potential bias that may influence the overall accuracy of overlay commands extraction. Additionally, the accuracy of the models involved in the processing applied and the potential error that resides within them can have an impact on the overall outcome.

External Validity. While certain components of our approach (*i.e.,* frame identification) are agnostic to the specific game used, others may exhibit dependence (*i.e.,* controls overlay).

6.6 Final Remarks

Video games are one of the most solid pillars of the development industry, so improving the stages of the software development process can be important to support companies in creating better gaming experiences.

We presented RePlay, a simple approach able to (i) distinguish frames with command overlay from those without, (ii) extract commands and (iii) replicate the games. Despite the very promising results, RePlay also has clear limitations. First, it requires that the gameplay videos report an overlay that shows user input commands. However, this is not always true. Second, some steps of RePlay are closely dependent on the game under consideration, restricting its generalizability unless these aspects are tailored for the targeted game. Also, we assume that game overlays are defined as specific geometric shapes. To adapt this approach to new games, it is required (i) the extraction of commands and (ii) the definition of an appropriate mask.


”
No matter how dark the night,
morning always comes,
and our journey begins anew.
— Final Fantasy

CHAPTER 7

Conclusion

In recent years, there has been a growing interest in video games, highlighting the importance of developing reliable and high quality games.

Video games, unlike traditional software systems, present a unique set of challenges when it comes to identifying and replicating bugs. The dynamic nature of player interactions, the complexity of game mechanics, and the use of non deterministic systems all contribute to the difficulty of detecting and replicating problems. Additionally, the immersive environments that games strive to create further complicate the process, as issues can arise from a combination of user input and game state, making the identification and resolution of problems particularly complex.

Millions of hours of gameplay are posted on these platforms daily [63] and, as for any other player, the streamers may run into bugs, which are thus documented in these videos. Gameplay videos provide developers with a great deal of valuable information, as they offer insight into problems encountered by players during gameplay sessions. Lin *et al.* [97] conducted an in depth study of gameplay videos posted by players on the Steam platform to understand the number of gameplay videos and the difficulty of distinguishing bug videos from other gameplay videos.

In this thesis, we explored methodologies and techniques to identify and replicate problems in video games through the automated analysis of gameplay videos. By leveraging the large amout of publicly available gameplay videos, we aim to identify issues that often go undetected during traditional testing methods, such as beta testing. By extracting the vast amount of data embedded in these videos, we aim to address the gap between players' real world experiences and the limitations of traditional quality assurance processes. The main contributions that enable automatic identification and replication of video game problems through analysis of game videos are as follows: We proposed GELID, an approach for extracting meaningful segments from gameplay videos based on textual features to identify game related issues. We introduced HASTE for detecting performance issues such as stuttering events in gameplay videos through visual analysis. We conducted an experiment to capture player engagement during real time gameplay, focusing on how game balance affects the player's experience. We proposed RePlay, a technique that allows replication of player actions from gameplay videos in order to create the sequence of events leading to an issues.

Based on the empirical investigations we conducted, we distilled some lessons learned, that we present below.

Lesson Learned 1

The textual features alone are often insufficient for identifying certain types of issues, such as performance problems or balance related issues. Slang and informal language used by streamers can complicate the automatic classification of gameplay issues.

This suggests that approaches may need to incorporate specialized video based analysis to handle the real world gaming problems. Enhancing the segmentation and clustering of gameplay events by embedding contextual data could help developers identify complex issues with higher precision.

Lesson Learned 2

The visual analysis, particularly frame by frame game analysis executed with HASTE, can be used to detect performance problems. However, the lack of hardware information in gameplay videos can lead to false positives, where stuttering events are caused by hardware limitations rather than game specific bugs.

Developers should consider using HASTE in combination with other testing tools to differentiate hardware related issues from game related problems.

Lesson Learned 3

The player engagement is a subjective experience, which is difficult to assess through gameplay video analysis without an explicit oracle from the player.

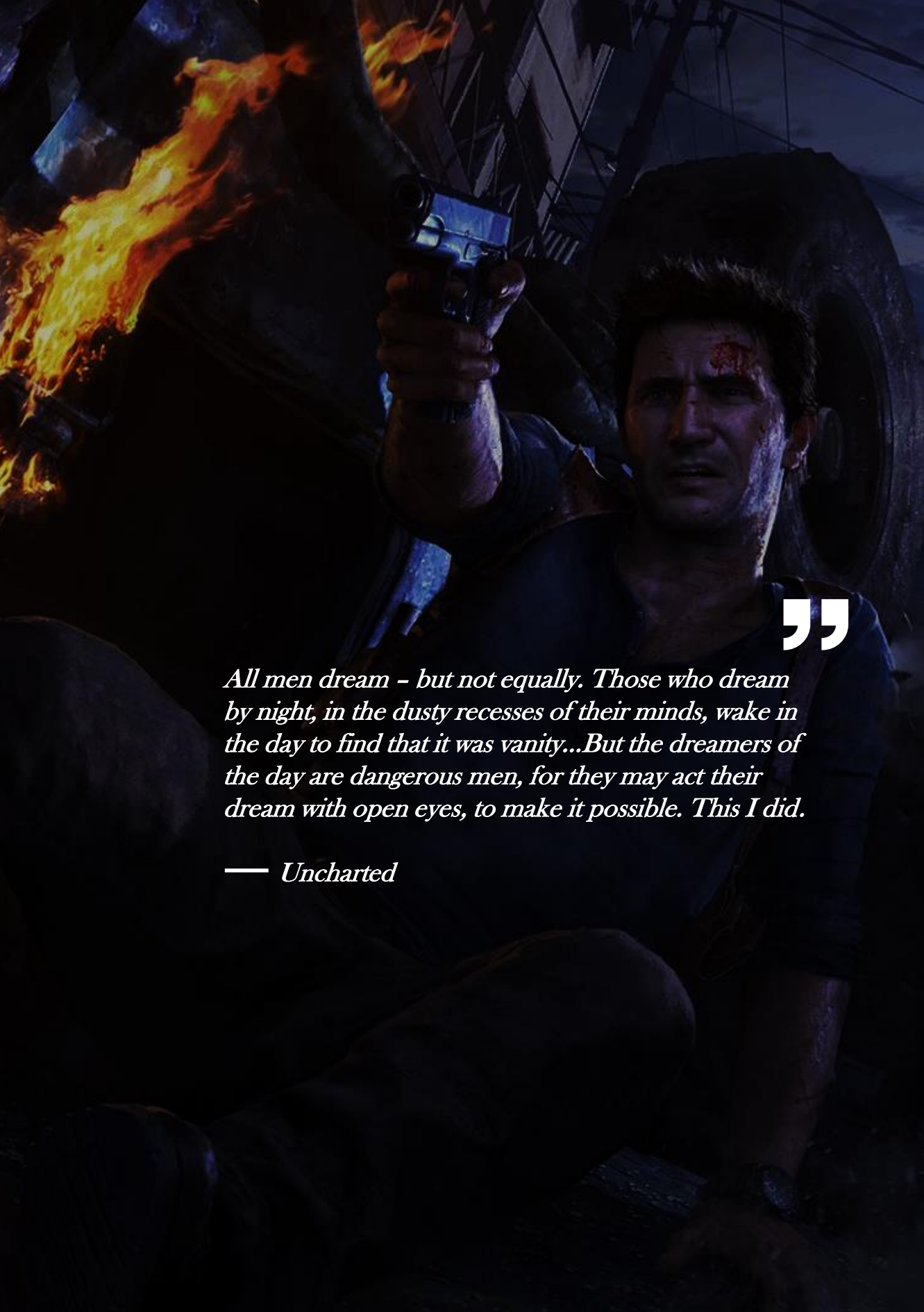
This suggests that combining video analysis with other data sources, such as direct player feedback, can provide better guidance to developers. Additionally, refining engagement detection by including contextual elements like player chat could provide a more nuanced view of player experience.

Lesson Learned 4

Through the use of these approaches developers can not only identify problems but also reconstruct the input sequences executed by players, making it easier to reproduce and correct these problems. However, a key challenge remains in the variability of input actions: different inputs can sometimes lead to the same results, while identical inputs can trigger different behaviors due to factors such as game state or randomness.

This highlights the need for precise synchronization between input data and video analysis of gameplay to ensure that developers can reproduce and solve problems consistently.

Automated analysis of gameplay videos offers a promising direction for improving the game development process, particularly for identifying and fixing bugs that are often elusive in controlled testing environments. Through this thesis, we expect to support video game developers by utilizing insights drawn from gameplay videos, which could enhance their ability to detect and resolve in-game issues. Consequently, this may also benefit the wider gaming community by potentially improving the overall quality of the gaming experience.



All men dream – but not equally. Those who dream by night, in the dusty recesses of their minds, wake in the day to find that it was vanity...But the dreamers of the day are dangerous men, for they may act their dream with open eyes, to make it possible. This I did.

— *Uncharted*

Appendices

APPENDIX A

Publications

J2 **E. Guglielmi**, S.Scalabrino, G. Bavota and R. Oliveto. *Using gameplay videos for detecting issues in video games* Empirical Software Engineering 28, 136 (2023). DOI https://doi.org/10.1007/s10664-023-10365-0

J1 **E. Guglielmi**, G. Bavota, R. Oliveto and S.Scalabrino. *Automatic Identification of Game Stuttering via Gameplay Videos Analysis*. Transactions on Software Engineering and Methodology (2024). DOI https://doi.org/10.1145/3695992

C2 S. Campanella, **E. Guglielmi**, G. Bavota, R. Oliveto and S.Scalabrino *Towards the Automatic Replication of Gameplays to Support Game Debugging*. Proceedings of the 1st ACM International Workshop on Foundations of Applied Software Engineering for Games (FSE4Games2024). DOI https://doi.org/10.1145/3663532.3664465

C1 **E. Guglielmi**, G. Bavota, N.Novielli, R. Oliveto and S.Scalabrino. *Is it Really Fun? Detecting Low Engagement Events in Video Games*. Mining Software Repository (2025). To Appear

A.1 Other Publications

J3 **E. Guglielmi**, G. Rosa, G. Bavota, S. Scalabrino and R. Oliveto. *Help Them Understand: Testing and Improving Voice User Interfaces*. ACM Transactions on Software Engineering and Methodology, 2024. DOI https://doi.org/10.1145/3654438

C4 **E. Guglielmi**, G. Rosa, G. Bavota, S. Scalabrino and R. Oliveto. *Sorry, I don't Understand: Improving Voice User Interface Testing*. Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2022. DOI https://doi.org/10.1145/3551349.3556934.

C3 A. Mastropaolo, L Pascarella, **E. Guglielmi**, M. Ciniselli, S. Scalabrino, R. Oliveto and G. Bavota. *On the robustness of code generation techniques: An empirical study on github copilot*. IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023. DOI 10.1109/ICSE48619.2023.00181

Bibliography

[1] Twitch. https://www.twitch.tv/. [Online; June 2011].

[2] Youtube. https://www.youtube.com. [Online; June 2011].

[3] Cyberpunk 2077 - steam. https://steamcommunity.com/app/1091500/discussions/7/4029094770944301584/, 2023.

[4] Cyberpunk 2077 has flatlined-crashing explained. https://www.gamepressure.com/newsroom/2023-cyberpunk-2077-has-flatlined-20-crashing-explained/z9601b, 2023.

[5] Cyberpunk 2077 three years later. https://www.grimdarkmagazine.com/review-cyberpunk-2077-three-years-later/, 2023.

[6] scikit-image. https://scikit-image.org, 2023. [Online].

[7] E. Adams and A. Rollings. *Fundamentals of game design (game design and development series)*. Prentice-Hall, Inc., 2006.

[8] Affectiva. Affectiva. https://www.affectiva.com/, 2023.

[9] Affectiva. Affectiva Media Analytics for Entertainment Content Testing. https://www.affectiva.com/product/affectiva-media-analytics-for-entertainment-content-testing/, 2023.

[10] Affectiva. All about emotion detection and affectiva's emotion metrics. Affectiva Blog, 2023.

[11] T. Ahumada and A. Bergel. Reproducing bugs in video games using genetic algorithms. In *2020 IEEE Games, Multimedia, Animation and Multiple Realities Conference (GMAX)*, pages 1–6. IEEE, 2020.

[12] S. Aleem, L. F. Capretz, and F. Ahmed. Critical success factors to improve the game development process from a developer's perspective. *Journal of computer science and technology*, 31:925–950, 2016.

[13] O. AlZoubi, B. AlMakhadmeh, M. Bani Yassein, and W. Mardini. Detecting naturalistic expression of emotions using physiological signals while playing video games. *Journal of Ambient Intelligence and Humanized Computing*, 14(2):1133–1146, 2023.

[14] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[15] Argentics. Video game testing 101: The basics everyone should know. `https://www.argentics.io/video-game-testing-101-the-basics-everyone-should-know`, 2024.

[16] S. Ariyurek, A. Betin-Can, and E. Surer. Automated video game testing using synthetic and humanlike agents. *IEEE Transactions on Games*, 13(1):50–67, 2019.

[17] E. Azizi and L. Zaman. Astrobug: Automatic game bug detection using deep learning. *IEEE Transactions on Games*, 2024.

[18] J. Banyte and A. Gadeikiene. The effect of consumer motivation to play games on video game-playing engagement. *Procedia economics and finance*, 26:505–514, 2015.

[19] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou. scvripper: video scraping tool for modeling developers' behavior using interaction data. In *2015 IEEE/ACM*

37th IEEE International Conference on Software Engineering, volume 2, pages 673–676. IEEE, 2015.

[20] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou. Extracting and analyzing time-series hci data from screen-captured task videos. *Empirical Software Engineering*, 22(1):134–174, 2017.

[21] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

[22] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*, pages 600–603. IEEE, 2020.

[23] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk. Translating video recordings of mobile app usages into replayable scenarios. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 309–321, 2020.

[24] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[25] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 0885-6125.

[26] J. H. Brockmyer, C. M. Fox, K. A. Curtiss, E. McBroom, K. M. Burkhart, and J. N. Pidruzny. The development of the game engagement questionnaire: A measure of engagement in video game-playing. *Journal of experimental social psychology*, 45(4):624–634, 2009.

[27] E. Brown and P. Cairns. A grounded investigation of game immersion. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1297–1300, 2004.

[28] Bum Hyun Lim, Jin Ryong Kim, and Kwang Hyun Shim. A load testing architecture for networked virtual environment. In *2006 8th International Conference Advanced*

Communication Technology, volume 1, pages 5 pp.–848, 2006. doi: 10.1109/ICACT.2006.206095.

[29] N. A. Butt, S. Sherin, M. U. Khan, A. A. Jilani, and M. Z. Iqbal. Deriving and evaluating a detailed taxonomy of game bugs. *arXiv preprint arXiv:2311.16645*, 2023.

[30] S. Campanella, E. Guglielmi, R. Oliveto, G. Bavota, and S. Scalabrino. Replication Package of "Towards the Automatic Replication of Gameplays to Support Game Debugging". `https://doi.org/10.6084/m9.figshare.24581826`, 2023.

[31] Y. Cao. Understanding emotional experience in video games: A psychophysiological investigation. New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392112. URL `https://doi.org/10.1145/3505270.3558370`.

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, pages 321–357, 2002.

[33] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, pages 767–778, 2014.

[34] X. Chen, L. Niu, A. Veeraraghavan, and A. Sabharwal. Faceengage: Robust estimation of gameplay engagement from user-contributed (youtube) videos. *IEEE Transactions on Affective Computing*, 13(2):651–665, 2019.

[35] J. H. Cheong. Affectiva-API-APP. `https://github.com/cosanlab/affectiva-api-app`, 2018.

[36] J. H. Cheong. Py-Feat. `https://py-feat.org/pages/intro.html`, 2022.

[37] J. H. Cheong, E. Jolly, T. Xie, S. Byrne, M. Kenney, and L. J. Chang. Py-feat: Python facial expression analysis toolbox. *Affective Science*, 4(4):781–796, 2023.

[38] C. Cho, D. Lee, K. Sohn, C. Park, and J. Kang. Scenario-based approach for blackbox load testing of online game servers. In *2010 International Conference on*

Cyber-Enabled Distributed Computing and Knowledge Discovery, pages 259–265, 2010. doi: 10.1109/CyberC.2010.54.

[39] S. Choudhury and A. Bhowal. Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. In *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pages 89–95. IEEE, 2015.

[40] J. Clement. Video game market revenue worldwide from 2019 to 2029. `https://www.statista.com/forecasts/1344668/revenue-video-game-worldwide`, "2024".

[41] J. Clement. Number of active streamers on twitch worldwide from january 2018 to april 2024. `https://www.statista.com/statistics/746173/monthly-active-streamers-on-twitch/`, "2024".

[42] J. Clement. Number of hours watched on twitch worldwide from 2nd quarter 2018 to 3rd quarter 2022. `https://www.statista.com/statistics/1030852/hours-watched-twitch/`, "2024".

[43] J. Clement. Satista gaming. `https://www.statista.com/topics/1680/gaming`, 2024.

[44] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.

[45] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[46] J. Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[47] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk. It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 957–969. IEEE, 2021.

[48] R. Cowie, N. Sussman, and A. Ben-Ze'ev. Emotion: Concepts and definitions. *Emotion-oriented systems: the HUMAINE handbook*, pages 9–30, 2011.

[49] A. D'Angelo, C. Di Sipio, C. Politowski, and R. Rubei. Playmydata: a curated dataset of multi-platform video games. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 525–529, 2024.

[50] F. "De Mesentier Silva, S. Lee, J. Togelius, and A. Nealen". "ai as evaluator: Search driven playtesting of modern board games". In *"WS-17-01"*, "AAAI Workshop - Technical Report", pages "959–966". "AI Access Foundation", "2017". "31st AAAI Conference on Artificial Intelligence, AAAI 2017".

[51] M. Dimitrievski. Truelist. `https://truelist.co/blog/gaming-statistics/`, 2024.

[52] K. Doherty and G. Doherty. Engagement in hci: conception, theory and measurement. *ACM Computing Surveys (CSUR)*, 51(5):1–39, 2018.

[53] EA. Electronic arts playtesting. `https://www.ea.com/playtesting/about.`, 2023. [Online].

[54] L. Ermi and F. Mäyrä. Fundamental components of the gameplay experience: Analysing immersion. In *DiGRA Conference*. Citeseer, 2005.

[55] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[56] Fandom. Development of grand theft auto v. `https://ballsgames.fandom.com/wiki/Development_of_Grand_Theft_Auto_V`.

[57] S. Feng and C. Chen. Gifdroid: Automated replay of visual bug reports for android apps. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1045–1057, 2022.

[58] S. Feng, M. Xie, Y. Xue, and C. Chen. Read it, don't watch it: Captioning bug recordings automatically. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2349–2361. IEEE, 2023.

[59] P. A. Flach. Roc analysis. In *Encyclopedia of machine learning and data mining*, pages 1–8. Springer, 2016.

[60] R. Fridlund and E. Gustafsson. Engagement in video games: A comparison between a linear and a branching narrative, 2023.

[61] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[62] I. Games. Game Balance: A Pivotal Issue in Game Design. `https://www.innovecsgames.com/blog/game-balance-a-critical-issue-in-designing-top-titles/`, 2023.

[63] S. Gaming. Number of hours streamed on leading gaming live stream platform. `https://www.statista.com/statistics/1030809/hours-streamed-streamlabs-platform/`, 2023.

[64] I. Gauk and C.-P. Bezemer. Detecting discrepancies between subtitles and audio in gameplay videos with echotest. *IEEE Transactions on Games*, 2024.

[65] D. Girardi, A. Ferrari, N. Novielli, P. Spoletini, D. Fucci, and T. Huichapa. The way it makes you feel predicting users' engagement during interviews with biofeedback and supervised learning. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 32–43, 2020.

[66] S. Gnanambal, M. Thangaraj, V. Meenatchi, and V. Gayathri. Classification algorithms with attribute selection: an evaluation study using weka. *International Journal of Advanced Networking and Applications*, 9(6):3640–3644, 2018.

[67] M. Granato, D. Gadia, D. Maggiorini, and L. A. Ripamonti. Feature extraction and selection for real-time emotion recognition in video games players. In *2018 14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 717–724. IEEE, 2018.

[68] E. Guardiola. The gameplay loop: a player activity model for game design and analysis. In *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*, pages 1–7, 2016.

[69] E. Guglielmi, S. Scalabrino, G. Bavota, and R. Oliveto. Replication Package of "Automatic Identification of Game Stuttering via Gameplay Videos Analysis". `https://figshare.com/s/600d3be6169203ce6cac`, 2022.

[70] E. Guglielmi, S. Scalabrino, G. Bavota, and R. Oliveto. Using gameplay videos for detecting issues in video games. *Empirical Software Engineering*, 28(6):136, 2023.

[71] E. Guglielmi, S. Scalabrino, G. Bavota, and R. Oliveto. Replication package of using gameplay videos for detecting issues in video games. `https://figshare.com/s/3de4d6958a57073dfa1b`, 2023.

[72] E. Guglielmi, S. Scalabrino, G. Bavota, and R. Oliveto. Replication package of "automatically detecting low engagement events in video games". `https://figshare.com/s/40e95d35efe037fee1f7`, 2023.

[73] M. Guzdial, S. Shah, and M. Riedl. Towards automated let's play commentary. *arXiv preprint arXiv:1809.09424*, 2018.

[74] I. Habunek. Twitch-dl. `https://github.com/ihabunek/twitch-dl/`, 2024. [Online].

[75] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11 (1):10–18, 2009.

[76] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[77] T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[78] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. 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)*, pages 426–435. IEEE, 2015.

[79] IMotion. IMotion. `https://imotions.com/`, 2022.

[80] T. Intharah and G. J. Brostow. Deeplogger: Extracting user input logs from 2d gameplay videos. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 221–230, 2018.

[81] S. E. Jones. *The meaning of video games: Gaming and textual strategies*. Routledge, 2008.

[82] Y. "Jung, B.-H. Lim, K.-H. Sim, H. Lee, I. Park, J. Chung, and J. Lee. "venus: The online game simulator using massively virtual clients". In *"Systems Modeling and Simulation: Theory and Applications"*, pages "589–596", "2005".

[83] J. Juvrud, G. Ansgariusson, P. Selleby, and M. Johansson. Game or watch: The effect of interactivity on arousal and engagement in video game media. *IEEE Transactions on Games*, 14(2):308–317, 2021.

[84] P. Karvelis, D. Gavrilis, G. Georgoulas, and C. Stylios. Topic recommendation using doc2vec. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2018.

[85] T. Killedar, G. Suriya, P. Sharma, M. Rathor, and A. Gupta. Fuzzy logic for video game engagement analysis using facial emotion recognition. In *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 481–485, 2021. doi: 10.1109/SPIN52536.2021.9566124.

[86] P. Krieter and A. Breiter. Analyzing mobile application usage: generating log files from mobile screen recordings. In *Proceedings of the 20th international conference on human-computer interaction with mobile devices and services*, pages 1–10, 2018.

[87] A. Kultima and K. Alha. “hopefully everything i’m doing has to do with innovation”: Games industry professionals on innovation in 2009. In *2010 2nd International IEEE Consumer Electronics Society’s Games Innovations Conference*, pages 1–8. IEEE, 2010.

[88] S. Kumar. How much did gta 5 cost to make? https://afkgaming.com/gaming/general/how-much-did-gta-5-cost-to-make#:~:text=Considering%20this%2C%20the%20game%20was,%24265%20Million%20USD%20in%20total., 2023.

[89] A. Kunst. Video live streaming sites usage in the u.s. in 2023. https://www.statista.com/forecasts/997210/video-game-streaming-sites-usage-in-the-us, 2024.

[90] H. Kuramoto, M. Kondo, Y. Kashiwa, Y. Ishimoto, K. Shindo, Y. Kamei, and N. Ubayashi. Do visual issue reports help developers fix bugs? a preliminary study of using videos and images to report issues on github. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 511–515, 2022.

[91] S. Kwon, J. Ahn, H. Choi, J. Jeon, D. Kim, H. Kim, and S. Kang. Analytical framework for facial expression on game experience test. *IEEE Access*, 10: 104486–104497, 2022.

[92] LambdaTest. Game testing. https://www.lambdatest.com/learning-hub/game-testing, 2024.

[93] G. Laudato, S. Scalabrino, N. Novielli, F. Lanubile, and R. Oliveto. Predicting bugs by monitoring developers during task execution. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1–13. IEEE, 2023.

[94] R. S. Lazarus. *Emotion and adaptation*. Oxford University Press, 1991.

[95] C. Lewis, J. Whitehead, and N. Wardrip-Fruin. What went wrong: a taxonomy of video game bugs. In *Proceedings of the fifth international conference on the foundations of digital games*, pages 108–115, 2010.

[96] C. Li, S. Gandhi, and B. Harrison. End-to-end let's play commentary generation using multi-modal video representations. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–7, 2019.

[97] D. Lin, C.-P. Bezemer, and A. E. Hassan. Identifying gameplay videos that exhibit bugs in computer games. *Empirical Software Engineering*, 24(6):4006–4033, 2019.

[98] C. Ling, K. Tollmar, and L. Gisslén. Using deep convolutional neural networks to detect rendered glitches in video games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 66–73, 2020.

[99] Z. Luo, M. Guzdial, and M. Riedl. Making cnns for video parsing accessible: event extraction from dota2 gameplay video using transfer, zero-shot, and network pruning. In *Proceedings of the 14th international conference on the foundations of digital games*, pages 1–10, 2019.

[100] T. W. MacFarland, J. M. Yates, T. W. MacFarland, and J. M. Yates. Mann–whitney u test. *Introduction to nonparametric statistics for the biological sciences using R*, pages 103–132, 2016.

[101] L. MacLeod, M.-A. Storey, and A. Bergen. Code, camera, action: How software developers document and share program knowledge using youtube. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 104–114. IEEE, 2015.

[102] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[103] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[104] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[105] J. V. Moniaga, A. Chowanda, A. Prima, M. D. T. Rizqi, et al. Facial expression recognition as dynamic game balancing system. *Procedia Computer Science*, 135: 361–368, 2018.

[106] P. Moreno-Ger, J. Torrente, Y. G. Hsieh, and W. T. Lester. Usability testing for serious games: Making informed design decisions with user data. *Advances in Human-Computer Interaction*, 2012(1):369637, 2012.

[107] E. Murphy-Hill, T. Zimmermann, and N. Nagappan. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In *Proceedings of the 36th International Conference on Software Engineering*, pages 1–11, 2014.

[108] L. Myers and M. J. Sirois. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences*, 12, 2004.

[109] OpenAI. Gym Documentation. `https://www.gymlibrary.dev/`, 2022.

[110] A. Ortony, G. L. Clore, and A. Collins. *The cognitive structure of emotions*. Cambridge university press, 2022.

[111] F. O. Ozkok and M. Celik. A new approach to determine eps parameter of dbscan algorithm. *International Journal of Intelligent Systems and Applications in Engineering*, 5(4):247–251, 2017.

[112] C. Paduraru, M. Paduraru, and A. Stefanescu. Rivergame-a game testing tool using artificial intelligence. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 422–432. IEEE, 2022.

[113] S. Pan, G. J. Xu, K. Guo, S. H. Park, and H. Ding. Video-based engagement estimation of game streamers: An interpretable multimodal neural network approach. *IEEE Transactions on Games*, 2023.

[114] L. Pascarella, F. Palomba, M. D. Penta, and A. Bacchelli. How is video game development different from software development in open source? In A. Zaidman, Y. Kamei, and E. Hill, editors, *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 392–402. ACM, 2018.

[115] J. Pfau and M. Seif El-Nasr. Balancing video games: A player-driven instrument. In *Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, pages 187–195, 2023.

[116] J. Pfau and M. Seif El-Nasr. On video game balancing: Joining player-and data-driven analytics. *ACM Games: Research and Practice*, 2(3):1–30, 2024.

[117] J. Pfau, J. D. Smeddinck, and R. 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*, pages 153–164, 2017.

[118] J. Pfau, A. Liapis, G. N. Yannakakis, and R. Malaka. Dungeons & replicants ii: automated game balancing across multiple difficulty dimensions via deep player behavior modeling. *IEEE Transactions on Games*, 15(2):217–227, 2022.

[119] C. Pinto Gomez and F. Petrillo. Improving bug reproduction through game engine state analysis. In *Proceedings of the ACM/IEEE 8th International Workshop on Games and Software Engineering*, pages 28–35, 2024.

[120] C. Politowski, F. Petrillo, and Y.-G. Guéhéneuc. A survey of video game testing. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 90–99. IEEE, 2021.

[121] C. Politowski, F. Petrillo, G. C. Ullmann, and Y.-G. Guéhéneuc. Game industry problems: An extensive analysis of the gray literature. *Information and Software Technology*, 134:106538, 2021.

[122] C. Politowski, Y.-G. Guéhéneuc, and F. Petrillo. Towards automated video game testing: Still a long way to go. In *Proceedings of the 6th international ICSE workshop on games and software engineering: engineering fun, inspiration, and motivation*, pages 37–43, 2022.

[123] C. Politowski, F. Petrillo, G. ElBoussaidi, G. C. Ullmann, and Y.-G. Guéhéneuc. Assessing video game balance using autonomous agents. In *2023 IEEE/ACM 7th International Workshop on Games and Software Engineering (GAS)*, pages 25–32. IEEE, 2023.

[124] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza. Codetube: extracting relevant fragments from software development

video tutorials. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 645–648. IEEE, 2016.

[125] PyShapes. Pyshapes. `https://github.com/sudoRicheek/PyShapes`, 2023.

[126] Python. Opencv. `https://opencv.org`, 2023. [Online].

[127] Python. Pytube. `https://github.com/pytube/pytube`, 2023. [Online].

[128] Python. spacy. `https://spacy.io/`, 2023. [Online].

[129] Python. Video-kf. `https://pypi.org/project/video-kf/`, 2023. [Online].

[130] H. Ramchoun, Y. Ghanou, M. Ettaouil, and M. A. Janati Idrissi. Multilayer perceptron: Architecture optimization and training. *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(1):26–30, 2016.

[131] X. Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[132] S. Roohi, C. Guckelsberger, A. Relas, H. Heiskanen, J. Takatalo, and P. Hämäläinen. Predicting game difficulty and engagement using ai players. *Proceedings of the ACM on Human-Computer Interaction*, 5(CHI PLAY):1–17, 2021.

[133] B. Rosner. *Fundamentals of Biostatistics*. Brooks/Cole, Boston, MA, 7th edition edition, 2011.

[134] R. E. Santos, C. V. Magalhães, L. F. Capretz, J. S. Correia-Neto, F. Q. da Silva, and A. Saher. Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2018.

[135] Satista. `https://www.statista.com/topics/1680/gaming/.`, 2022. [Online].

[136] Satista. `https://www.statista.com/statistics/292056/video-game-market-value-worldwide/.`, 2023. [Online].

[137] Satista. Satista. https://www.statista.com/statistics/292056/video-game-market-value-worldwide/, 2023.

[138] S. Scalabrino, G. Bavota, B. Russo, M. Di Penta, and R. Oliveto. Listening to the crowd for the release planning of mobile apps. *IEEE Transactions on Software Engineering*, 45(1):68–86, 2017.

[139] H. Schønau-Fog and T. Bjørner. "sure, i would like to continue" a method for mapping the experience of engagement in video games. *Bulletin of Science, Technology & Society*, 32(5):405–412, 2012.

[140] S. Shah, M. Guzdial, and M. O. Riedl. Automated let's play commentary. *arXiv preprint arXiv:1909.02195*, 2019.

[141] O. Sharaievskyi. What is the game testing process? https://pinglestudio.com/blog/game-testing/what-is-the-game-testing-process, 2024.

[142] J. Shen, H. Yang, J. Li, and Z. Cheng. Assessing learning engagement based on facial expression recognition in mooc's scenario. *Multimedia Systems*, pages 1–10, 2022.

[143] A. M. Smith, M. J. Nelson, and M. Mateas. Computational support for play testing game sketches. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'09, page 167?172. AAAI Press, 2009.

[144] J. R. Smith. *Integrated spatial and feature image systems: Retrieval, analysis and compression*. Columbia University, 1997.

[145] T. Souček and J. Lokoč. Transnet v2: An effective deep network architecture for fast shot transition detection, 2020.

[146] Steam. Conan exiles. https://store.steampowered.com/app/440900/Conan_Exiles/, 2024.

[147] Steam. Dayz. https://store.steampowered.com/app/221100/DayZ/, 2024.

[148] Steam. New world. `https://store.steampowered.com/app/1063730/New_World/`, 2024.

[149] S. Stöckli, M. Schulte-Mecklenbeck, S. Borer, and A. C. Samson. Facial expression analysis with affdex and facet: A validation study. *Behavior research methods*, 50: 1446–1460, 2018.

[150] M. R. Taesiri, F. Macklon, and C.-P. Bezemer. Clip meets gamephysics: Towards bug identification in gameplay videos using zero-shot transfer learning. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 270–281, 2022.

[151] M. R. Taesiri, F. Macklon, S. Habchi, and C.-P. Bezemer. Searching bug instances in gameplay video repositories. *IEEE Transactions on Games*, 2024.

[152] S. Tang, L. Feng, Z. Kuang, Y. Chen, and W. Zhang. Fast video shot transition localization with deep structured models. In *Asian Conference on Computer Vision*, pages 577–592. Springer, 2018.

[153] The Last of Us. `https://youtu.be/yH5MgEbBOps?t=3494`. [Online].

[154] Y. Tian, D. Lo, and J. Lawall. Sewordsim: Software-specific word similarity database. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 568–571, 2014.

[155] E. J. Toy, J. V. Kummaragunta, and J. S. Yoo. Large-scale cross-country analysis of steam popularity. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1054–1058. IEEE, 2018.

[156] A. Truelove, E. S. de Almeida, and I. Ahmed. We'll fix it in post: What do bug fixes in video game update notes tell us? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 736–747. IEEE, 2021.

[157] A. Truelove, S. Rong, E. S. de Almeida, and I. Ahmed. Finding the needle in a haystack: Detecting bug occurrences in gameplay videos. *arXiv preprint arXiv:2311.10926*, 2023.

[158] R. Tufano, S. Scalabrino, L. Pascarella, E. Aghajani, R. Oliveto, and G. Bavota. Using reinforcement learning for load testing of video games. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 2303–2314. ACM, 2022. doi: 10.1145/3510003.3510625. URL https://doi.org/10.1145/3510003.3510625.

[159] Twitch Stream Time. https://twitchtracker.com/statistics/stream-time, 2024. [Online].

[160] Vgamepad. Vgamepad. https://github.com/yannbouteiller/vgamepad, 2023.

[161] T. Wan, H. Jun, H. Zhang, W. Pan, and H. Hua. Kappa coefficient: a popular measure of rater agreement. *Shanghai archives of psychiatry*, 27(1):62, 2015.

[162] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13 (4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

[163] WEKA. https://www.weka.io/, "2024". [Online].

[164] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.*, pages 194–203. IEEE, 2004.

[165] What is microstuttering and how do I fix it. https://www.pcgamer.com/what-is-microstutter-and-how-do-i-fix-it/, "2022". [Online].

[166] J. Whitehill, Z. Serpell, Y.-C. Lin, A. Foster, and J. R. Movellan. The faces of engagement: Automatic recognition of student engagementfrom facial expressions. *IEEE Transactions on Affective Computing*, 5(1):86–98, 2014.

[167] E. N. Wiebe, A. Lamb, M. Hardy, and D. Sharek. Measuring engagement in video game-based environments: Investigation of the user engagement scale. *Computers in Human Behavior*, 32:123–132, 2014. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2013.12.001. URL https://www.sciencedirect.com/science/article/pii/S0747563213004433.

[168] E. N. Wiebe, A. Lamb, M. Hardy, and D. Sharek. Measuring engagement in video game-based environments: Investigation of the user engagement scale. *Computers in Human Behavior*, 32:123–132, 2014.

[169] B. Wilkins, C. Watkins, and K. Stathis. A metric learning approach to anomaly detection in video games. In *2020 IEEE Conference on Games (CoG)*, pages 604–607. IEEE, 2020.

[170] W. E. Wright. Parallelization of bresenham's line and circle algorithms. *IEEE Computer Graphics and Applications*, 10(5):60–67, 1990.

[171] A. Wrześniewska and M. Skublewska-Paszkowska. Video game performance analysis on selected operating systems. *Journal of Computer Sciences Institute*, 29:317–324, 2023.

[172] H. Wu, Y. Guo, and C. B. Seaman. Analyzing video data: A study of programming behavior under two software engineering paradigms. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 456–459. IEEE, 2009.

[173] Y. Wu, Y. Chen, X. Xie, B. Yu, C. Fan, and L. Ma. Regression testing of massively multiplayer online role-playing games. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 692–696. IEEE, 2020.

[174] Y. Yan, N. Cooper, O. Chaparro, K. Moran, and D. Poshyvanyk. Semantic gui scene learning and video alignment for detecting duplicate video-based bug reports. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[175] W. Yang, M. Rifqi, C. Marsala, and A. Pinna. Physiological-based emotion detection and recognition in a video game context. In *2018 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.

[176] X. Zhang and A. M. Smith. Retrieving videogame moments with natural language queries. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–7, 2019.

[177] Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1): 43–52, 2010.

[178] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 772–784. IEEE, 2019.

[179] G. Zoeller. Game development telemetry in production. *Game analytics: Maximizing the value of player data*, pages 111–135, 2013.

External Thesis Evaluators:

Prof.ssa Venera Arnaoudova, Washington State University, Pullman.
Prof. Fabio Petrillo, École de Technologie Supérieure - ÉTS, Canada.

Final Exam Commitee:

Dr. Sarra Habchi, Ubisoft La Forge, Canada.
Prof. Massimiliano Di Penta, University of Sannio, Italy.
Prof. Stefano Ricciardi, University of Molise, Italy.

I hereby declare that this thesis, submitted as partial fulfilment to obtain the academic degree of Doctor of Philosophy (Ph.D.) in Biology and Applied Science, is my own unaided work. I have not used sources other than those indicated, and all direct and indirect sources are acknowledged as references. Parts of this dissertation have been published in international journals and/or conference articles.

Termoli (CB), 28/11/2024