Debugging Video Games: A Systematic Mapping

Adrien Vanègue, Valentin Bourcier, Fabio Petrillo, Steven Costiou

Debugging Video Games: A Systematic Mapping

Adrien Vanègue
Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL
F-59000 Lille, France
adrien.vanegue@inria.fr

Valentin Bourcier
Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL
F-59000 Lille, France
valentin.bourcier@inria.fr

Fabio Petrillo
École de Technologie Supérieure (ÉTS)
Montréal, Canada
fabio.petrillo@etsmtl.ca

Steven Costiou
Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL
F-59000 Lille, France
steven.costiou@inria.fr

Abstract

The video game industry is a vast and lucrative sector that generates significant revenue. However, a recurring issue within this industry is the release of games with numerous bugs. The process of debugging, a challenging and expensive undertaking in software development, becomes crucial in rectifying these issues. It is worth noting that such bugs can significantly impact the commercial performance of games in the market. Therefore, an imperative arises to foster further academic research that presents dedicated debugging techniques aimed at enhancing the process specifically within the realm of video game development.

In this paper, we employ a systematic mapping study methodology to discern the existing body of knowledge concerning debugging practices for video games. Through a systematic selection process, we selected 21 relevant studies for analysis, enabling the synthesis of data and facilitating the creation of a overview that encapsulates the state of the art on debugging video games. We identified work analyzing challenges of debugging games, proposing bug detection techniques or dedicated debugging tools. However, these work are always bound to specific contexts or kinds of games. Our analysis shows that there is no academic body of knowledge about debugging video games.

Thus, this initial exploration not only raises pertinent questions but also presents prospects for conducting empirical and controlled experiments to enhance our understanding of effective debugging strategies within the context of video game development. We complete our analysis with a discussion of testing and debugging techniques, and how they could help and open new research opportunities for the debugging of video games.

CCS Concepts: • **Software and its engineering** → *Maintaining software*; *Development frameworks and environments*; *Software maintenance tools*; *Software notations and tools.*

Keywords: debugging, video game, software quality, software engineering

1 Introduction

Debugging represents a significant cost of software development [32]. Video games are no exception. For example, in December 2020, the game *Cyberpunk 2077* had many bugs on the PS4 console platform that led to countless refunds to players. The editor ultimately removed the game from their online store [6]. Other games are released with lots of bugs, and consequently some of them fail to meet success. *Skyrim* is considered to be one of the buggiest game of all time [6]. Five years after its PC release in 2011, some bugs were still present in the PS4 port and are still unfixed as of today. These bugs, however, are mostly overlooked by players because the game is considered as one of the best of all time. *Battlefield 2042* was released in 2021 with lots of bugs too. It was even impossible to play a game for an entire day, and, for example, players were able to drive vehicles in the air to blow up helicopters [5]. As a consequence, its number of players on Steam is inferior to previous Battlefield titles [22]. In 2018, *Call Of Duty: Black Ops 4*'s zombies mode had been barely playable for a month. Crashes happened randomly on all platforms, which was frustrating for the players. Some game elements took a month to be discovered and were unplayable because of the game's instability [11].

This is surprising for such a profitable industry [31]. We therefore wondered why video games were so buggy, and what was the state-of-the-art of video game debugging. Murphy-Hill et al. [20] studied how video games development differs from traditional software development. They concluded that game designers have a vague idea of their objectives, which often change. Because of that, important software engineering are not performed well or at all. Few tests are performed, little to no time is spent to design the software architecture, little to no code is reused as creativity and optimization are prioritized. Truelove et al. [33] identified and explained the most frequent, the most recurrent and the most severe types of bugs in video games. Crashes appear to be the most recurring bug in video games, followed by graphical bugs [33]. According to developers, this is because it is hard to identify the exact cause of these bugs [33]. However, they did not study how to debug each bug type.

We therefore wondered what were the debugging practices in the video game industry, what methods, tools and techniques were used to debug games, and how theses practices were covered by the academic literature. We performed a systematic mapping of the literature by searching on the Scopus database. The Scopus search returned 89 papers, out of which we selected 21 papers that only partially covered the topics of debugging video games.

Thus, we map the literature (section 2), and we analyze its results (section 3). We extend our analysis with insights from software engineering methods and techniques, and how we could build new research on video game debugging (section 4). Finally, we expose the possible threats to validity for our study (section 5) before we conclude (section 6).

2 Research Method

This research adheres to the established guidelines for conducting systematic mapping studies [10, 26]. In the subsequent sections, we outline the key aspects of our study design, in accordance with these guidelines.

2.1 Research Question

Our goal is to understand the debugging practices in the video game industry. This will help us to clarify research directions to improve methods, tools and techniques for debugging video games. We therefore formulate the following research question:

> What is the state of the art and practice of video-game debugging?

2.2 Search and Selection Process

The search and selection process utilized in this study has been designed as a multi-phase process. The purpose behind this approach is to maintain complete control over both the quantity and specific attributes of the studies considered at each stage. By implementing such a systematic and well-defined process, we ensure scrutiny and management of the research artifacts throughout the various phases of our investigation.

Initial search. We conducted an automated search on the Scopus scientific databases and indexing systems. The choice of these electronic databases and indexing systems was based on several factors: (i) their established reputation as reliable resources for conducting systematic literature reviews in the field of software engineering [26], (ii) their wide accessibility, and (iii) their capability to export search outcomes in formats that are easily computable and well-defined.

```
TITLE-ABS-KEY(debug* AND ("video game" OR "computer game*"))
    AND (LIMIT-TO(SUBJAREA,"COMP"))
```

Listing 1. Query string for automatic search on Scopus.

The query string utilized in this study is presented in Listing 1. We designed the query to encompass a wide range of studies pertaining to the subject matter at hand, specifically the existing research about debugging and video game on the computer science domain. To ensure inclusiveness, a generic approach was adopted, focusing solely on the scope of our investigation. For consistency, we applied the query string to the titles, abstracts, and keywords of papers obtained from all the selected data sources utilized in this mapping.

Selection. The process of study selection involved systematic approach to identify relevant research articles within the scope of this mapping. The selected studies were then subjected to thorough examination and evaluation based on predetermined inclusion and exclusion criteria, ensuring their suitability for the research objectives. Through this process, a subset of studies meeting the defined criteria was identified for further analysis and synthesis. The inclusion (I) and exclusion (E) criteria of our study are:

- I1 - Research studies pertaining to the identification or analysis of bugs in video games, discussing aspects such as bug detection and bug analysis within the context of video game development.
- I2 - Research studies addressing debugging techniques specifically tailored for video games. These papers delve into the investigation, analysis, and proposal of effective methods for debugging within the context of video game development. By focusing on the unique challenges posed by video games, these studies contribute valuable insights and suggestions aimed at improving the overall debugging process in this domain.
- I3 - Research studies exploring the challenges associated with debugging video games. These papers delve into the intricate complexities that arise during the debugging process specific to the domain of video game development.
- E1 - Research studies whose main goal is not debugging in the context of video games.
- E2 - Research studies using video games and/or debugging as demonstrative scenarios of an unrelated topic.
- E3 - Research studies that are not peer reviewed.

After this step, we selected 18 studies.

Snowballing. We conducted an exhaustive snowballing process in both directions (backward and forward) [26]. The main goal of this stage is to expand the set of possibly relevant papers by focusing on papers either citing or being cited by each previous selected studies. Backward snowballing proved to be iterative, as we first conduct this process right after the selection process on one depth level (we did not investigate the additional papers references), before adding

relevant cited papers during the data extraction, thus increasing the depth level. After the snowballing process, we selected 3 additional studies.

The 21 selected studies are referenced in Table 1. We discuss our findings from the data synthesis of the selected studies in Sections 3 and 4.

3 Debugging Video Games

In this section, we analyze the results from our literature exploration. We divided the 21 papers in four categories: publication trends, challenges of video game debugging, approaches to detect bugs in games and tools to debug video games. We discuss how these work contribute to answer the question of how to debug video games.

3.1 Publication Trends

Publication trends allows us to gain insights into the evolving landscape of research in video game debugging. We identify patterns, emerging themes, and notable gaps in the literature. The set of selected studies distribution is illustrated among publication years, approaches and main topics.

Publication Years. Figure 1 illustrates the distribution of studies conducted over the years, shedding light on the evolving landscape of research in the field. From 21 selected studies, fifteen (71%) were published over the last ten years. Despite the relatively modest sample size of studies, this shows a notable upsurge in interest over the past decade.

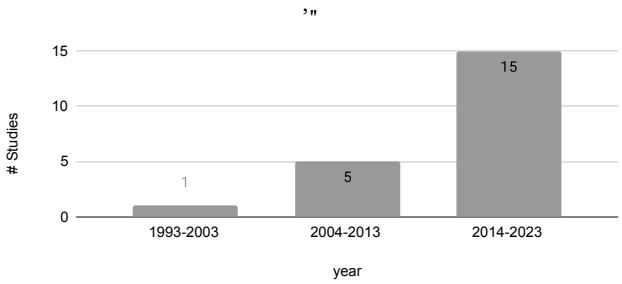

Figure 1. Selected studies over the years (1993 – 2023)

Approaches. Figure 2 provides an overview of selected studies, categorized based on their research approaches. The findings reveal several predominant approaches adopted by researchers in the field. Artificial Intelligence (AI) emerges as the most prevalent approach, accounting for 7 out of 21 studies. Another notable approach observed in the studies is the proposal of tools, with 5 out of 21 studies focusing on this aspect. These studies emphasize the development and design of practical tools that can aid in specific tasks or domains.

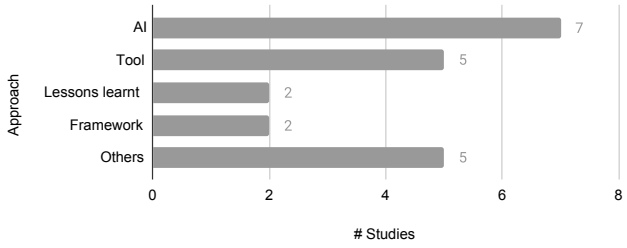

Figure 2. Approaches of selected studies.

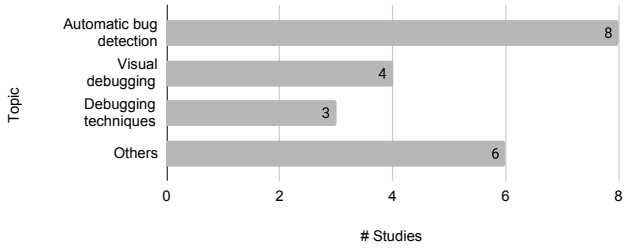

Figure 3. Topics of selected studies.

Main topics. Figure 3 provides a comprehensive overview of the diverse range of topics addressed in the selected studies. Notably, automatic bug detection emerges as the predominant area of focus, attracting the attention of 8 out of the total 21 studies. This substantial representation underscores the significance placed on developing automated approaches for bug identification.

Another area of interest is visual debugging, which captures the attention of 4 studies. Additionally, investigations into various debugging techniques are accounted by 3 studies, showing exploration of different methodologies to enhance the debugging experience.

3.2 Challenges of video games debugging

We examine hereafter the specific pain points that need to be addressed to improve video game debugging.

Volatile and unclear specifications. The rich and dynamic nature of game worlds presents a significant challenge when it comes to planning for all possible scenarios [36]. Due to the complex interplay of various game elements, accurately specifying game behavior often requires multiple iterations of implementation, testing, and refinement. Without comprehensive specifications, detecting bugs in games becomes even more challenging since they can arise from scenarios that have not been adequately covered.

Non-determinism and reproducibility. Non-determinism poses significant challenges for debugging, as it renders the reproducibility of values, events, and behaviors either impossible or exceptionally difficult. This inherent uncertainty is particularly prevalent in domains such as

Table 1. List of selected secondary studies

Ref.	Title	Year	Type	Approach	Topic
[36]	An Intelligent IDE for Behavior Authoring in Real-Time Strategy Games	1993	Short paper	Tool	Time-Travel debugging
[12]	Debugging CUDA	2011	Full paper	Lessons learnt	Debugging techniques
[9]	SteerBug: an interactive framework for specifying and detecting steering behaviors	2009	Full paper	Framework	Steering behaviors
[13]	Creating and Debugging Performance CUDA C	2012	Chapter	Lessons learnt	Debugging techniques
[23]	Distributed DeepThought: Synchronising complex network multiplayer games in a scalable and flexible manner	2013	Short paper	Tool	Network debugging
[24]	RoboCup Advanced 3D Monitor	2004	Full paper	Tool	Visual debugging
[17]	Go-Explore Complex 3D Game Environments for Automated Reachability Testing	2022	Article	AI (RL)	Automatic bug detection
[7]	Beyond Playing to Win: Creating a Team of Agents with Distinct Behaviours for Automated Gameplay	2023	Article	AI (MAP-Elites)	Automatic bug detection
[37]	A Metric Learning Approach to Anomaly Detection in Video Games	2020	Short paper	AI (Deep learning)	Automatic bug detection
[1]	A co-evolutionary genetic algorithms approach to detect video game bugs	2022	Article	AI (Genetic Algorithms)	Automatic bug detection
[8]	Why is debugging video game AI hard?	2019	Short paper	Interviews	Debugging techniques
[34]	Automated bug finding in video games: A case study for runtime monitoring	2017	Article	Runtime Monitoring	Automatic bug detection
[21]	A Framework for the Semi-Automatic Testing of Video Games	2021	Short paper	Framework	Visual debugging
[35]	Validating the plot of Interactive Narrative games	2021	Full paper	Tool	Narrative Debugging
[4]	Supporting Distributed Real-Time Debugging in Online Games	2017	Short paper	Tool	Tracing
[25]	Testing and Debugging Functional Reactive Programming	2017	Article	Formal verification	Temporal assertion
[14]	Sparse Record and Replay with Controlled Scheduling	2019	Full paper	Record and Replay	Automatic bug detection
[3]	GLIB: Towards Automated Test Oracle for Graphically-Rich Applications	2021	Full paper	AI (CNN)	Automatic bug detection
[18]	Automatically Detecting Visual Bugs in HTML5 Canvas Games	2023	Full paper	Approach	Visual debugging
[16]	Identifying gameplay videos that exhibit bugs in computer games	2019	Article	AI (Random Forest)	Visual Debugging
[15]	A deep reinforcement learning technique for bug detection in video games	2023	Article	AI (Deep RL)	Automatic bug detection

games and graphical simulations. Video games, for instance, frequently exhibit non-deterministic states, *e.g.,* due to user inputs. User inputs contribute to non-determinism as they introduce a dynamic and unpredictable element. Each player interaction, such as keystrokes, mouse movements, or controller actions, introduces a unique set of inputs, leading to diverse outcomes and branching paths within the game. Consequently, reproducing a specific sequence of actions precisely can prove arduous, hindering the debugging process. In the field of video games development this difficulty is amplified by two characteristics:

1. Autonomous agents, or AI entities [8]: in simulations where agents interact with each others in large-scale virtual worlds, interactions between two agents can be and difficult to track [9].
2. Concurrency: in distributed architectures [4], reproducing a game behavior requires to reproduce sequences of distributed events in the right order. To ensure this order, it sometimes requires synchronization between components [14]. The higher are the number of possible states and the number of different components, the more complex it is to reproduce a bug.

3.3 Detecting Bugs in Games

In the following, we review the different methodologies, algorithms, and techniques proposed in the literature to detect bug in video games. We distinguish in the literature two major approaches: temporal states-based techniques and AI-based techniques.

Temporal states-based techniques. Temporal states-based techniques are used to monitor and analyze the various temporal states encountered within a video game. These techniques serve two primary purposes: checking the consistency of the observed temporal states and automatically replaying specific states. One application of these techniques involves checking the consistency of the observed temporal states [34]. This process enables the identification of temporal anomalies that may arise during gameplay. By defining a set of expected temporal properties, developers can ensure that the game functions as intended and identify potential bugs or issues that may hinder the player's experience. On the one hand, by expressing such properties developers formally define what behavior is expected from the game that a manual tester could misunderstand. On the other hand, these properties allows developers to define what should be verified and when it should be verified. Temporal states-based techniques can be utilized to automatically replay specific states encountered within the game [34]. This capability is particularly useful for debugging purposes, as it allows developers to reproduce and analyze specific scenarios that led to a particular state or event.

An efficient method of ensuring the fulfillment of a specific property during gameplay is by employing a run-time monitor. Such monitor continually observes the game's execution and automatically verifies if the desired property is being satisfied. However, in order for the monitoring process

to be effective, the game needs to be instrumented to send relevant events to the monitor. One possibility is to integrate the event-sending mechanism directly into the game loop [34]. This allows for seamless communication between the game and the monitor, ensuring that relevant events are promptly captured and analyzed. Alternatively, functional languages can be leveraged to facilitate event transmission [25]. By taking advantage of the inherent properties of functional languages, developers can design the game in a way that naturally generates events, which are then forwarded to the monitoring system.

In certain cases, especially in the context of distributed video games, monitoring may require additional tools or techniques [4]. These tools enable the monitoring system to handle the complexities introduced by distributed environments, ensuring that property verification remains accurate and reliable even in such scenarios.

AI-Based Technique. Advancement in AI technology has opened up new possibilities for detecting bugs and irregularities in games. In this context, two notable approaches have been proposed, which leverage AI algorithms to enhance anomaly detection capabilities in different game environments.

One such approach, presented by Lu et al. [17], adapts the *Go-Explore* algorithm specifically for 3D game environments. The algorithm maintains a cache of previously explored game states and utilizes heuristics to select and explore promising states, aiming to test for *reachability*. By systematically exploring the game space, this technique effectively uncovers potential anomalies that might occur during gameplay, enabling developers to address them proactively.

Another approach, introduced by Guerrero-Romero et al. [7], focuses on generating agents with generic game behavior heuristics to detect bugs in 2D sprite-based games. These agents are designed with diverse goals, allowing them to identify different types of bugs within the game. However, it is important to note that this approach is limited to detecting bugs in 2D sprite-based games only. Nonetheless, the portability of these agents allows them to be reused to play new levels, thereby facilitating automated testing of the game. By employing a "team" of different agents that play the game, developers can systematically evaluate its performance and identify potential anomalies.

Nantes et al. [21] present an innovative framework designed to enhance and streamline the process of bug detection on video games. The framework introduces agents that leverage AI and computer vision modules to detect graphical bugs, making the process more efficient and reliable. These agents are equipped with AI capabilities and are specifically trained to identify graphical anomalies within video games. By utilizing a computer vision module, the agents are able to detect and isolate graphical objects of interest. For instance, when detecting shadow aliasing, the AI agent focuses on isolating and analyzing shadows within the game. After isolating the relevant graphical objects, the computer vision module applies pattern recognition techniques to the extracted images. This step allows for the identification and categorization of visual bugs present in the game. By automating the detection and analysis of these bugs, the authors argue that the framework can significantly contribute to the automation of non-regression tests, particularly those related to visual aspects of the game.

Wilkins et al. [37] propose an approach utilizing deep metric learning to detect anomalies, defined as *"the manifestation of a bug in the raw observation space of a human player"*. The authors demonstrate the effectiveness of their method by employing games as a dataset of anomalies, allowing an AI to learn and identify representative state spaces associated with these anomalies. While this approach presents promising results, it is important to acknowledge its limitations. One of the primary limitations of this approach is its inability to detect freezing anomalies. Freezing anomalies refer to situations where a system becomes unresponsive or hangs, often leading to a frustrating user experience. Due to the unique nature of freezing anomalies, they may not be adequately captured or detected by the deep metric learning framework.

Albaghajati and Ahmed [1] propose an approach that leverages the power of genetic algorithms to generate game states and uncover potential bugs within the vast space of all possible states in a game. This solution goes beyond the detection of a single type of bug and is capable of identifying various bug categories, including invalid states, conflicting overlapping goal states, deadlocks, and level-design issues. By exploring the state space using genetic algorithms, the proposed approach demonstrates its effectiveness in uncovering complex bugs in three different games. However, it is important to note that this solution may not be comprehensive in detecting bugs related to physics, animations, level design, and visual effects. These types of bugs often involve intricate interactions and intricate details that are not easily captured solely through the exploration of the state space.

3.4 Debugging Tools in Video Games

Finally, we explore the tools specifically designed for debugging video games. We survey the existing landscape of debugging tools to provide an overview of the available resources that aid developers to understand and resolve issues within video games.

Perez and Nilsson [25] developed a debugger specifically designed for *Yampa* FRP games, aiming to enhance the debugging experience. Their approach involved extending the functionality of *Yampa* by incorporating a communication channel that facilitates the exchange of commands between the game and an external debugger. Additionally, they implemented a mechanism to record the history of all inputs and sampling times. During each iteration of the game loop, the program actively checks for incoming debugger commands,

enabling various operations. These operations include the ability to save, load, or replace specific portions or the entirety of an input trace. Furthermore, the debugger allows users to pause, stop, or play the simulation as a whole or until a specific condition is met. It also provides functionality for navigating backward and forward in time, either by moving or stepping through the simulation. One notable feature of their debugger is its capability to detect violations of temporal properties. By leveraging temporal properties, the debugger can identify instances where these properties are not upheld, providing valuable insights during the debugging process. This debugger is limited to games implemented in FRP languages, on which it relies to debug deterministically.

Virmani et al. [36] developed an IDE to help code, detect and debug game AIs in real-time strategy games (RTS). The developer determines the condition failures for all the existing AI behavior. Then, these behaviors execute inside the game while recording a trace. The tool offers a timeline visualization of when an AI game behavior failed. The developer can go back in time just before that moment to debug, and replay part of an execution with a new AI behavior. However, this approach focuses only on RTS games.

Penedo et al. developed the RoboCup Advanced 3D Monitor [24] to visualize simulated soccer games. This tool can be used as a debugger to inspect in real time the internal state of a player, such as its position. However, this tool remains specific to this particular soccer game.

Langdon [12, 13] gives debugging techniques to debug CUDA, stating that a lot of game hardware use CUDA. However, CUDA is not limited to game hardware, and these technique do not help directly for debugging video games.

Chen et al. [3] propose GLIB, a test oracle that allows to detect UI glitches. The authors classified causes of UI glitches in games, in order to be able to generate code that generates UI glitches. Then, these glitches are used to teach patterns of these different UI glitches to a neural network-based model, in order to be able to detect them. This approach can be applied to any graphically-rich application, and is not specific to video games.

Veloso et al. [35] developed a prototype to detect issues in the design of interactive narrative in video games. The tool is able to load an interactive story to offer a visualization of it as a branching tree, with all possible narrative paths. In order to detect different types of narrative design issues, it is possible to keep track of various variables such as game variables, endings' reachability or unacessible plots.

Finally, Festa et al. [4] suggest a debugging architecture to debug online games that have been designed to be multiplatform and are performant enough to collect all bullets in an FPS game. Pedersen et al. [23] propose another distributed architecture for multiplayer games that allows developers to easily access game data at any time, which eases the use of debugging tools to visualize or alter these data.

3.5 Discussion

The main topics addressed in the studies primarily focus on debugging challenges, automatic bug detection, and various debugging techniques, but a notable gap exists in the exploration of debugging practices within game development. From this literature study, we learned nothing about the reality of video game debugging and its practices.

Even though our literature review shows some work has been done on software engineering activities related to game debugging, it is important to point out that most of these work do not provide a solution that could be generalized to any game.

The suggested automatic bug detection techniques are all tailored to a specific range of bug types (temporal failures, visual bugs, invalid states, AI bugs, concurrency bugs, issues in interactive narrative design...) and the suggested debugging techniques are tailored to games with specific characteristics (FRP, 2D sprite-based, RTS, online multiplayer, ...). Some solutions don't provide any solution to debugging games specifically because they deal with games as a subset of other applications (hardware used in games, graphically-rich applications, concurrent application).

Basically, there is no academic nor practical body of knowledge of video game debugging. This opens new intriguing questions, and calls for further research. For example, if there is no body of knowledge, where from and how do game developers learn to debug their games? What tools do they use? Can debugging tools and techniques used in other software engineering domains be applied to video game debugging? In the next section, we briefly reflect on our software engineering experience to highlight the potential of starting new research for testing and debugging video games.

4 How Can Software Engineering Help Debugging Video Games?

In this section, we discuss software testing and software debugging research that could help video games debugging. In our literature study, we did not find applications of research in these areas to the domain of games. Therefore, we argue that more research should attempt to apply testing and debugging techniques to video games.

4.1 Software Testing

Bugs are notoriously hard to detect in video games, notably because many possible interactions need to be tested [33]. The only requirement of a game is that it must be fun, which is subjective and thus complicated to evaluate and test. As a result, the testing activity mainly focuses on testing the gameplay [28] instead of software quality. This testing phase occurs after a substantial part of the software has already been developed, because testers need to wait until games become playable.

As games have unclear and volatile specifications [20], classical testing techniques such as *Test-Driven Development* cannot be used to test game features early. This also makes automatic bug detection techniques difficult to apply [28]. Testing is therefore considered a potential waste of time and is thus overlooked [20]. That could explain why video games lack unit testing, non-regression testing, etc.

Video game studios employ few Quality Assurance testers because of money [28], which makes it difficult to test games manually. The problem is that, compared to traditional software, the life cycle of a game is unknown and will depend on its success, so leaders do not know if the invested money will be worth it. In addition, managers generally lack knowledge in software development, so they ignore the importance of good practices in software engineering such as testing [20].

Some techniques exist in the literature to automate video game testing. However, video game developers are globally skeptical about them [27]. This is considered a waste of time and money that could be spent elsewhere. For example, one technique requires to build a pipeline of complex testing [27]. Building such a pipeline requires hiring dedicated engineers, which costs more than manual testers. These solutions are not viable for indie games because they are too costly.

We argue that software testing could play an important role in delivering high-quality games, and that more research should be conducted on that topic. The current research on video game testing shows that improving testing is partially blocked by cultural and economic contexts. As a consequence, developers often resort to ad-hoc techniques that prove difficult to apply universally across various game genres, and the hard testing problems (such as volatile game specifications) remain untackled.

4.2 Software Debugging

Researchers proposing novel debugging methods, techniques and tools sometimes use video games as examples to illustrate or evaluate the application of their work. They sometimes claim that their approach is adequate for the debugging of video games. A systematic survey of debugging research using video games as evaluation scenarios is outside the scope of this paper. We only give various examples of such research.

Active Expressions [29] is a technique for reactive programming that focuses on state changes detection. Users choose places in the code where they check if the state of their program changes, and which software module to notify of the changes (*e.g.*, a debugger). The authors argue that this technique is applicable in software architectures using a global loop, such as video games, where their instrumentation can be called at each frame update.

Edit Transactions [19] is a live programming technique that encapsulates program changes into reversible transactions that can be applied dynamically to a running program. It is illustrated through an example of a simple game with an animated ball. In that example, the authors change the code of the step animation of the ball by adding a call to a new method that is not yet implemented. Live programming environments usually directly apply the change to the step animation code, which leads to a game crash, as an unknown method is called. Using Edit Transactions, the code changes are delayed until the developers judge that all changes make sense together. The transactions then dynamically update the code of the running game.

Timelapse [2] is a debugger that records and replay executions of web applications. The debugger provides tools to locate and replay interactive behaviors (*i.e.*, user inputs, events) that provoke failures. Timelapse is illustrated on a 2D space invader with a bug due to a problematic user input that makes the spaceship fire two bullets instead of one. In the example, the record and replay feature help discover that the bug's source is due to the user pressing the *fire* key between two event generations. This video game scenario is used in a small control experiment to evaluate the debugger's effectiveness. While the authors claim Timelapse made bug reproduction simpler, their experiment did not show any improvement or degradation of the developers' effectiveness compared with standard tools.

The *Reactive Inspector* [30] is a tool that implements a methodology to debug reactive programs. It proposes abstractions and actions dedicated techniques to reactive programming, such as navigating signals or inspecting values and nodes in the program dependency graph. The authors evaluate their tool empirically on different tasks, one of them being an arcade pong game. During that task, participants had to answer questions about the state of the game when particular events happened (*e.g.*, when the ball bounces the third time on the bottom border). Using the Reactive Inspector, participants were significantly faster at debugging the pong game than with traditional tools.

Despite some potential contributions, these studies do not show evidence that their contributions specifically improve the debugging of video games. Authors only claim that their approaches are convenient to debug video games [29] or that their game example is representative of a class of programs that encompasses games [2]. When video games are used in empirical evaluations [2, 30], they are simplistic games with few lines of code (compared to AAA games). Therefore the question remains of the suitability of those techniques for debugging video games and if they would significantly improve the situation for big games. We argue that a literature survey would be an interesting starting point for identifying contributions that could benefit video game debugging, as well as the associated challenges.

5 Threats to Validity

The systematic mapping is a challenging process. There is a possibility of certain papers going unnoticed, and data

extraction often tends to be biased since achieving complete impartiality is hard. Thus, we made every effort to adhere strictly to established guidelines and best practices. Our aim is to be as objective as possible in our approach. Therefore, we have taken the necessary steps to identify and address the main threats to the validity of our study. In this regard, we describe the potential challenges to validity and outline measures we have implemented to mitigate some of them. Some challenges are out of scope of this paper, and we detail how we plan to mitigate them in further research.

Search Query Scope. Although we made efforts to narrow down our search query by targeting the keywords *"debugging"* and *"video/computer games"*, it is important to acknowledge that there may be studies that fall outside the defined scope. These studies could potentially provide valuable insights or alternative perspectives on the topic. We plan to address this limitation in future extended work, by incorporating additional keywords such as *"program comprehension"*, *"bug tracking"*, or *"software quality"* in the context of video games. By expanding our exploration within the debugging domain, we can increase the chances of capturing a more comprehensive range of relevant studies.

Database Selection. Our study search was limited to the Scopus database, which may introduce bias by excluding potentially relevant studies from other databases or sources. To mitigate this limitation, we implemented a snowballing process, both forward and backward, to identify additional relevant articles that might have been missed in the initial search. Snowballing helps minimize the risk of overlooking important evidence, but it is still possible that some relevant studies may have been inadvertently excluded. In future work, we plan to search additional databases (*e.g.*, Google Scholar) and apply a similar snowballing methodology. This might yield additional results that were potentially missed by the Scopus searched.

Inclusion/Exclusion Criteria. Despite applying rigorous inclusion/exclusion criteria to filter out irrelevant studies, it is possible that some articles obtained in our search do not directly address our research question. This issue may arise due to the ambiguity of the study abstracts or titles, or the lack of consistent terminology used in the field. However, we carefully reviewed each selected article to ensure its relevance to our research objectives and exclude studies that do not align with our topic of interest.

Limited Research Availability. We observed that a significant portion of the results obtained from our search were unrelated to our research question. This could be due to the scarcity of literature specifically exploring the intersection of debugging and video games. While this finding may limit the number of studies included in our analysis, it highlights the novelty and potential for further research in this area.

6 Conclusion and future work

In this paper, we try to learn and understand how developers debug video games. We performed a literature analysis of video game debugging, from which we selected 21 relevant papers. We extracted information about video games debugging challenges, bug detection and debugging tools. We have gained insights into the evolving research landscape, identified specific pain points, explored diverse detection strategies, and surveyed available debugging resources. However, we were surprised to find only 21 relevant papers, as bugs are a major pain in the video game industry. In addition, none of these papers actually plainly address the problem of debugging video games, and we learned nothing about the practices of game developers when they encounter bugs.

While the selected studies shed light on some debugging challenges, the prevalence of automatic bug detection in research work, and general debugging techniques, the absence of research investigating video game debugging practices calls for a more focused attention. Exploring and advancing our understanding of debugging games has the potential to significantly benefit the industry and to contribute to the overall advancement of game development practices. To that end, we plan to expand our literature exploration further, by searching for intersections between video games and topics closely related to debugging (*e.g.*, program comprehension). We also plan to interview game developers to understand their debugging practices, and to obtain direct knowledge from the field.

Acknowledgments

This work was funded by the ANR JCJC OCRE Project (https://anr.fr/Project-ANR-21-CE25-0004).

References

[1] Aghyad Albaghajati and Moataz Ahmed. 2022. A co-evolutionary genetic algorithms approach to detect video game bugs. *Journal of Systems and Software* 188 (2022), 111261.

[2] Brian Burg, Richard Bailey, Amy J Ko, and Michael D Ernst. 2013. Interactive record/replay for web application debugging. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 473–484.

[3] Ke Chen, Yufei Li, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Wei Yang. 2021. GLIB: Towards Automated Test Oracle for Graphically-Rich Applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1093–1104. https://doi.org/10.1145/3468264.3468586

[4] Dario Festa, Dario Maggiorini, Laura Anna Ripamonti, and Armir Bujari. 2017. Supporting distributed real-time debugging in online games. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. 737–740. https://doi.org/10.1109/CCNC.2017.7983226

[5] Gamepur. [n. d.]. The 10 buggiest games on release. https://www.gamepur.com/guides/10-buggiest-games-on-release (accessed on January 13rd 2022).

[6] Gamerant. [n. d.]. Cyberpunk 2077 & 9 Other PS4 Games That Were Too Buggy To Enjoy At Launch. https://gamerant.com/cyberpunk-2077-ps4-games-buggy-launch/ (accessed on January 13rd 2022).

[7] Cristina Guerrero-Romero, Simon Lucas, and Diego Perez-Liebana. 2023. Beyond Playing to Win: Creating a Team of Agents with Distinct Behaviours for Automated Gameplay. *IEEE Transactions on Games* (2023).

[8] Nathan John, Jeremy Gow, and Paul Cairns. 2019. Why is debugging video game AI hard?. In *Proc. AISB AI & Games symposium*. 20–24.

[9] Mubbasir Kapadia, Shawn Singh, Brian Allen, Glenn Reinman, and Petros Faloutsos. 2009. Steerbug: an interactive framework for specifying and detecting steering behaviors. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics symposium on computer animation*. 209–216.

[10] Barbara Kitchenham and Pearl Brereton. 2013. A systematic review of systematic review process research in software engineering. *Information and Software Technology* 55, 12 (2013), 2049–2075. https://doi.org/10.1016/j.infsof.2013.07.010

[11] Kotaku. [n. d.]. Black Ops 4 Zombies Won't Stop Crashing, Frustrating Easter Egg Hunters. https://kotaku.com/black-ops-4-zombies-wont-stop-crashing-frustrating-eas-1830308966 (accessed on January 13rd 2022).

[12] William B. Langdon. 2011. Debugging CUDA. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 415–422. https://doi.org/10.1145/2001858.2002028

[13] W. B. Langdon. 2012. *Creating and Debugging Performance CUDA C*. Springer Berlin Heidelberg, Berlin, Heidelberg, 7–50. https://doi.org/10.1007/978-3-642-28789-3_2

[14] Christopher Lidbury and Alastair F Donaldson. 2019. Sparse record and replay with controlled scheduling. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 576–593.

[15] Dayi Lin, Cor-Paul Bezemer, and Ahmed E. Hassan. [n. d.]. ([n. d.]).

[16] Dayi Lin, Cor-Paul Bezemer, and Ahmed E. Hassan. 2019. Identifying gameplay videos that exhibit bugs in computer games. *Empirical Software Engineering* 24, Article 2 (set 2019), 4006 – 4033 pages. Issue 6. https://doi.org/10.1007/s10664-019-09733-6

[17] Cong Lu, Raluca Georgescu, and Johan Verwey. 2022. Go-Explore Complex 3D Game Environments for Automated Reachability Testing. *IEEE Transactions on Games* (2022).

[18] Finlay Macklon, Mohammad Reza Taesiri, Markos Viggiato, Stefan Antoszko, Natalia Romanova, Dale Paas, and Cor-Paul Bezemer. 2023. Automatically Detecting Visual Bugs in HTML5 Canvas Games. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 15, 11 pages. https://doi.org/10.1145/3551349.3556913

[19] Toni Mattis, Patrick Rein, and Robert Hirschfeld. 2017. Edit Transactions: Dynamically Scoped Change Sets for Controlled Updates in Live Programming. *The Art, Science, and Engineering of Programming* 1, 2 (apr 2017). https://doi.org/10.22152/programming-journal.org/2017/1/13

[20] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. 2014. Cowboys, Ankle Sprains, and Keepers of Quality: How is Video Game Development Different from Software Development?. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/2568225.2568226

[21] Alfredo Nantes, Ross Brown, and Frederic Maire. 2021. A Framework for the Semi-Automatic Testing of Video Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 4, 1 (Sep. 2021), 197–202. https://doi.org/10.1609/aiide.v4i1.18697

[22] PcGamesN. [n. d.]. Battlefield 2042 player count destroyed by 50k Battlefield 1 players. https://www.pcgamesn.com/battlefield-2042/player-count (accessed on January 13rd 2022).

[23] Karsten Pedersen, Christos Gatzidis, and Barry Northern. 2013. Distributed DeepThought: Synchronising complex network multi-player games in a scalable and flexible manner. In *2013 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change (GAS)*. 40–43. https://doi.org/10.1109/GAS.2013.6632589

[24] Carla Penedo, João Pavão, Pedro Nunes, and Luis Custódio. 2004. RoboCup Advanced 3D Monitor. In *RoboCup 2003: Robot Soccer World Cup VII*, Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 637–644.

[25] Ivan Perez and Henrik Nilsson. 2017. Testing and Debugging Functional Reactive Programming. *Proc. ACM Program. Lang.* 1, ICFP, Article 2 (aug 2017), 27 pages. https://doi.org/10.1145/3110246

[26] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[27] Cristiano Politowski, Yann-Gaël Guéhéneuc, and Fabio Petrillo. 2022. Towards Automated Video Game Testing: Still a Long Way to Go.

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

[29] Stefan Ramson and Robert Hirschfeld. 2017. Active Expressions: Basic Building Blocks for Reactive Programming. *The Art, Science, and Engineering of Programming* 1, 2 (apr 2017). https://doi.org/10.22152/programming-journal.org/2017/1/12

[30] Guido Salvaneschi and Mira Mezini. 2016. Debugging for reactive programming. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 796–807.

[31] Brendan Sinclair. 2015. Gaming will hit $91.5 billion this year. https://www.gamesindustry.biz/gaming-will-hit-usd91-5-billion-this-year-newzoo (accessed on January 11th, 2023).

[32] Gregory Tassey. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology* (2002).

[33] Andrew Truelove, Eduardo Santana de Almeida, and Iftekhar Ahmed. 2021. 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)*. 736–747. https://doi.org/10.1109/ICSE43902.2021.00073

[34] Simon Varvaressos, Kim Lavoie, Sébastien Gaboury, and Sylvain Hallé. 2017. Automated Bug Finding in Video Games: A Case Study for Runtime Monitoring. *Comput. Entertain.* 15, 1, Article 1 (mar 2017), 28 pages. https://doi.org/10.1145/2700529

[35] Carolina Veloso and Rui Prada. 2021. Validating the plot of Interactive Narrative games. In *2021 IEEE Conference on Games (CoG)*. 01–08. https://doi.org/10.1109/CoG52621.2021.9618897

[36] Suhas Virmani, Yatin Kanetkar, Manish Mehta, Santiago Ontañón, and Ashwin Ram. 2021. An Intelligent IDE for Behavior Authoring in Real-Time Strategy Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 4, 1 (Sep. 2021), 209–214. https://doi.org/10.1609/aiide.v4i1.18699

[37] Benedict Wilkins, Chris Watkins, and Kostas Stathis. 2020. A Metric Learning Approach to Anomaly Detection in Video Games. In *2020 IEEE Conference on Games (CoG)*. IEEE, 604–607.