

Deriving and Evaluating a Detailed Taxonomy of Game Bugs

Nigar Azhar Butt^{*†1}, Salman Sherin^{‡2}, Muhammad Uzair Khan^{§1}, Atif Aftab Jilani^{¶1}, and Muhammad Zohaib Iqbal^{||2}

¹Department of Software Engineering, National University of Computer and Emerging Sciences, Islamabad, Pakistan

²QUEST, Islamabad, Pakistan

Abstract

Game development has become an extremely competitive multi-billion-dollar industry. Many games fail even after years of development efforts because of game-breaking bugs that disrupt the game-play and ruin the player experience. The goal of this work is to provide a bug taxonomy for games that will help game developers in developing bug-resistant games, game testers in designing and executing fault-finding test cases, and researchers in evaluating game testing approaches. For this purpose, we performed a Multivocal Literature Review (MLR) by analyzing 436 sources, out of which 189 (78 academic and 111 grey) sources reporting bugs encountered in the game development industry were selected for analysis. We validate the proposed taxonomy by conducting a survey involving different game industry practitioners. The MLR allowed us to finalize a detailed taxonomy of 63 game bug categories in end-user perspective including eight first-tier categories: Gaming Balance, Implementation Response, Network, Sound, Temporal, Unexpected Crash, Navigational, and Non-Temporal faults. We observed that manual approaches towards game testing are still widely used. Only one of the approaches targets sound bugs whereas game balancing and how to incorporate machine learning in game testing is trending in the recent literature. Most of the game testing techniques are specialized and dependent on specific platforms.

Keywords— Game bugs, Software testing, Fault Taxonomy, MLR, Postmortem analysis

1 Introduction

Game development is a multi-billion-dollar industry, with approximate revenue of \$162.32 billion annually, and is expected to reach \$256.97 billion by 2025 [1]. Due to improvements in the computational power available on mobile devices, desktop systems, and gaming consoles, there has been a significant increase in the number of people who play games, leading to an increase in the number of games being developed. Consequently, the game development market is now highly competitive.

^{*}Email: i212842@nu.edu.pk

[†]Corresponding author

[‡]Email: salman.sherin@questlab.pk

[§]Email: uzair.khan@nu.edu.pk

[¶]Email: atif.jilani@nu.edu.pk

^{||}Email: zohaib.iqbal@questlab.pk

Game-breaking bugs that disrupt the game-play have caused many games to fail commercially. One such example is the retraction of *Cyberpunk 2077*¹ from the PlayStation Store due to its poor quality despite seven years of development and millions of dollars of investment [2]. According to Polish Business Insider, *CD Projekt Red*'s stock value fell by more than 75% after the game was launched².

Automated testing techniques have significantly improved in the last 30 years, with various innovative techniques being published. However, the most pervasive way of testing games in the industry is through manual playtesting of simple to complex scenarios on the game to test its functionality [3]. The industry relies upon intrinsic knowledge accumulated by the playtesters to make the testing process fruitful [2]. Issues in the testing process are considered to be one of the major causes of game failures [4]. These include disorganized testing in an already constrained testing budget [3], lack of predefined goals for testing and misunderstanding the types of bugs that may appear.

Games are highly interactive software having a very dynamic development lifecycle and cross-cutting dependencies. There are rapid changes and updates involved which make them susceptible to gameplay errors. These gameplay errors are not necessarily software bugs in terms of the incorrect function value. Nonetheless, they have the potential to not only disrupt the gameplay experience but also result in revenue losses for game developers. Such issues must be identified and resolved.

Taxonomies in general have been used to provide a systematic method for increasing understanding of a diverse data. The fault taxonomies are well-known means of collecting and classifying the potential bugs in application domains. Bug taxonomies categorize software bugs based on characteristics like cause, severity, etc., and help in organizing and prioritizing bugs for efficient testing. The structured categorization also identifies patterns and common causes, which helps prevent similar bugs in the future. Thus, bug taxonomies and testing are interconnected, with the former providing a framework for the latter to improve the testing process [5]. Bug taxonomies are a way to guide testers by providing high-level goals to generate new testcases [6] and achieve greater coverage for the system under test [7]. Game designers can benefit from these organized classifications by ensuring that the identified faults do not occur in their games. Test engineers use these to improve their test cases by ensuring that the test cases specifically test for the identified faults [5]. Researchers gain guidance from the taxonomies in designing their test strategies [6].

Similarly, they can act as a warning for the developers to be aware of such faults in the first place. For researchers, bug taxonomies not only provide a method of validating proposed testing techniques by clearly stating types of prevalent bugs to test against [7]; but also open new avenues of research in designing mutation operators corresponding to a particular taxonomy category [8], improvements in existing testing approaches and proposing new techniques [7].

In the game development and testing domain, taxonomies have been proposed for bug identification as well as the specification of vocabulary needed to explain the differences in failures that occur in games due to ambiguity in design [9]. However, the bug taxonomy proposed by Lewis et al. [7] is limited to temporal and non-temporal implementation faults. It fails to cover the plethora of bugs that fall under navigation, sound, crash, gaming balance, and network faults.

In this paper, we present an updated and detailed taxonomy of game bugs which extends the taxonomy of game failures by Lewis et al. [7] based on a Multi-vocal Literature Review (MLR) to incorporate the perspective of both researchers and industry practitioners. Typically, MLR's are used when the topic under focus has a distinct industrial relevance [10]. We analyzed 436 sources in detail – including academic literature (143), postmortems (200), blogs and articles (43), and videos (50). Out of 436, we finally selected 189 sources i.e. academic literature (70), postmortems (62), blogs and articles (20), videos (24), and talk sessions (5) for reported bugs encountered in the games. The selected sources assist in the identification and classification of commonly appearing bugs that result in game failures even after launch. We have conducted a survey of game players and gaming industry professionals to validate that the proposed taxonomy can aid industry practitioners, such as developers and testers, and researchers by providing guidance for playtesting and for validation of new testing techniques [7].

The paper is structured as follows: Section 2 describes the multi-vocal literature review method, that we used for the selection of resources. Section 2.3 shows the process of deriving, evolving, and verifying our

¹<https://www.cyberpunk.net/>

²<https://www.gizchina.com/2022/07/19/cyberpunk-2077-makes-cd-projekt-lose-75-of-its-market-value/>

detailed taxonomy. Section 3 presents our taxonomy. Section 4 describes the method of validation of our taxonomy. Section 5, provides a comprehensive discussion regarding our taxonomy. Section 6 explain the threats to the validity of our study. Section 7, briefly describes related literature. 8 concludes the paper with future work.

2 Systematic Multivocal Literature Review Method

In order to ensure the thoroughness of our proposed taxonomy, we analyzed both white and grey literature. White literature includes peer-reviewed workshop, conference and journal studies [11]. Grey Literature consists of books, book chapters, government reports, news articles, annual reports, presentations, wiki articles, videos, blogs, thesis documents, emails and tweets, and letters [11]. We performed the search for academic and grey literature till **2023/01/04** and considered the following:

1. *Academic Literature*, including both white and grey literature studies such as:
 - (a) peer-reviewed journal, conference, and workshop studies
 - (b) non-peer-reviewed thesis, magazine articles, and book chapters.
2. *Grey Literature*, including:
 - (a) Game development postmortems,
 - (b) Session talks and presentations from well-known game development conferences
 - (c) Articles and blogs.
 - (d) Videos

The goal for the SLR of Academic Literature was to find the game bugs and implementation faults that are currently being identified by the existing game testing techniques. We then performed a systematic survey of the grey literature to verify the identified game bug categories. We decided to opt for game testing approaches to identify game bug categories because testing approaches focus on implementation and functional faults including anomalies. Studies on game development highlight faults in the game development process rather than implementation faults. Similarly, game-play approaches do not focus on bug identification rather they highlight useability challenges as well as score maximization and goal completion aspects of the game-play [12]. The complete dataset is available at <https://doi.org/10.5281/zenodo.8425685>.

2.1 Academic Literature

We used six sources of academic literature: Google Scholar³, IEEE Explore⁴, Springer⁵, ACM⁶, Science Direct⁷, and Wiley⁸. The following generic query was modified to be used in different digital libraries. Our objective was to make our query more inclusive to prevent the possibility of overlooking critical faults.

((“approach” OR “technique” OR “method”) AND (“video game”) AND (“verify” OR “verification” OR “playtest*” OR “test*” OR “debug*” OR “fault find*” OR “fault detect*” OR “anomaly detect*” OR “glitch detect*” OR “glitch find*”))

³<https://scholar.google.com/>

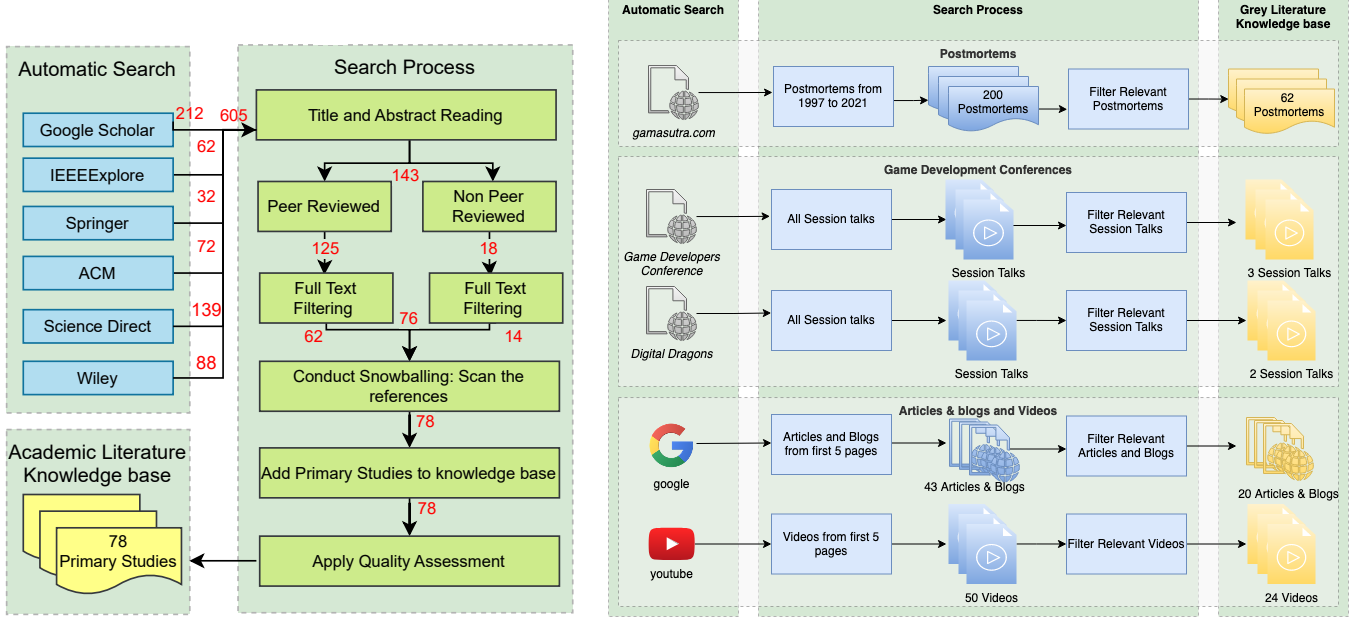
⁴<https://ieeexplore.ieee.org/>

⁵<https://link.springer.com/>

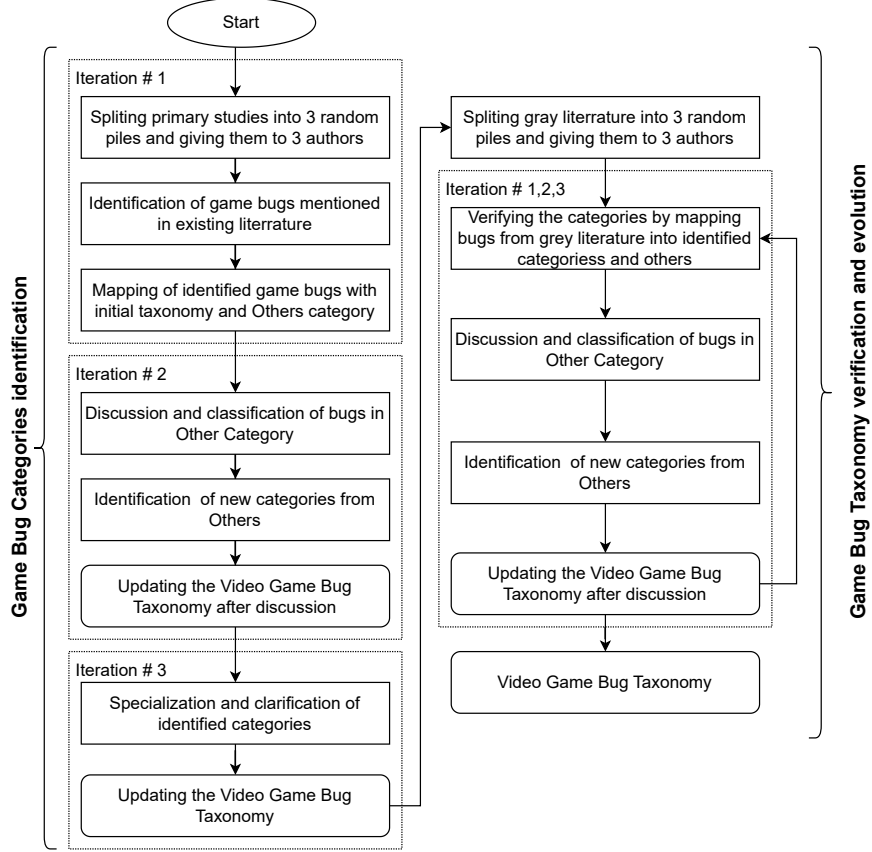
⁶<https://www.acm.org/>

⁷<https://www.sciencedirect.com/>

⁸<https://onlinelibrary.wiley.com/>



(a) The methodology of systematic review of academic literature. (b) The search process of the systematic survey of grey literature.



(c) The process of derivation of the detailed bug taxonomy for games is depicted.

Figure 1: Illustrates the phases of the search process and the number of primary studies in each phase along with the process of deriving the detailed bug taxonomy.

Figure 1a illustrates the phases of the search process and the number of studies in each phase of the SLR of Academic Literature. We adapted our search process from [13]. The preliminary search of digital libraries yielded a total of **605 studies** (Google Scholar(212), IEEE Explore(62), Springer(32), ACM(72), Science Direct(139), and Wiley(88)). The *inclusion* and *exclusion* criteria for the selection of primary studies is described as follows:

- **Inclusion Criteria for Academic Literature:**

- IC1 Studies must be about game testing and fault finding.
- IC2 Studies must discuss a functional game testing approach.
- IC3 Full text must be available.
- IC4 Studies must be in English.

- **Exclusion Criteria for Academic Literature:**

- EC1 Studies only focused on automated gameplay.
- EC2 Studies that have a relevant extension.
- EC3 Studies simply re-applying an existing technique.
- EC4 Studies verifying game design at the design phase.
- EC5 Studies that provide an approach or platform to observe and analyze data for game testing rather than a game testing approach itself.

2.1.1 Filtering: Title and Abstract Reading

We read through the title and abstract of each of **605 studies** extracted as a result of the initial query. We filtered **177** studies, including **122** from Google Scholar, **34** from IEEE Explore, **7** from Springer, **21** from ACM, **4** from Science Direct, and **2** from Wiley digital library. Studies like [14], for which fulltext was not available were filtered out at this step as well. From the remaining relevant studies, we filtered duplications and finally ended with **143** relevant studies.

2.1.2 Separating Peer-reviewed and non-peer-reviewed Studies

We divided the **143** relevant studies into peer-reviewed and non-peer-reviewed categories. We had **125** peer-reviewed studies that were published in journals (28), conferences (80), workshops (10), and symposiums (7). And **18** non-peer-reviewed studies which included thesis (8), non-peer-reviewed articles (9), and magazine article (1).

2.1.3 Filtering: Full-Text Reading

We read and analyzed the full text of the studies we had previously filtered and comprehensively applied the inclusion and exclusion criteria. All studies that fulfilled the inclusion criteria were included. Since our focus was on implementation faults, we reviewed and analyzed literature that focused on testing of functional aspects of games. Studies like [15] were excluded based on EC1. AI-based automated game-playing is a complete research domain in itself. The focus of such studies is the successful completion of a pre-defined game scenario or maximizing the score. While such approaches can be used for triggering bugs, their inherent focus is not bug identification and were therefore excluded. Studies like [16] which have a comprehensive extension like [S7], were excluded based on EC2, since including both would have resulted in duplication. Similarly, studies like [17], were excluded despite being the most recent version because [S7] contains the details of test sequence generation, modeling, and implementation fault identification that is congruent to our analysis. Studies that do not present a novel approach but rather report on using an existing method like [18] were excluded based on EC3. Game design verification can be considered game testing at the design phase; however, such studies do not focus on implementation faults or game bugs that appear in the end product during a later stage of the game development lifecycle. Hence, such studies [19]

were excluded based on EC4. Studies that present an approach or platform to observe and analyze data for game testing rather than a game testing approach itself like [20] were excluded based on EC5. We only included those studies in our academic literature, the knowledge base of primary studies, that fulfilled the inclusion criteria. We obtained **76 studies** including both peer-reviewed (62) and non-peer-reviewed (14) studies. Since the goal of our study is to devise a comprehensive taxonomy of game bugs, we have included non-peer-reviewed literature to ensure the completeness of the proposed taxonomy.

2.1.4 Snowballing

We performed the backward and forward snowballing on the 76 studies. We analyzed the title and abstract of each study again to verify it’s relevance. We found two more relevant studies during snowballing which we then added to our knowledge base. The final dataset has **78 studies** including peer-reviewed (**64**) ([S1], [S2], [S3], [S4], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S21], [S22], [S23], [S24], [S25], [S26], [S27], [S28], [S29], [S30], [S31], [S32], [S33], [S34], [S35], [S36], [S37], [S38], [S39], [S40], [S41], [S42], [S43], [S44], [S45], [S46], [S47], [S48], [S49], [S50], [S51], [S52], [S53], [S54], [S55], [S56], [S57], [S58], [S59], [S60], [S61], [S62], [S63], [S64]) and non-peer-reviewed (**14**) studies ([S65], [S66], [S67], [S68], [S69], [S70], [S71], [S72], [S73], [S74], [S75], [S76], [S77], [S78]).

2.1.5 Quality Assessment

In SLR’s, quality assessment of selected studies is usually done for study selection, weighing each study, highlighting differences in quality of the studies, or to build confidence in results of the review paper [21]. We have used quality assessment to weigh each individual study and to provide researchers and practitioners, confidence in the results and conclusions. We took inspiration for quality assessment criteria from [22]. The following questions were used to assess the quality of our selected primary studies:

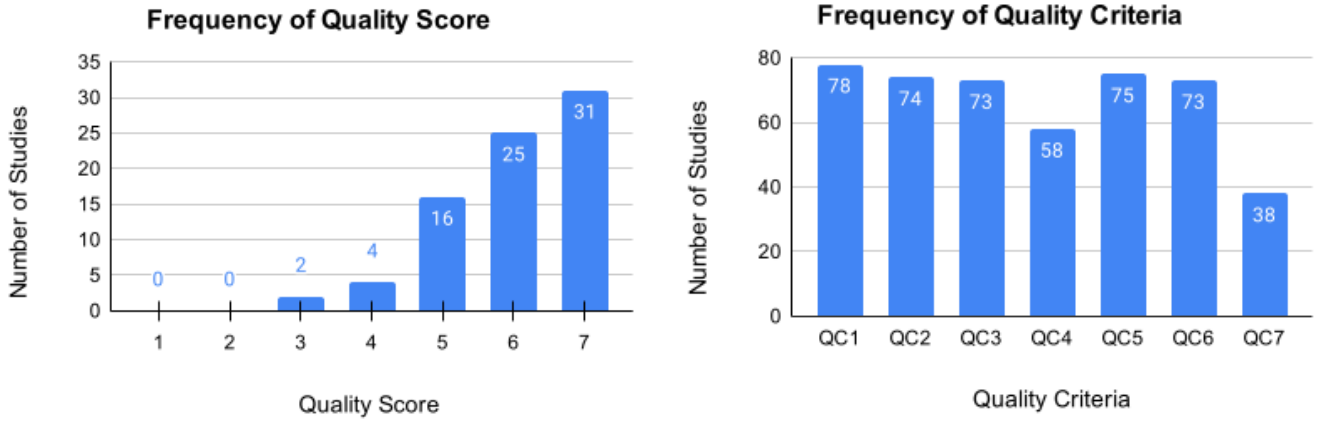
- QC1 Is/are the research objective/s clearly identified?
- QC2 Was the test generation method clearly explained?
- QC3 Was the test execution method clearly explained?
- QC4 Was the game bug detection mechanism clearly explained?
- QC5 Was the proposed approach demonstrated using a case study?
- QC6 Were game bugs discussed?
- QC7 Were the identified bugs described with an example?

These questions were answered either “yes” or “no”, rated as one or zero, respectively. The sum of the scores for all these questions was used to assess the quality of a primary study. We, however, did not exclude any study based on its quality score; rather, this score just depicts the quality rank of the primary studies included in this study. We observed that most of our selected studies scored well on the overall quality assessment criteria. Figure 2a illustrates that 72 studies had quality scores greater than or equal to 5. We also observed that over half of the selected studies (40 studies) failed on QC7 and 20 studies for QC4 (figure 2b). The detailed analysis of quality assessment is shown in table A.1 of appendix A.

We analyzed our primary studies in detail and compared the identified game bugs with the existing taxonomy [7]. We found that it failed to cover all the bugs and hence we developed our updated game bug taxonomy and verified it further with the help of grey literature.

2.2 Grey Literature

The SLR of academic literature allowed us to identify the game bugs and implementation faults that are currently being identified by the existing game testing techniques. We then performed a systematic survey of the grey literature to not only verify whether the identified game bug categories are present in popular games and influence the gameplay experience of gamers, but also to add the perspective of industry professionals to our taxonomy. Hence, we modified the inclusion and exclusion criteria for the categories of



(a) Illustrates frequency of Quality Score for the selected studies (b) Illustrates how many selected studies fulfilled each Quality Criteria.

Figure 2: Illustrates Quality Assessment results of selected studies.

reviewed grey literature. Figure 1b illustrates the phases of the search process and the number of primary studies in each phase of the systematic survey of grey literature. The *inclusion* and *exclusion* criteria for the selection of primary web sources is described as follows:

- **Inclusion Criteria for Grey Literature:**

IC5 The source must be about bug identification in a game.

IC6 Source is about an experience regarding an encounter with a bug in a game.

IC7 Source must be available.

IC8 Source must be in English.

- **Exclusion Criteria for Grey Literature:**

EC9 Sources in a language other than English.

EC10 Sources describing a game without mentioning an example of game bugs.

EC11 Sources only describing a duplicate bug.

EC12 Sources describing design flaws at the design phase.

2.2.1 Postmortems

We read the postmortems available at the Gamasutra website⁹. It is a successor of the Game Developers Magazine and is currently considered to be the most complete resource for digital-game postmortems [23]. After applying the inclusion and exclusion criteria we selected 62 relevant postmortems ([PM1], [PM2], [PM3], [PM4], [PM5], [PM6], [PM7], [PM8], [PM9], [PM10], [PM11], [PM12], [PM13], [PM14], [PM15], [PM16], [PM17], [PM18], [PM19], [PM20], [PM21], [PM22], [PM23], [PM24], [PM25], [PM26], [PM27], [PM28], [PM29], [PM30], [PM31], [PM32], [PM33], [PM34], [PM35], [PM36], [PM37], [PM38], [PM39], [PM40], [PM41], [PM42], [PM43], [PM44], [PM45], [PM46], [PM47], [PM48], [PM49], [PM50], [PM51], [PM52], [PM53], [PM54], [PM55], [PM56], [PM57], [PM58], [PM59], [PM60], [PM61], [PM62]) that discuss implementation faults in games whether they were pre-release or post-release.

⁹<http://www.gamasutra.com>

2.2.2 Game Development Conferences

We searched the talk sessions and presentations regarding fault-finding in specialized conferences on game development like Digital Dragons¹⁰ and GDC¹¹ [2]. The five relevant talks are from GDC ([T1], [T2], [T3]), and Digital Dragons ([T4], [T5]) that we included in our grey literature knowledge base. We read transcripts and watched the presentations; and summarized the points related to implementation faults and used them to verify and evolve our taxonomy.

2.2.3 Web Articles and Blogs

We queried Google Search to find articles about bugs in games on websites and blogs using keywords.

**“video game” AND (“glitch*” OR “fault*” OR “bug*” OR “flaw*”) AND
“game-breaking”.**

Our focus was to identify the game critical bugs that make it very difficult for gamers to play the games. To make the search as systematic as possible we followed the steps by Bajwa et al. [24] and used the Google search in Google Chrome internet browser. We performed the following steps before running the query:

- i Signed out from google¹².
- ii Cleared the browser search history.
- iii Cleared out the browser cache.
- iv Disabled the instant search predictions option from google.
- v Enabled the ‘100 results/links per page’ in browser settings.
- vi Added the plugin SEOquake¹³ to the browser.

We selected 20 most relevant articles after applying exclusion and inclusion criteria including seven news articles (A), nine blogs (B), two wiki articles (W), and two forum articles (F) ([W1], [W2], [W3], [W4], [W5], [W6], [W7], [W8], [W9], [W10], [W11], [W12], [W13], [W14], [W15], [W16], [W17], [W18], [W19], [W20]). We read the complete articles and summarized the points related to game bugs and used them to verify and evolve our taxonomy.

2.2.4 Videos

We searched to find YouTube videos about game bugs using the following keywords:

**“video game” AND (“glitch*” OR “fault*” OR “bug*” OR “flaw*”) AND
“game-breaking”.**

YouTube provides the game community with context-rich bug information and is a great resource for bug identification [25]. Our focus was to identify the game-critical bugs that make it difficult for gamers to play games. To make the search as systematic as possible, we took inspiration from Bajwa et al. [24] and used the Google Search in the Google Chrome internet browser. We performed the steps described in the previous subsection to run the query.

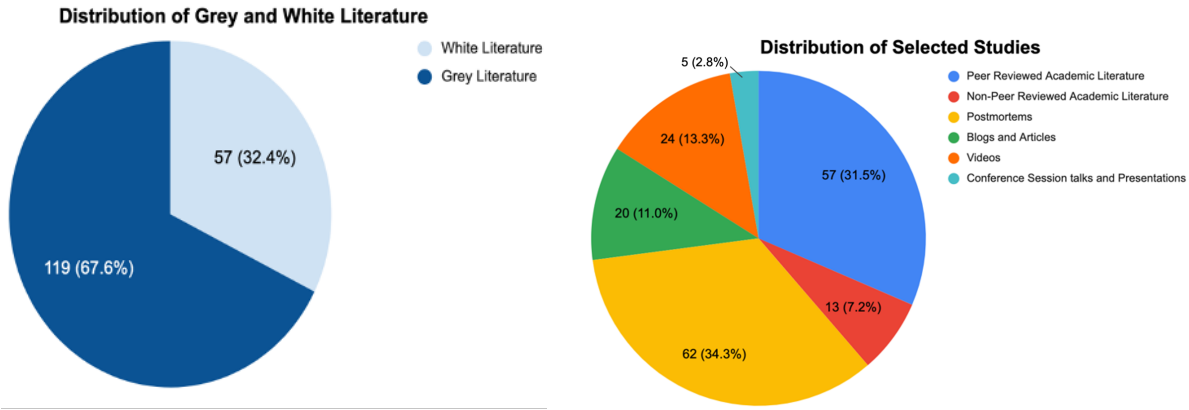
The 24 most relevant videos were selected ([V1], [V2], [V3], [V4], [V5], [V6], [V7], [V8], [V9], [V10], [V11], [V12], [V13], [V14], [V15], [V16], [V17], [V18], [V19], [V20], [V21], [V22], [V23], [V24]) after applying exclusion and inclusion criteria over 50 retrieved videos. We watched the complete videos having a total of 20 hours of watch time. We then summarized the points related to game bugs and used them to verify and evolve our taxonomy.

¹⁰<http://digitaldragons.pl/>

¹¹<https://www.gdconf.com/>

¹²<https://www.google.com/>

¹³<https://www.seoquake.com/index.html>



(a) Illustrates the distribution of grey and white literature. (b) Illustrates the different types of selected sources.

Figure 3: Overview of the distribution of selected sources.

2.3 Deriving the Detailed Bug Taxonomy

In this subsection, the process of derivation of the detailed bug taxonomy for games from the selected sources is described in detail as shown in figure 1c. Figure 3 illustrates the distribution of selected studies in terms of white and grey literature (figure 3a) and different types such as academic, postmortems, blogs, videos, and conference sessions (figure 3b).

2.3.1 Game Bug Categories identification

Once we had our primary studies from academic literature, we analyzed them in detail and compared the game bugs with the existing game bug taxonomy [7]. We found that it failed to cover all the bugs and hence we developed our updated taxonomy and verified it further with the help of grey literature. This process was performed with the consensus of all authors. The selected academic literature was randomly divided into three sets of equal size for data extraction and classification. We performed multiple iterations to identify game bug categories.

In the first iteration, we started by extracting game bugs identified in each study and classifying them into the main categories of the initial taxonomy. The initial taxonomy consisted of the existing game bug taxonomy [7] (depicted by the underlined categories in figure 5) as well as two new main categories Network Faults (subsection 3.3) and Gaming Balance Faults (subsection 3.1), which we identified during the preliminary analysis of the domain. Any bugs that did not fall into these categories were classified into the Others category with additional comments.

In the second iteration, we discussed the bugs classified into the Others category and identified new categories. Examples of such categories are Unexpected Crash (subsection 3.6) and Navigational Bugs (subsection 3.7).

In the third iteration, we had discussions regarding existing categories and subcategories to finalize whether the subcategories were under the correct main categories. The Implementation Response Issues category was modified. In [7], it was presented as a sub-category of Temporal faults and included scenarios in which game response speed did not match the expected speed. However, we converted it into a main-tier category. We observed that faulty system response was not only restricted to the speed of required response rather non-temporal elements were also involved such as faulty collision detection or reward estimation. Hence, it was decided that Implementation Response Faults will be a main-tier category.

Lastly, we had discussions regarding the specialization of the categories based on examples identified from the selected studies. The seventh non-Temporal fault sub-category is Invalid Graphical Representation (subsection 3.8.7). It is one of the categories that we modified, which was originally proposed by Lewis et al. [7]. We extended it into further sub-categories, which include Abnormal Text, Corrupted Frame, Extra Game Asset, Incorrect Visual State Transition, Missing Game Asset, and Slow loading. Figure 4, shows

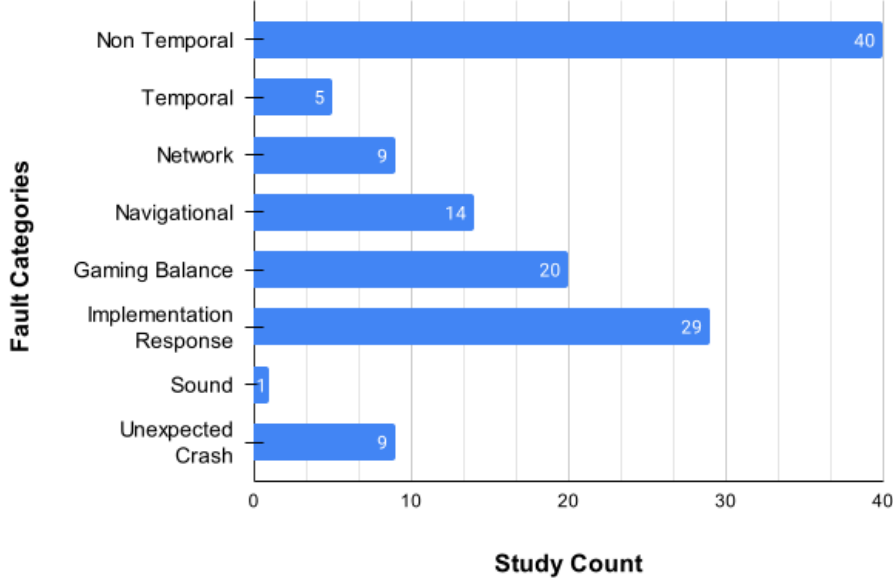


Figure 4: Distribution of Game bugs in academic literature.

the distribution of game bugs in academic literature. Once we had the draft of our taxonomy, we further verified and evolved it with the help of grey literature. The detailed classification of selected studies with respect to game bug categories is available in *Video Game Bug Taxonomy MLR-step6-Academic Literature-Taxonomy* file of our dataset [26] as well as in table B.1 of appendix B.

2.3.2 Game Bug Taxonomy verification and evolution

Once we had compiled the selection of grey literature into our knowledge base, we analyzed them in detail and identified and classified the game bugs into different categories. For this purpose, we followed the same process as done for academic literature. The selected grey literature was randomly divided into three sets of equal size for data extraction and classification. We performed multiple iterations to identify game bug categories. In the first iteration, we simply classified the bugs into an Others category if they did not fall into an existing category. Then we had discussions regarding bugs placed in the Others category. It was debated among authors whether to place it into an existing category or create a new one. We observed a lack of work done in the identification of Sound Faults (subsection 3.4) in games in the academic literature. We had only encountered one such study [S58], and hence had placed such faults in the Others category. However, our extensive survey of game postmortems and grey literature highlighted the presence of faults related to game sounds. We repeated this process of classification and discussion for three iterations until all authors were satisfied with the taxonomy categories. The dataset [26] provides the final iteration of the classification of the game bugs identified in grey literature into categories identified in the game bug taxonomy. The table B.1 in appendix B shows the list of sources from where each of the main-tier bug categories were extracted and verified.

Our proposed taxonomy is shown in figure 5. It considers a total of 63 categories. These include the 20 categories proposed by Lewis et al. [7], of which we have modified five categories via the creation of new subcategories and one category (subsection 3.5.5) by extending its definition.

3 Taxonomy

In this section, we will go over the taxonomy in depth. The focus of our taxonomy is **Implementation Faults** that occur during game execution. This starts from the time the game application is launched, or the game player logs in until the moment the game application is shut down or the game player logs out of the game application. In accordance with ISO 24765 (2017), a **fault** occurs when the system behaves in a

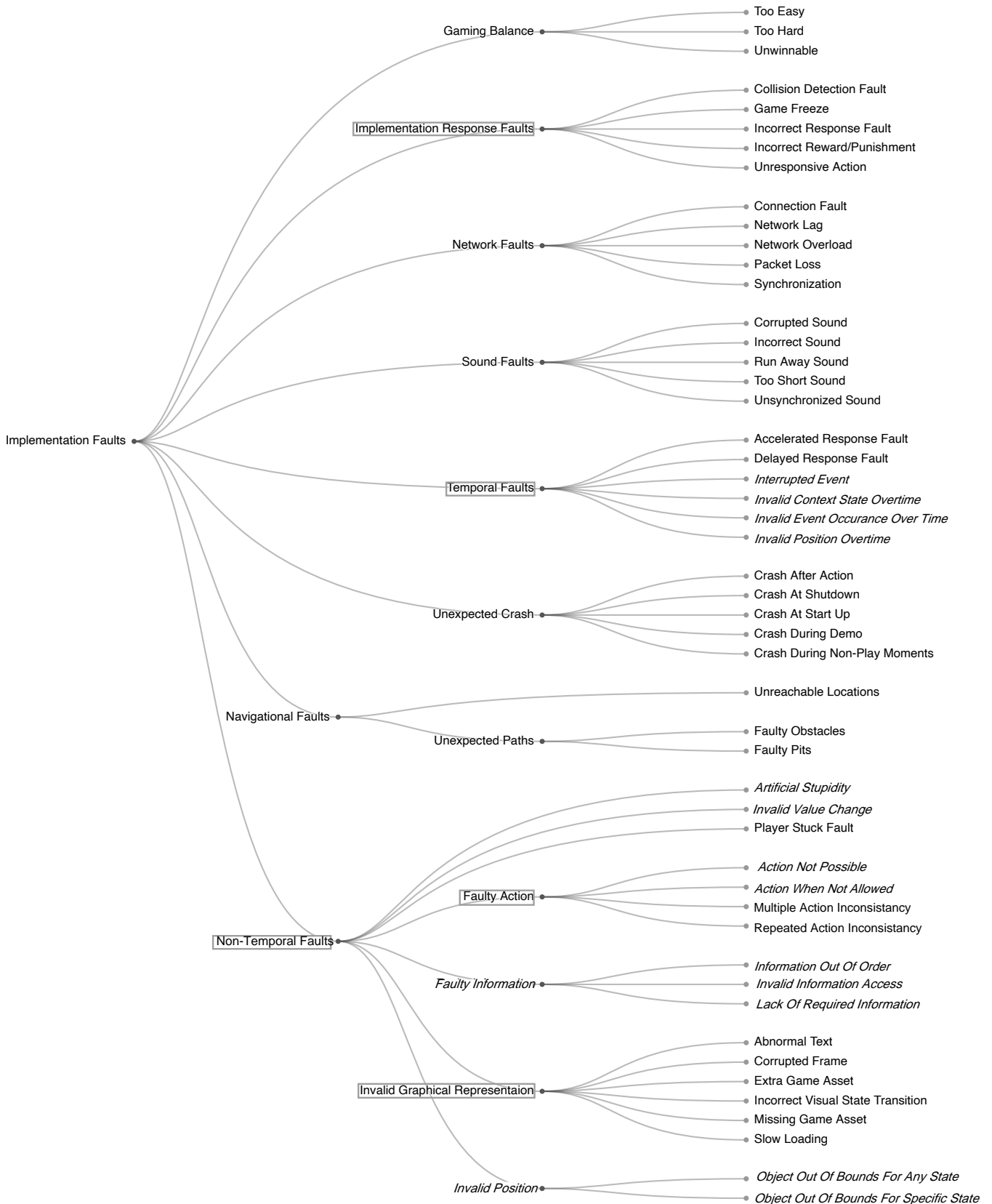


Figure 5: The proposed Detailed Taxonomy of **63** Game Bug categories including the **five** categories enclosed in a box, that have been modified from existing literature while the italicized **15** categories are used without modification from existing literature.

manner that is anomalous or inconsistent with the requirements or specifications [27]. It is also called a bug or a defect. The taxonomy has been designed from the end-user perspective and is based on user-observable behavior that is anomalous or inconsistent with user expectations.

Implementation Faults are further divided at first-tier into eight categories: Gaming Balance, Implementation Response, Network, Sound, Temporal, Unexpected Crash, Navigational, and Non-Temporal faults. Table B.2 in appendix B provides brief and concise descriptions with examples of the specialized new game bug categories from the taxonomy. We have also provided a video resource¹⁴ to better help in understanding tricky game bug categories with visual examples. The categories present in our taxonomy are described in detail in the following sub-sections.

3.1 Gaming Balance

A balanced game is one in which the players that have equivalent or similar skillsets have an equal chance of winning, i.e., whether it is player-vs-player (PvP) or player-vs-environment (PvE). Additionally, this includes a gamer’s ability to complete the game while matching its expected difficulty level, which means that the player should neither find it too easy to play nor too difficult. While such terms are incredibly subjective, this category refers to an excessive advantage or disadvantage to the player playing the game. Fairness is a universally sought-after attribute in games by players [28]. We consider it to be an implementation fault because the system is behaving in a manner inconsistent with design expectations. In total 20 of 78 selected academic studies on game testing approaches focus on such scenarios.

This category specifically refers to unbalanced gameplay and is further divided into three sub-categories. It is a fault such that a player finds the game either too easy that it becomes boring and repetitive; or too hard that it triggers the desire to give up; or unwinnable that regardless of the player’s effort the game is impossible to beat.

3.1.1 Too Easy

This category includes gameplay faults due to which players find it too easy to play a game. It is critical if a particular set of parameters leads to unfair advantages in multiplayer games [V7]. If a game level is easier than player expectations, it is placed in this category [PM25].

Additionally, this category includes bugs that make a game scenario easy to complete. For example, season 11 of *League of Legends*¹⁵ contains a bug that allowed players to kill enemies including high-level bosses in a single attack with a hundred percent probability [V16].

3.1.2 Too Hard

This category includes gameplay Faults such that players find it too difficult to play a game. This includes difficulty in finishing a task due to obscure design, ambiguous task requirements, or unreasonable constraints. For example, the counter-intuitive structure of the first level in *Frozen Synapse*¹⁶ resulted in many players requiring an undesirably long amount of time to beat it [PM26].

3.1.3 Unwinnable

This category includes gameplay faults such as players being unable to finish the game especially if it is due to an overly competent AI, convoluted game logic, game constraints such as small countdown timers, or impossible to obtain in-game items. Being unwinnable renders the game unplayable. In the game *Political Machine*¹⁷, the AI opponent is too competent. When the players got up to George Washington, the opponent AI won every state making it impossible for the players to advance any further in the game [PM51].

¹⁴<https://bit.ly/vgbtvideo>

¹⁵<https://www.leagueoflegends.com/>

¹⁶https://store.steampowered.com/app/98200/Frozen_Synapse/

¹⁷<https://www.politicalmachine.com/>

3.2 Implementation Response Faults

These types of faults were originally presented in the context of temporal faults and included scenarios where game response speed did not match the expected speed [7]. We observed that faulty system response was not restricted to the speed of required response, but also faulty responses to player actions.

We extended this category by adding five sub-categories into it including Collision Detection, Game Freeze, Incorrect Response, Incorrect Reward or Punishment, and Unresponsive Action. They are described in the following sub-categories.

3.2.1 Collision Detection Fault

Collision is a very common element in games. Its effects vary from interaction with benign components of the environment which at most block the path to helpful components to the dangerous components that can ‘kill’ the player.

Generally, in a video game, the collision of a player avatar with in-game elements or in-game elements with each other [PM12] has associated functionality that depends on correct and timely detection of that interaction. For example, if the game *Mario Bros.*¹⁸ is working correctly then, Mario stops when his collision is detected with the pipe and ‘dies’ when his collision is detected with the Goomba. In [S2], an open-source Mario game is seeded with a faulty collision detection so, Mario could pass through the Goomba without any damage.

3.2.2 Game Freeze

This fault occurs when the game becomes unresponsive. It can be complete unresponsiveness such that the game screen remains unchanged or partial unresponsiveness such that the player can change camera angles or view different aspects of the game but there is no response to any in-game actions. Such a fault may occur when the player causes the game to enter an abnormal state (section 3.6).

For example in *Fortnite*¹⁹, in the ‘build-a-brella’ steps after the player has completed customization and clicked purchase battle pass, the game stopped responding and the only way to make it work was to restart the game [V6]. There are rare cases in multi-player games, in which a player can cause another player’s game to freeze or crash. In *Tony Hawks Underground 2*²⁰, players could cause other players’ games to freeze by sending them messages longer than the restricted limit [W1].

3.2.3 Incorrect Response Fault

In-game actions have corresponding expected responses. When the response to an action is contrary to expectation then it is considered faulty such as a button that is expected to open a door instead closes it. *Final Fantasy V*²¹ had a weapon (the Chicken Knife), that would sometimes make the player run away from battle instead of doing an attack [W2].

3.2.4 Incorrect Reward/Punishment Fault

Certain actions within a game result in a reward that either increases score, game-specific attributes, or gifts items; or a punishment that deducts game-specific attributes or loss of items [28]. This fault occurs when an in-game action results in less or more reward compared to expected, or conversely less or more punishment than expected.

Such bugs can take forms like in *Faxanadu*²², the reward for beating dungeons are items that can be used to clear certain blockades or obstacles in certain screens. The rewarded items disappear after being used. If the player leaves the screen, the blockade reappears, but the items needed to clear the blockades

¹⁸<https://supermariobros.io/>

¹⁹<https://www.epicgames.com/fortnite/en-US/home>

²⁰https://tonyhawkgames.fandom.com/wiki/Tony_Hawk's_Underground_2

²¹https://store.steampowered.com/app/1173810/FINAL_FANTASY_V/

²²<https://nintendo.fandom.com/wiki/Faxanadu>

do not respawn, even after beating the dungeons again. Hence there is a lack of reward for an action that has an expected reward [W2].

3.2.5 Unresponsive Action Fault

In-game actions have expected responses. When the action does not result in a response then it is considered faulty such as a button that is expected to open a door does not open the door. It differs from *Incorrect Response* because this category covers a lack of response rather than a response that is different from expected. It also differs from *Game freeze* which involves the complete game and all its actions becoming unresponsive. In this category, only one action becomes unresponsive.

In *Cyberpunk 2077*²³, when a non-playing character (NPC) Takemura calls, the player takes the call but the NPC does not go away on end call action regardless of how many time the action is performed [V21].

3.3 Network Faults

This category includes network faults in online games. The network problems were reported in [29] in the context of online, especially online multiplayer games like *DOTA-2*²⁴. This category encompasses problems that arise due to the host game application's connection and communication with the game server as well as the stable performance of game server during increased or unexpected network traffic. The academic literature focuses mostly on establishment of connection, and load testing; with some work focusing on the detection of lost data packets [29]. However, the survey of the grey literature highlighted network problems that are caused by a lack of bandwidth, latency, and synchronization, especially in Massively Multiplayer Online Role-playing Games (MMORPG). Network problems are difficult to reproduce and isolate even when they are identified because they do not necessarily occur in the game code itself. The Network Faults category is further divided into five sub-categories including Connection Fault [29], Network Lag [29], Network Overload [29], Packet Loss [29], and Synchronization. They are described in proceeding sub-categories.

3.3.1 Connection Fault

The first requirement for playing an online game is creation of a connection to ensure stable communication. It is not limited to the initial connection and login to the game server. In *Diablo 3*²⁵, initial connection was established without any issues, only if a player switched shields with a certain Templar follower, server gave the player an error message and kicked them out of game. Furthermore, the server prevented the player from logging back in [V2].

3.3.2 Network lag

This fault occurs when gameplay experience in online games is impacted by lagging response due to network speed constraints and data packet size constraints. Network Lag can cause a player's in-game actions to be delayed. Consequently, by the time the action is received and executed by the server, the game state has changed. This lag between the game server and the player results in the player's view of the game world being out of date, impeding the player's ability to make proper decisions in the game. In First Person Shooter (FPS) games like *DOTA 2*²⁶, this lag would result in the players' shots missing.

3.3.3 Network Overload

A critical requirement for multiplayer online games is the ability to handle increased traffic and load with stable performance. It is especially necessary for games in the MMORPG genre. Load testing and stress

²³<https://www.cyberpunk.net/>

²⁴<https://www.dota2.com/home>

²⁵<https://kr.diablo3.blizzard.com/en-us/>

²⁶<https://www.dota2.com/home>

testing are great tools for discovering bottlenecks before deployment and prevent situations like those encountered by *Wireless Pets*²⁷. It could not even handle 10,000 users playing it simultaneously before crashing [PM17]. *Diablo II*²⁸ faced bugs that only appeared at much higher usage rates, like more than 100,000 players, to trigger certain buggy situations [PM6].

3.3.4 Packet Loss

Packet loss describes packets of data not reaching their destination after being transmitted across a network. Packet loss is commonly caused by network congestion, hardware issues, software bugs, and several other factors. These types of problems are extremely difficult to reproduce and isolate. Even when they are identified, they are often impossible to fix because they do not occur due to fault in the game code itself. A certain level of packet loss is expected during any type of communication via the Internet. However, if it causes obvious game-play issues like lack of execution of critical player commands; then it should be resolved. During the development of *Toontown*²⁹, a variety of packet loss problems were encountered. Eventually, the development team decided to use Secure Sockets Layer(SSL) to send game data back and forth from the clients, to ensure security as well as to prevent the misinterpretation of game data by various pieces of networking hardware between *Toontown* servers and players on internet [PM10].

3.3.5 Synchronization

Network-based synchronization faults are generally specific to multiplayer online games in which players with or against each other. In such games, game states must be synchronized across multiple players to allow them to play the game. Issues with synchronization can result in the player's view of the game world being out of pace with other players. In *Sins of a Solar Empire: Rebellion*³⁰, there were situations in which players became desynchronized during multiplayer matches. Partway through a match, players would find inconsistencies between game states for different players. A planet might seem to be owned by one player on one machine but owned by a second player on another machine. A battle could be raging in one corner of the galaxy between two players, while other players would see nothing [PM49].

3.4 Sound Fault

Our extensive survey of game postmortems, and grey literature highlighted the presence of sound-related faults in games. Game audio can be split into two separate categories, diegetic and non-diegetic sound. The non-diegetic sounds refer to a game's soundtrack, background ambiance which is not directly generated by or linked to game environment, and narrative commentary like in *Grand Theft Auto IV: The Lost and Damned*³¹, the player is thrown from a car and dies, initiating the 'wasted' sound effect. Alternatively, Diegetic sounds usually refer to a game's sound effects, which are directly generated by game elements, and the dialogue that takes place between characters, like pressing the acceleration button in a racing game leads to accelerating car sound. Sound effects provide important gameplay cues for players. They clarify or reinforce player actions, giving feedback on players' decisions [28], [30]. It is especially critical for rhythm and dance games which require on-spot sound effects and synchronized music [W2]. This category encapsulates faulty in-game sounds. This category is further divided into five sub-categories including Corrupt, Incorrect, Run Away, Too Short, and Unsynchronized sounds. They are described in proceeding sub-categories.

3.4.1 Corrupt sound

These faults involve the in-game sounds including both diegetic and non-diegetic, becoming distorted, warped, pitched, garbled, or crackled. This can occur due to issues with sound settings in the game

²⁷<https://www.gamedeveloper.com/programming/postmortem-games-kitchen-s-i-wireless-pets-i->

²⁸<https://diablo2.blizzard.com/en-us/>

²⁹<https://www.toontownrewritten.com/>

³⁰<https://www.sinsofasolarempire.com/>

³¹https://gta.fandom.com/wiki/The_Lost_and_Damned

application itself, due to faults in the deployment platform as well as audio and video drivers. While the issues with settings in the game application like audio synchronization with different video and framerate per second are comparatively easy to identify, the issues that arise due to the numerous combinations of deployment platforms are not as easy to predict and debug. In *Final Zone II*³², a horrible buzzing sound would sometimes start after the intro cutscene and continue throughout the game [W2].

3.4.2 Incorrect sound

These faults involve the correctness of diegetic sounds. This category encapsulates faults in which the sound behind an animation is incorrect or does not match the circumstances or is completely missing. It can be something obvious like dialog not matching sound or gunfire sounding like water dripping. In *Whacked*³³, missing sound resources led to either sounds missing completely or wrong sounds being played behind animations [PM41].

3.4.3 Run Away sound

When the sound continues even after the animation completes or the sound effect continues to play even after animation or scenario corresponding to the sound ends. It differs from the Unsynchronized sounds category because the sound starts at the right time but it ends too late as opposed to Unsynchronized sounds that may also start with a delay. In *The Italian Job*³⁴, due to a lack of sound resources available, the engine sounds looped very badly [PM35].

3.4.4 Too Short sound

The sound ends before the animation or corresponding scenario completes. Non-diegetic sounds usually work fine, especially if they are just looping WAV files. However, sound effects are extremely difficult to get right if the audio person is not in-house experiencing the creation of the title firsthand. A sound having the wrong length is a serious problem. If the requirement is for a sound to go along with the swooshing pendulum blades, then it should be specified for how long the pendulum is going to swing [PM41]. *Whacked* and *Descent: Freespace*³⁵, both encountered the problem of short sounds [PM41]. It differs from the Unsynchronized sounds category because the sound starts at the right time but it ends too early as opposed to Unsynchronized sounds that may also start before the animation begins.

3.4.5 Unsynchronized sound

This involves sounds that do not synchronize with their corresponding scenarios and animations. These include sounds starting or ending too early; or starting and ending too late; or skipping in between. Such faults are extremely critical in Rhythm and dance games in which players rely on audio cues to play the game. Similarly, in RTS games like *DOTA 2* if the sounds are delayed then it becomes difficult for players to respond to threats in a timely manner. In MMORPG games, the sound synchronization issues can also be due to network problems. *DJMAX Portable Black Squares*³⁶ and *Claziquai Editions*³⁷, background music had a bad habit of skipping and desynchronizing every now and then. In a Rhythm Game, this is a big problem, as it can make the song more difficult to play [W2].

3.5 Temporal

This category includes those faults which require some knowledge of the previous game state to accurately categorize. The game state refers to the complete status of all game elements and their attributes at

³²https://en.wikipedia.org/wiki/Final_Zone_II

³³<https://en.wikipedia.org/wiki/Whacked!>

³⁴[https://en.wikipedia.org/wiki/The_Italian_Job_\(2003_video_game\)](https://en.wikipedia.org/wiki/The_Italian_Job_(2003_video_game))

³⁵https://en.wikipedia.org/wiki/Descent:_FreeSpace_-_The_Great_War

³⁶https://en.wikipedia.org/wiki/DJMax_Portable_Black_Square

³⁷https://en.wikipedia.org/wiki/DJMax_Portable_Claziquai_Edition

a specific moment in time [30]. For example, in Super Mario, Mario jumps up to the right height and continues to hover in mid-air (Invalid position over time). At any single point in time, Mario is at a valid height however, by analyzing the coordinates of Mario over multiple states, the fault can be identified. This category is further divided into six sub-categories in which two sub-categories are new i.e. Delayed Response and Accelerated Response; one sub-category is redefined but originally taken from [7], Invalid Event Occurrence Over time. The remaining three of the subcategories from Lewis et al. [7] are adopted as it is. They are Interrupted Event, Context State Over time, and Invalid Position Over time. They are described in the proceeding sub-sections.

3.5.1 Accelerated Response

The game or player response to or consequence of an action is accelerated giving an unfair advantage or disadvantage, especially in real-time games. In *King's Quest IV*, when Rosella (the character the player controls) is in the ogre's house and must reach the door before he catches her, the ogre travels across the screen too fast for a player to react [W2]. In essence, the response itself is not faulty at any single point in time but due to the accelerated nature of the response as compared to previous game states a fault can be identified.

3.5.2 Delayed Response

The game or player response to or consequence of an action is delayed or halted giving an unfair advantage or disadvantage, especially in real-time games. The response itself is not faulty at any single point in time but due to the delayed nature of the response as compared to previous game states a fault can be identified. This fault is not necessarily caused by the configuration of the game application itself but could be a consequence of network problems resulting in lag in online games; or even limitation of deployment platform in terms of limited RAM etc. In *King's Quest IV*³⁸, at some random intervals the response to Rosella (the character the player controls) and other characters would lag especially in less computationally powerful computers which can be very damaging for players in real-time games [W2].

3.5.3 Interrupted Event

This category includes any event that stopped unexpectedly. These include a sound effect cutting off or an enemy stopping mid-attack for no visible reason. If the enemy stopped because someone or something applied a freeze effect on it then it is not an Interrupted Event because it meets the expectations of the player in the game. The system's in-game response is only faulty because it results in an event termination before its expected time.

3.5.4 Invalid Context State Over time

State in this case refers to the user-observable display of the game rather than game code flags and values. This category includes faults in which a valid in-game element state occurs before or after the expected time; or continues longer than or terminates earlier than the expected time. In *Super Mario*, when Mario catches a star, its image starts flickering and Mario enters an invulnerable state for 20 seconds. If the invincible state lasts less than or more than 20 seconds then it will fall into this category.

3.5.5 Invalid Event Occurrence Over time

This category classifies discrete events that occur too often or too seldom. They include incidents such as firing a gun as discrete events. We have expanded it to include faults in which a valid in-game event occurs before or after expected time; or continues longer than or terminates earlier than expected time. Events in our case include game events such as *Hell Event* in *Lords Mobile*³⁹. They are daily Turf Events which run for 55 minutes and refresh every hour. Players complete corresponding tasks to win points and get

³⁸<https://playclassic.games/games/adventure-dos-games-online/play-kings-quest-iv-perils-rosella-online/>

³⁹<http://lordsmobile.igg.com/>

rewards. If the *Hell Event* starts at the wrong time or terminates before the expected 55 minutes, then it will fall into this category.

3.5.6 Invalid Position Over time

This category includes faults in which an in-game element retains a position that is valid in any single point in time but invalid or faulty in context with previous states. In other words, the element is in a valid position for less than or more than a valid amount of time. In *Super Mario*, if Mario jumps up to the right height and continues to hover in mid-air. At any single point in time Mario is at a valid height however, by analyzing the coordinates of Mario over multiple state, the fault can be identified.

3.6 Unexpected Crash

It is a new category added in the updated taxonomy. This type of fault causes the entire game application to crash and exit [S6]. The common causes for Unexpected Crash to occur are memory leaks, division by zero, and recursive function calls. They are considered critical bugs that need to be resolved on a priority basis. Unfortunately, the occurrence of such faults can sometimes be non-deterministic and unrepeatable. For example, if the game crashes unexpectedly due to memory leaks, then it will not always occur at a single point in game [PM33]. The *Knightly Adventure*⁴⁰ game, had problems with devices like iPod touch 4th generation and iPhone 3 which led to memory crashes [PM33]. This category is further divided into five sub-categories based on time of occurrence. They include Crash After Action, Crash at Shutdown, Crash at Start Up, Crash During Demo, and Crash During Non-Play Moments. They are described in the proceeding sub-sections.

3.6.1 Crash After Action

One of the most obvious types of bugs in the game are the ones that result in the game unexpectedly crashing after an in-game action has been performed [S8]. In *Cuphead*⁴¹, game crashed on the parry jump action [PM9]. When the player successfully performed the parry action, took a shot after a pause, and then died, the game would crash. The testers found that the function responsible for the execution of the parry action had a dependency on the player module which was unresolved if the player died. The team had a lot of trouble reproducing the crash scenario. They recommended using breakpoints in Integrated Development Environments (IDEs) to stop the game in run-time and analyze game state via code. In *Mighty No.9*⁴², the game unexpectedly crashed when the player fired a weapon or returned to the main menu during gameplay [V8].

3.6.2 Crash At Shutdown

It is critical to ensure that the procedures involved in the shutdown of game in console and PC games, and player exit from game in online games are executed correctly. These shutdown procedures generally include saving the player's game state as well as certain game-specific actions like putting player resources to hibernate. If a game application crashes during shutdown down this can result in loss of player data and issues in initializing the game especially if the faulty shutdown results in game data corruption. In *Super Meat Boy*⁴³, one of the biggest faults was the game crash during shutdown [PM55].

3.6.3 Crash At Start Up

It is critical to ensure that the procedures involved in initialization of game applications in console and PC games, and player login to games in online games are executed correctly. These start-up procedures generally include correctly initializing game applications, retrieving saved game states, restoring players'

⁴⁰<https://www.metacritic.com/game/pc/knightly-adventure>

⁴¹<http://www.cupheadgame.com/>

⁴²<http://www.mightyno9.com/>

⁴³<http://www.supermeatboy.com/>

game data, and in multiplayer game synchronization with other players. If a game application crashes during start-up, then it means that the game has failed to initialize. In *Grand Theft Auto: Vice City*⁴⁴, the game crashed at startup if the player had saved game state at the ice-cream factory save point while reliving the crime kingpin fantasy, which corrupted the save file. Hence, when player reloaded the game from corrupted save file the game crashed at startup [V3]. A similar save file corruption resulted in the same start-up fault in *Prey* [V2].

3.6.4 Crash During Demo

Game demos generally have two variations: playable and non-playable. For this category, we focus on playable demos while non-playable demos are covered in the next sub-section (Crash During Non-Play Moments). Playable demos generally have the same gameplay as the upcoming full game with the purpose of acting as a tutorial to teach new functionalities to the players [30]. While the demo appears to be just a short version of the full-fledged game, that is usually not the case from the development and coding perspective. Hence, even if certain functionalities have been tested during the actual gameplay, they must be tested for the demo separately. Since the demos are generally coded separately from the game base code. In *PONCHO*⁴⁵, there was a crash bug in the game demo, which required game developers to reboot the game [PM36].

3.6.5 Crash During Non-Play Moments

Games generally have in-game cinematic events called cut-scenes or event scenes which are non-play non-interactive game sequences that interrupt gameplay. They are used to reward players, display conversations between characters, consequences of player actions or foreshadow future events. A non-playable demo is essentially the gaming equivalent of a teaser trailer and does not require any player response other than to skip if possible. These can be pre-made videos or rendered during gameplay especially if the cut-scenes require visualization of a customizable character. A game can crash during non-play moments such as during the introduction sequence of a loop boss, Throne II in *Nuclear Throne*⁴⁶ on windows [W1]. Testing for such bugs separately is important because testers generally have animations and cutscenes completely disabled or they skip them.

3.7 Navigational Faults

It is a navigational fault when players are unable to navigate to a place where they should be able to reach or conversely, they are able to navigate to a location they should not be able to access. It can be something simple like the height of a platform being so high that a player cannot reach it. In *Vampire: The Masquerade*⁴⁷, one of the major problems identified in the development process was the problem of path-finding and navigation of variably-sized characters across a completely free-form 3D environment [PM61]. The navigational faults category is further divided into two sub-categories as described in the proceeding sub-sections.

3.7.1 Unreachable Locations

it is an Unreachable Location fault when players are unable to navigate to a place where they should be able to navigate to. It can be something simple like the height of a platform being so high that a player cannot reach it. If a visible platform was unreachable for Mario but Mario needed to climb it to continue with the game progression then it would be a navigational fault. In *Rastan*⁴⁸ on the Commodore 64 computer, at the second level, it is impossible to make a jump over a flaming pit over two ropes to continue with

⁴⁴https://en.wikipedia.org/wiki/Grand_Theft_Auto:_Vice_City

⁴⁵[https://en.wikipedia.org/wiki/Poncho_\(video_game\)](https://en.wikipedia.org/wiki/Poncho_(video_game))

⁴⁶https://en.wikipedia.org/wiki/Nuclear_Throne

⁴⁷<https://www.worldofdarkness.com/vampire-the-masquerade>

⁴⁸[https://en.wikipedia.org/wiki/Rastan_\(video_game\)](https://en.wikipedia.org/wiki/Rastan_(video_game))

the game progression [W2]. Similarly, an enemy in a certain room in *Beyond Good and Evil*⁴⁹ drops a key when it is defeated. However, depending on how the player defeats the enemy, the key can spawn in unreachable locations like inside corners, or in the ceiling, or even slightly beneath the floor [W2].

3.7.2 Unexpected Paths

When players can navigate to a location, they should not be able to access it falls into this category. If in Super Mario, Mario can walk through the green pipe when that should not be possible, it would be an unexpected path's fault. In only faulty collision detection, Mario would not be able to completely pass through the pipe. In *Fallout*, players were sometimes able to reach places and levels that were still being developed and were not meant to be accessed [V2]. These faults can occur due to design and implementation conflicts and are a result of two situations, Faulty Obstacles and Faulty Pits.

Faulty Obstacles: This fault is a combination of faulty collision detection with obstacles and an exploit that allows players to move through the obstacle. So, players can navigate across obstacles against which they should collide. In *Mass Effect*, Matriarch Benezia's use of Biotic powers would toss the main character through the wall and out into the empty void surrounding the rendered game area [W2].

Faulty Pits: This fault allows players to navigate over a pit or empty space through which they should fall. They are a combination of faulty collision detection with the environment and an exploit that allows players to move on air apparently when they should not be able to. If the player can fly or has the ability to hover then falling is not expected, then it would not be a fault. If in *Super Mario Galaxy*⁵⁰, Mario is in flying Mario form then, its ability to hover over pits is not a fault. But if Mario is in any other form and still does not fall through a pit then it is faulty.

3.8 Non-Temporal

This category includes those faults which can be found by inspecting the game state at any point in time. For example, in *Super Mario*, Mario jumps up to a height higher than should be permitted. This category is further divided into seven sub-categories. One sub-category is new, Player Stuck Fault. We have accepted four of the subcategories by Lewis et al. [7] as is, such as Artificial Stupidity, Invalid Value Change, Invalid Position, and Information. We have extended two of the subcategories via the creation of new sub-categories. They are Faulty Action and Invalid Graphical Representation. They are described in the proceeding sub-sections.

3.8.1 Player Stuck Fault

When a player reaches a game state such that there is no means of progression and the game does not terminate, it falls into this category. It can be something obvious like Mario in *Super Mario*, falling into a pit that does not kill him, but its boundaries are high enough that Mario cannot jump out of the pit. Similarly, in *Guacamelee*, the player could get locked in a fight arena unable to exit it or make any further progression in the game [PM13]. It can also be subtle like in *Paper Mario: The Origami King*⁵¹, the *Spring of Rainbows* - VIP card in the game's Shangri-Spa area is needed to get to unlock a secret stage. When the player enters that stage the card is automatically used. If the player exits the stage without completing the quest then there is no way to get the card again and the player is essentially stuck [V23].

3.8.2 Artificial Stupidity

This category includes faults related to undesirable NPC behavior which disillusioned the player to NPC's lack of intelligence. It includes cases in which NPCs blocked the players path or became unresponsive to gameplay interactions. Basically, behaving, as the name implies, in a stupid manner. In *Tresspassers*⁵²,

⁴⁹[https://en.wikipedia.org/wiki/Beyond_Good_%26_Evil_\(video_game\)](https://en.wikipedia.org/wiki/Beyond_Good_%26_Evil_(video_game))

⁵⁰https://en.wikipedia.org/wiki/Super_Mario_Galaxy

⁵¹https://en.wikipedia.org/wiki/Paper_Mario:_The_Origami_King

⁵²[https://en.wikipedia.org/wiki/Tresspasser_\(video_game\)](https://en.wikipedia.org/wiki/Tresspasser_(video_game))

Dinosaurs were governed by a set of emotions that theoretically should have prompted them to pick appropriate responses at any time. However, in practice they oscillated rapidly between many activities, sometimes even standing still, and twitching as they tried to decide what to do [PM12]. This category specifically refers to game AI and elements controlled by it.

3.8.3 Invalid Value Change

This category includes faults that impact the values of counters in an unexpected manner. It includes timers counting down too fast, score increasing unexpectedly, an attack causing an unexpected loss in health or conversely a ‘heal’ spell increasing health unexpectedly. The *Mafia II*⁵³ game, the health bug is a notorious example of this fault. It caused the player’s health to continuously decrease without any reprieve or cause [V15].

3.8.4 Faulty Information

This category includes all the faults related to in-game information that the user has or should have access to including information about the player as well as other players if the game requires such communication.

Invalid Information access: This category includes the faults related to players having access to information; they should not have. In single-player games, this can ability to see through walls and obstacles, or information regarding enemy locations or game maps that should only be accessible in later stages of games. In multiplayer games, this includes information regarding statistics and movements of other players.

Lack of required information: This category includes the faults related to players lacking access to information that they should have. In single-player games, this can include the inability to see open pathways (a malfunction of in-game camera), or information regarding enemy locations or game maps that should have become accessible in specific stages of games. In multiplayer games, this can include information regarding statistics and movements of allied players needed to make strategic decisions.

Information out of order: This category includes the faults in which players receive information in an unexpected or faulty order. It is most relevant in role-playing games where players have many alternatives of receiving certain information but it is likely that other sources of such information are not updated to reflect that information access. In *F.E.A.R.*⁵⁴ game, at one point, player is asked to download data from a laptop. There are multiple laptops with downloadable data to get the backstory. However, the player is told to download this data from a particular laptop, a long time after encountering this laptop. However once the data has been downloaded from any machine, it cannot be done again and the mission objectives are not updated to show task completion. The player basically must restart from the last save and wait for the objective to be given, then download the data [W2].

3.8.5 Invalid Position

It includes faults in which an in-game item is in an invalid position. It differs from Invalid Position Over time in which an in-game element retains a position that is valid at any single point in time but invalid or faulty in context with previous states. If Mario in *Super Mario bros.* jumps to an invalid height it is an invalid position. Instead, if Mario hovers at the right height for too long then it is Invalid Position Over time.

Object out of bounds for any State: This category includes faults in which an in-game element is in an invalid position regardless of game state. In *Project Gotham Racing 3*⁵⁵, there was a bug where cars started 30 feet in the air, facing the wrong way around [PM37].

Object out of bounds for a specific state: This category includes faults in which an in-game element is in an invalid position only because of the corresponding game state. If the flying Mario in *Super Mario Bros.* cannot be underwater, and it falls below the water surface, then it is out of bounds for that specific state. If it was not in flying Mario form then it would not be considered a fault.

⁵³https://en.wikipedia.org/wiki/Mafia_II

⁵⁴[https://en.wikipedia.org/wiki/F.E.A.R._\(video_game\)](https://en.wikipedia.org/wiki/F.E.A.R._(video_game))

⁵⁵https://en.wikipedia.org/wiki/Project_Gotham_Racing_3

3.8.6 Faulty Action

This category includes the faulty execution of an in-game action by a player either due to the inability to act when permissible or ability to take impermissible action. Actions are the “verbs” of game mechanics. They are base in-game operations a player can perform [28]. This category is further divided into four sub-categories including two sub-categories that are newly proposed after discussion amongst the authors - Multiple Action Inconsistency and Repeated Action Inconsistency. We have accepted two of the sub-categories by Lewis et al. [7] as is - Action Not Possible and Action When Not Allowed. They are described in the proceeding sub-sections.

Action Not Possible: In simple terms, a player is unable to take an in-game action in a specific game state that should be permissible in that game state. In *Sanitarium*⁵⁶ game, Max (the character the player controls) would get stuck around corners and claim “Can’t go that way”, even when path was available [PM11]. Similarly, *Crysis*⁵⁷ had a bug that made the final boss randomly become ‘untargetable’ (and thus invulnerable) which meant that no form of action was possible [W2]. *Space Station Silicon Valley*⁵⁸ (for Nintendo 64) is impossible to win with Hundred-Percent Completion because an action required to pick up the critical item is not possible [W2]. In *Sphinx and the cursed mummy*⁵⁹, if a player saves game at midpoint save point a gate needed to progress is closed permanently and open-door action is not possible [W1].

Action When Not Allowed: In this category, a player is able to take an in-game action in a specific game state that should not be permissible in that game state. *Global Agenda*’s 1.3.2 patch⁶⁰ contained a major bug in the auction house that allowed players to effectively create money from nothing [W2]. *League of Legends: Season 11* as a bug that allows players to one-shot enemies using all types of weapons even those that do not permit this action [V16].

Multiple Action Inconsistency: It encompasses faults that occur only when multiple different actions are taken simultaneously or in a certain order. Such faults are deterministic and repeatable. When Mario is made to fire an attack while jumping then the player is taking two actions simultaneously. Mario’s fire attack could be working as intended in isolation but it is possible that it does not work while Mario is jumping. In *Runescape 3*⁶¹, a game-breaking bug allows players to instant kill any boss in the game without using up any ‘death touched’ darts. This bug only occurs when certain weapons are equipped and used in a particular order [V1].

Repeated Action Inconsistency: It encompasses faults that occur only when an action is repeated. Such faults are deterministic and repeatable. When Mario is made to jump continuously it is meant to make small hops. Mario’s jump could be working as intended in isolation but it is possible that it does not work while Mario is made to jump multiple times repeatedly. *Psychonauts*⁶² had a quirk where a player became unable to use double jump in a level where it was needed to jump between flaming grates while the water level is rapidly rising [W2].

3.8.7 Invalid Graphical Representation

These faults occur when game graphics are faulty. In other words, the graphical representation of game elements is faulty either due to rendering or location of appearance. When Mario in *Super Mario Bros.* catches a fire flower, he transforms into Fire Mario. If the graphical representation of Mario remains unchanged or becomes something else then it will fall into this category of faults. The reasons for such faults can vary from the defects of hardware (e.g., GPU-related issues) to game application settings (e.g., the wrong setting of rendering special effects) to source-code bugs (e.g., incorrect transitions) [S11]. Such issues are most found in graphically demanding games such as *Assasin’s Creed*, *Red Dead redemption*⁶³,

⁵⁶[https://en.wikipedia.org/wiki/Sanitarium_\(video_game\)](https://en.wikipedia.org/wiki/Sanitarium_(video_game))

⁵⁷<https://www.crysis.com/>

⁵⁸https://en.wikipedia.org/wiki/Space_Station_Silicon_Valley

⁵⁹https://en.wikipedia.org/wiki/Sphinx_and_the_Cursed_Mummy

⁶⁰https://en.wikipedia.org/wiki/Global_Agenda

⁶¹<https://play.runescape.com/>

⁶²<https://en.wikipedia.org/wiki/Psychonauts>

⁶³https://en.wikipedia.org/wiki/Red_Dead_Redemption

and *Cyber punk*. This category is further divided into six new sub-categories that we have added to our updated taxonomy, including Abnormal Text [S11], [S22], [S41], [S46], Corrupted Frame [S11], [S22], [S30], [S41], [S46], [S69], Extra Game Asset [S22], [S41], [S46], [S70], Incorrect Visual State Transition [S30], [S43], [S46], [S70], [S73], Missing Game Asset [S11], [S22], [S41], [S43], [S46], [S70], and Slow loading [S41], [S46], [S69]. They are described as follows.

Abnormal Text: This fault encompasses issues with the appearance of text in the game application. Text is an important aspect of many modern games whether it is in form of dialog between in-game characters, instruction for a player to help with the game progression or communication between different players in multi-player games. This fault can take many forms. Text can appear in the wrong location; it may even cover a character or any other game object. Text can be incomprehensible either due to being blurry or due to low contrast with background or simply displaying wrong information [S11]. In *Pokémon Red, Blue, Green and Yellow*⁶⁴, the text describing *MISSINGNO* pokemon was often glitchy [V3].

Corrupted Frame: The most common example of a corrupted frame is the game screen becoming pixelated. Pixelation is caused by displaying a bitmap or a section of a bitmap at such a large size that individual pixels, small single-colored square display elements that comprise the bitmap, are visible. The image frame could become scrambled if incorrect post-processing effects have been added, or resolution requirements of the game applications are not compatible with the deployment platform. This could occur for the complete frame or just part of the frame. A more subtle case of this fault can occur if the frame as a whole has an unexpected colored tint. In *Amnesia: A Machine for Pigs*⁶⁵, earlier players noticed a blue ‘fog’ during gameplay which was actually due to compromised color-grading [PM56].

Extra Game Asset: As the name implies an in-game element that should not be in a particular game state is visible. In *Jak X: Combat Racing*⁶⁶, has a fault known as *saving glitch* where the save icon stays visible on the game at all times. Forcing the player to hard reset, if they want to get rid of it [W1].

Incorrect Visual State Transition: An in-game element transitions to an incorrect state. The animation for a jumping Mario in Super Mario Bros., is different to that of a swimming Mario. If swimming Mario animation is made visible when Mario is jumping then it would fall into this category. In *Slow Down, Bull*⁶⁷, Annette the bull catcher was notorious for not transitioning properly through her various tell states, even when all the variables were set, due to the un-intuitive transition settings in the animators overriding what animation was being told to play in the state machine [PM2]

Missing Game Asset: An in-game element that should be visible in a particular game state is not visible. In *Sins of a Solar Empire: Rebellion*, A battle could be raging in one corner of the galaxy between two players, while the third player would see nothing [PM49]. In *Donkey Kong Country 2*, if a player picks up the barrel as it breaks, then the avatar is seen carrying an invisible barrel [V3]. In *Pokémon Red, Blue, Green and Yellow*, a player could try to fight or capture a glitchy pokemon *MISSINGNO* which only appeared as a collection of corrupted pixels [V3].

Slow loading: The in-game graphics experience a delay in rendering on the screen. Most common example of Slow Loading is low frame rate. It also manifests in delays in game asset renderings especially in online games. These issues are not necessarily encoded in the game application, rather can also arise due to limitations of the host machine and network speed. They are commonly found in graphically demanding games. *Elder Scrolls V: Skyrim* was notorious for its rendering issues [V4]. Similarly, *Mighty No.9*, also fell victim to slow frame rate [V8].

4 Validation

In this section, we evaluate the developed game bug taxonomy which we have described in the previous section. We performed a survey to gain insights from the participants, using google forms, regarding the frequency and severity of the game bugs as well as the priority with which they should be fixed. Such surveys have been widely used in the literature for validation purposes as they are an effective means

⁶⁴https://awesomgames.miraheze.org/wiki/Pok%C3%A9mon_Red,_Blue,_Green_and_Yellow

⁶⁵<https://www.aamfp.com/>

⁶⁶https://en.wikipedia.org/wiki/Jak_X:_Combat_Racing

⁶⁷https://store.steampowered.com/app/333580/Slow_Down_Bull/

to validate proposed fault models directly from the experts [31]. In the following sub-sections, we have described the procedure by which we designed our survey questionnaire, the process of response collection and finally the result synthesis.

4.1 Survey Design

In this sub-section, we have described the procedure by which we designed our survey questionnaire. It consists of 60 questions. The details on the survey questionnaire are given in appendix C via table C.1. The actual questionnaire can be found at <https://bit.ly/vgbtsurveypdf>. We have followed the state-of-the-art survey guidelines for developing the online survey questionnaire presented in [32]. These guidelines specified that anonymity of the respondent must be ensured. They suggested use of likert-scales as well as provision of necessary instructions support readability and understanding.

The Question 1-9 cover the background information regarding the participants as well as their area of expertise. We used Likert scales to gauge the frequency, severity, and priority of our identified game bug categories, as follows:

1. 5-point Likert scale for frequency of game bugs following the guidelines in [33]
2. 6-point Likert scale for severity is used in the survey questionnaire for game bugs following the guidelines in [34].
3. 6-point Likert scale for priority that should be allocated to fixing a certain type of bug in games following the guidelines in [35].

The questions related to severity and priority of bugs are correlated. Their presence helps us recognize quality responses. For example, Q11 asks the participants to rate severity of Gaming Balance Faults while playing games while Q12 asks the participants to prioritize the need to fix such faults. If the participant considers the bug to be critical then congruently its priority should be high.

4.2 Pilot Survey

To help ensure the understand-ability of the survey, all the authors went through the survey. We then conducted a pilot survey with five participants. A pilot survey is a common tool to test the research tools including the questions, and survey structure [36]. We asked two less frequent game players and three avid gamers to review the survey and give their response to ensure the questions were clear and complete. For this purpose we used the think aloud technique to get the feedback of participants regarding survey design. They only suggested minor edits. The changes we made include: adding descriptive information regarding answers where selection of multiple options was allowed, adding clarifying description for sub-categories of game bugs, adding an image of taxonomy before final remarks, adding three commonly played game genres to the relevant question.

4.3 Survey Execution

In this sub-section, we have described the process of response collection. We have followed the guidelines presented in [32] for conducting our online survey. The participants of the survey population were identified from LinkedIn⁶⁸ following the recommendations in [37]. We provided the descriptions of all the taxonomy terms as well as the explanations of the likert scale terms used to rate severity, frequency, and priority of game bugs. We included the descriptions in the survey which we sent to the survey population.

4.4 Evaluation of the game bug taxonomy

In this sub-section, we have described the process of synthesis of the result. We received 168 responses to our survey. The details of expertise of the respondents are shown in figures D.8, D.9, and D.10 of appendix D. The complete set of anonymized survey responses are available at <https://bit.ly/vgbtsurvey>.

⁶⁸<https://www.linkedin.com>

The survey showed that on average 96% of survey respondents encountered our main-tier game bug categories (figure D.5 of appendix D). Network faults are considered most frequent (with 10.2%, 43.5%, 36.4% and 9% respondents encountering them always, often, sometimes, and rarely respectively) while Sound faults are considered least frequent (with 6%, 16.1%, 32.2%, 35.2%, and 10.8% respondents encountering them always, often, sometimes, rarely, and never respectively). This is depicted in figure D.4 of appendix D which graphically presents the survey data. Similarly, **60%**(average percentage of respondents that consider main-tier faults with major or higher severity) of the respondents agree that all the identified main-tier game bug categories have a Major or higher severity with exception of sound and navigational faults (with only 48.9% and 47.2% considering them major and higher respectively) which are considered by most to be of minor or low severity. Alternately, network faults are considered most severe (with 10.8%, 37%, 30.3%, 17.3%, and 4% respondents considering them blocker, critical, major, minor and low respectively). This is depicted in figure D.6 of appendix D. Alternatively, on average **77.5%** of the respondents agree that all the identified main-tier game bug categories should have atleast medium priority fix with exception of sound and navigational faults (depicted in figure D.7 of appendix D)

In terms of frequency, less than 4% respondents stated that they had never encountered the identified bugs. Comparatively, over 96% of the respondents agreed that they had encountered such bugs. It should also be noted that even the subcategories least encountered were encountered by over 11% of the respondents. This is reflected in figure D.5 of the appendix D. The four percent can be explained by the genre of games that were played by the particular respondents for example, the respondent who claimed to not have encountered sound faults mostly played casual games like candy crush or racing, and platform games for such games sounds are not a critical part of gameplay.

Moreover, to identify relationships between our defined metrics (Frequency, Severity and Priority), we performed correlation analysis to quantify the degree to which our defined metrics are related and presented our findings in table 1. We used Spearman’s rank correlation to find the correlation between Frequency, Severity and Priority of game bug categories as we gathered ordinal data corresponding to each metric. The table 1 presents Spearman correlation coefficient values corresponding to each category. The Spearman correlation coefficient, r_s , can take values from +1 to -1. The r_s of +1 indicates a perfect association of ranks, while r_s of zero indicates no association between ranks and r_s of -1 indicates a perfect negative association of ranks. The closer r_s is to zero, the weaker the association between the ranks. If $0.00 \leq r_s \leq 0.20$, then the correlation is considered negligible(highlighted in red). If $0.21 \leq r_s \leq 0.40$, then the correlation is considered weak(highlighted in yellow). This means that there is a weak tendency for the two variables to increase or decrease together, but there are also many exceptions. If $0.41 \leq r_s \leq 0.60$, then the correlation is considered moderate(highlighted in green).

The tests show that the correlation between the three variables (frequency, severity and priority) is mostly weak to moderately **positive** as shown by values highlighted in yellow and green in the table 1. For example, correlation between severity of navigation faults with the priority to fix it is 0.4701. The reason for this could be that game developers tend to prioritize fixing issues that are severe, and navigation faults in games can have a significant impact on the player’s experience. Therefore, if a navigation fault is severe, it is likely to be given higher priority for fixing. The values highlighted in yellow depict weak positive correlation. For example, correlation between frequency of occurrence of network faults with the priority to fix it is 0.3458. This can be due to the fact, that network faults are not necessarily revealed at developers’ end rather they arise at players’ side due to issues in internet connection, network traffic etc [PM6]. The priority to fix them is not as high as the frequency of occurrence of network faults. The values highlighted in red depict a negligible correlation. The correlation between Frequency and severity is mostly **weakly-positive** as shown by values highlighted in yellow in table 1 with exception of Unexpected Crash which is negligible. The reason for this could be that not all crash faults lead to catastrophic consequences, such as loss of progress or data corruption. Some crash faults may simply result in the game closing or freezing temporarily, which may be frustrating but not necessarily severe. Additionally, some crash faults may only occur under specific circumstances or with specific hardware configurations, leading to a high frequency of occurrence among a subset of players but not across the entire player base.

The correlation between Frequency and Priority is **weakly-positive** or **negligible** as shown by values highlighted in yellow and red in table 1. This is because the bugs that occur most frequently are not necessarily the most severe or impactful ones. Therefore, the bugs that are less frequent but have a higher

impact on the gameplay experience would be prioritized. Additionally, other factors such as the complexity of fixing the bug, available resources, and time constraints may also influence the priority assigned to fixing a particular bug.

The correlation between Severity and Priority is **moderately-positive** as shown by values highlighted in green table 1. The higher correlation between the severity and priority of game bugs suggests that the severity of a bug is an important factor in determining its priority for fixing. Additionally, bugs with a higher severity rating may also be more urgent to fix because they could potentially cause players to stop playing the game.

None of the values of correlation are greater than 0.8 which implies a very strong positive correlation. The results of the statistical tests confirm the implications of the MLR itself. The frequency of occurrence of a game bug does not necessarily imply severity and priority. Rather severity depends on the impact the bug has on a player’s overall ability to complete a gameplay task. Any specific bug category, in general, is not the most critical rather it varies from game to game.

Table 1: Illustrates the spearman’s rank correlation between frequency of occurrence of bug with its perceived severity, frequency of occurrence of bug with its perceived priority and a bugs perceived severity with its perceived priority respectively.

Bug Category	Frequency & Severity	Frequency & Priority	Severity & Priority
Gaming Balance	0.3394	0.1870	0.4812
Implementation Response	0.3505	0.0275	0.2860
Network	0.3550	0.3458	0.5443
Sound	0.4162	0.2449	0.5715
Temporal	0.4369	0.3691	0.4931
Unexpected Crash	0.0940	0.1536	0.4890
Navigation	0.4883	0.3120	0.4701
Non Temporal	0.3417	0.1952	0.5276

5 Discussion

This section summarizes the lessons learned. We discuss the implications for each game bug category and provide an overview of research trends.

5.1 Gaming Bugs

5.1.1 Gaming Balance

There appears to be an upward trend in focus on gaming balance issues in academic literature, especially with the rise in interest in MMO games where variation in parameters of one archetype or model may influence game-play experience of all archetypes or models. This is also consistent with our survey, as 32.1% of the respondents considered gaming balance issues to have major severity, 22.6% considered them critical and 4.2% considered them to possess blocker severity. Most frequently encountered Gaming Balance faults were too easy and too hard, encountered by 46.7% of respondents.

5.1.2 Sound Faults

On the contrary, there is a lack of research related to the identification of sound bugs in games. We only found one approach (i.e., [S58]) in the academic literature that focused on identification of sound faults mentioned in Section 3.4. Sound effects (SFX) provide important gameplay cues for players where they clarify or reinforce player actions, giving feedback on the player’s decisions. Game sound is important as it gives the game context, individuality, and depth. Iconic games are synonymous with their soundtracks and sound effects, and it is beyond doubt that a strong palette of sound is integral to the player’s overall experience of a game. Inversely, issues in-game sounds can lead to bad user experience. It is especially critical for rhythm and dance games which require on-spot sound effects and synchronized music [W2].

The results of our survey show that while most of the respondents considered the sound faults to be a minor inconvenience, the players that made use of sounds for tactics (e.g. calculate distance of enemy from sound of footsteps in DOTA) considered it a severe issue.

5.1.3 Implementation Response Faults

In total 29 out of 78 selected academic studies on automated game testing approaches focus on implementation response faults. Most frequently encountered Implementation response issue is Game Freeze and it is encountered by over 65% (or 109) of respondents as compared to second most frequent which was collision encountered by 50 respondents (29.7%). However, only 4 and 6 academic studies focus on them respectively. Alternatively, incorrect response is covered by 16 studies but only 27% (46) of respondents claim to have encountered them

5.1.4 Network Faults

Network faults are considered most severe (with 10.8%, 37%, 30.3%, 17.3%, and 4% respondents considering them blocker, critical, major, minor and low respectively). However, only 9 out of 78 selected academic studies focused on network faults. The respondents who stated that they always encountered navigational faults mostly played MOBA and action genre games.

Most frequently encountered Network faults is Network lag and it is encountered by 63.1% (106) of respondents followed by connection fault which was encountered by 42.2% (71) of the respondents. However, none of the academic studies cover network lag.

5.1.5 Temporal Faults

Most frequently encountered Temporal faults are Delayed Response and it is encountered by 49.4% (83) of respondents followed by Interrupted Event which was encountered by 30.9% (52) of the respondents. Unfortunately, neither of these two categories is covered in academic literature.

5.1.6 Non-Temporal Faults

Most frequently encountered Non-Temporal faults are Player Stuck issue and it is encountered by 41.6% (70) of respondents followed by Faulty Action which was encountered by 32.9% (55) of the respondents. Contrarily only 4 studies in the academic literature cover the Player Stuck issue while 12 studies cover faulty actions.

5.1.7 Navigational Faults

Certain bugs like being able to move through walls, are sometimes considered tactic features. Michal Madej [T4] in his talk in *Digital Dragons* 2014 discussed how certain bugs in their game were not reported by the testers because those bugs were being used as strategy tactics [T4]. The respondents who stated that they *always* encountered navigational faults mostly played MOBA and action genre games. Contrarily, many of the respondents who stated *never* were players of casual games like candy crush and angry birds that do not have an in-game navigation element.

5.1.8 Unexpected Crash Faults

The survey shows that correlation between frequency and priority, and frequency and severity of such faults is nearly negligible. The reason for this could be that the frequency of game crashing faults may not necessarily indicate their impact on gameplay or the user experience. Some crashes may occur more frequently than others but may not have a significant impact on the overall gameplay or user experience. Alternatively, there is a moderately positive correlation between severity and priority. Most frequently encountered Crashes are Crash as Start Up and Crash after Action encountered by 49.4% (83) and 44.6% (75) of respondents respectively.

5.2 Research Trends

In recent years, there has been an increased focus on using machine learning for automated game testing and bug identification as concluded by [38] and shown in [S3], [S4], [S5], [S6], [S11], [S15], [S21], [S41]. A major reason for this rise, is the capacity of machine learning techniques to handle unexpected scenarios during dynamic game-play. The dynamic game-play is a consequence of presence of intelligent agents in PvE and PvP games.

Consequently, the popularity of dynamic game-play makes the automated generation of test oracle very difficult. In most of the existing game testing techniques, test oracle generation is manual. It is done either via human observation; by simply checking for game crashes; by testing for visual glitches; or by adding game-specific test assertion into the game source code [S6] like checking for navigability on game-map; or by modeling game environment in a deterministic form like state machines [S2], [S10].

However, a serious limitation of automated oracle generation techniques is that they are specific to games. Model-based techniques can permit a certain level of generalization, provided a game-specific model is defined. The model can be in form of a domain-specific language like VGDL [S20], a state machine [S2], [S10], or any other modeling language that can be used to model game scenarios like the graphical modeling language used in [S7] that allows testers to model action sequences needed to complete a game quest scenario in a role-playing game. Unfortunately, exhaustive modeling of complex MMO games like *Elder Scrolls V: Skyrim* is a challenging avenue.

5.3 Game Testing Techniques

The trend of game-testing techniques proposed in academic literature shows that most of the work is focused on AI-based techniques. Gaming balance faults are captured using RL [S15], [S26], [S36], [S40], and AI based approaches [S34], [S49], [S52], [S54], [S55], [S56]. Implementation faults are captured using manual scripting [S13], [S24], [S29], [S65], [S68] and AI based approaches [S8], [S24], [S49], [S72]. Network faults are identified using Stress testing. Temporal and Navigation faults are least focused. Temporal faults are mostly handled by using runtime monitoring systems and navigational ones are handled using RL and Search based approaches. Unexpected crashes are handled using RL and EA-based approaches. Non-temporal faults are captured using RL, AI and ML-based approaches like image recognition (figure 6). The detailed categorization of selected studies based on testing techniques versus bug category is present in *Video Game Bug Taxonomy MLR - bug capturing techniques* in the dataset as well as table B.3 in appendix B.

Difficulty in generalizing an automated testing technique across genres and platforms is also a major cause of lack of automation [2]. One difficulty is that different test techniques usually require different pieces of information about the system, e.g., specifications in different forms, or may target different types of systems as well as different goals [39]. An approach designed for 3-match games (like Candy Crush) will not map to a platform game (like Super Mario) which are popular genres according to our survey. On the other hand, an approach designed to test reachability of a player's avatar to a particular location for platform games (like Super Mario) could map to arcade or role-playing games (like Legend of Zelda).

Majority of the game testing approaches target system-level testing, which is also highlighted in post-mortems. Most of the testing is done in final stages of development which leads to time crunch and bugs being passed to post-production phase. Our conclusion is early integration of testing is needed which aligns with findings drawn by [3], [4], [PM15].

5.4 Lessons Learned from Grey Literature

During analysis of grey literature, we learned of an interesting aspect of presence of game bugs; their exploitation by game players to the extent that some are treated as features. Certain bugs like being able to move through walls are sometimes considered tactic features. Michał Madej [T4] in his talk in *Digital Dragons 2014* discussed how certain bugs in their game were not reported by the testers because those bugs were being used as strategy tactics [T4]. This introduces the concepts of *bugs as features*, which is a prevalent theme in MMO games. These *bugs* can belong to any of the proposed bug categories. Major

MMORPG game servers like *Runescape-3* and *Elite Dungeons* place temporary bans on players caught abusing bugs as well as penalize them via removal of accumulated wealth and bonuses [V1].

The impact of different game bugs is not consistent during gameplay experience for games of different genres. Not all bugs equally impact gameplay experience. Most of the respondents considered the severity of network issues to be high. The results of our survey show that while most of the respondents considered the sound faults to be a minor inconvenience, the players that made use of sounds for tactics (e.g. calculate distance of enemy from sound of footsteps in DOTA) considered it a severe issue.

A major challenge highlighted in the grey literature, especially in postmortems related to game testing is the difficulty in reproducibility of game bugs. Video games are complex software with cross-cutting dependencies which makes reproducing a bug difficult [PM9], [PM10], [PM49] in such situations log files and bug reports were found to be the most important resources. Postmortem analysis showed that industry mostly used external playtesters to test their games at last stages of development. Very rarely automated testing techniques were used and even those focused more on automated execution of scripted or recorded scenarios.

5.5 Discussion Regarding Proposed Taxonomy

An advantage of bug taxonomies is the identification of bugs that can be seeded into software to evaluate testing approaches. This opens another avenue of research, the proposition of mutation operators corresponding to categories in bugs. An example of this can be removing collision detection from an obstacle.

According to Lough et al. [40], a taxonomy should be comprehensible, complete, deterministic, mutually exclusive, repeatable, well defined, unambiguous and useful. We have designed our taxonomy keeping to the characteristics defined by [40]. However, we have observed that in the domain of games, mutual exclusivity is not possible due complexity of system interactions and sequence combinations. The resulting inevitability of cross-cutting and overlapping means that bugs may be placed in various categories. For example, a player’s avatar is able to pass through a blocking obstacle. This creates a bug that falls into category of both “Collision Detection Fault” and “Faulty Obstacle”. Hence, in such a scenario, it will depend on the observer perspective where the bug is placed. If the bug is categorized from perspective of the avatar then it would be collision detection fault; alternatively if the focus is the new obstacle then it could be faulty obstacle.

The focus of the taxonomy is the Implementation Faults that occur during game execution from the time game application is launched or game player logs in to the game, until it is shut down or game player logs out. Hence at this stage, we do not consider installation or deletion faults; or development or deployment platform compatibility faults within the scope of this taxonomy. Such faults can be the focus of future work.

6 Threats to Validity

In this section, the threats to validity are explained in detail along with steps followed to minimize their influence.

6.1 Internal Threats Validity

The selection of literature is an internal validity threat in this study. To mitigate, we used a systematic search method to find the white and grey literature. We searched the major digital libraries with different query phrases and utilized a uniform inclusion and exclusion criteria for final academic study selection. Despite using a systematic approach, there is still a potential of missing a relevant study, and there may also be studies published in languages other than English. However, we limited our search to studies that were published in English. Another threat to internal validity is researcher bias. We attempted to mitigate it by having each manuscript evaluated by at least two authors and having all conflicts in study selection discussed and resolved by several reviews and group meetings among the authors.

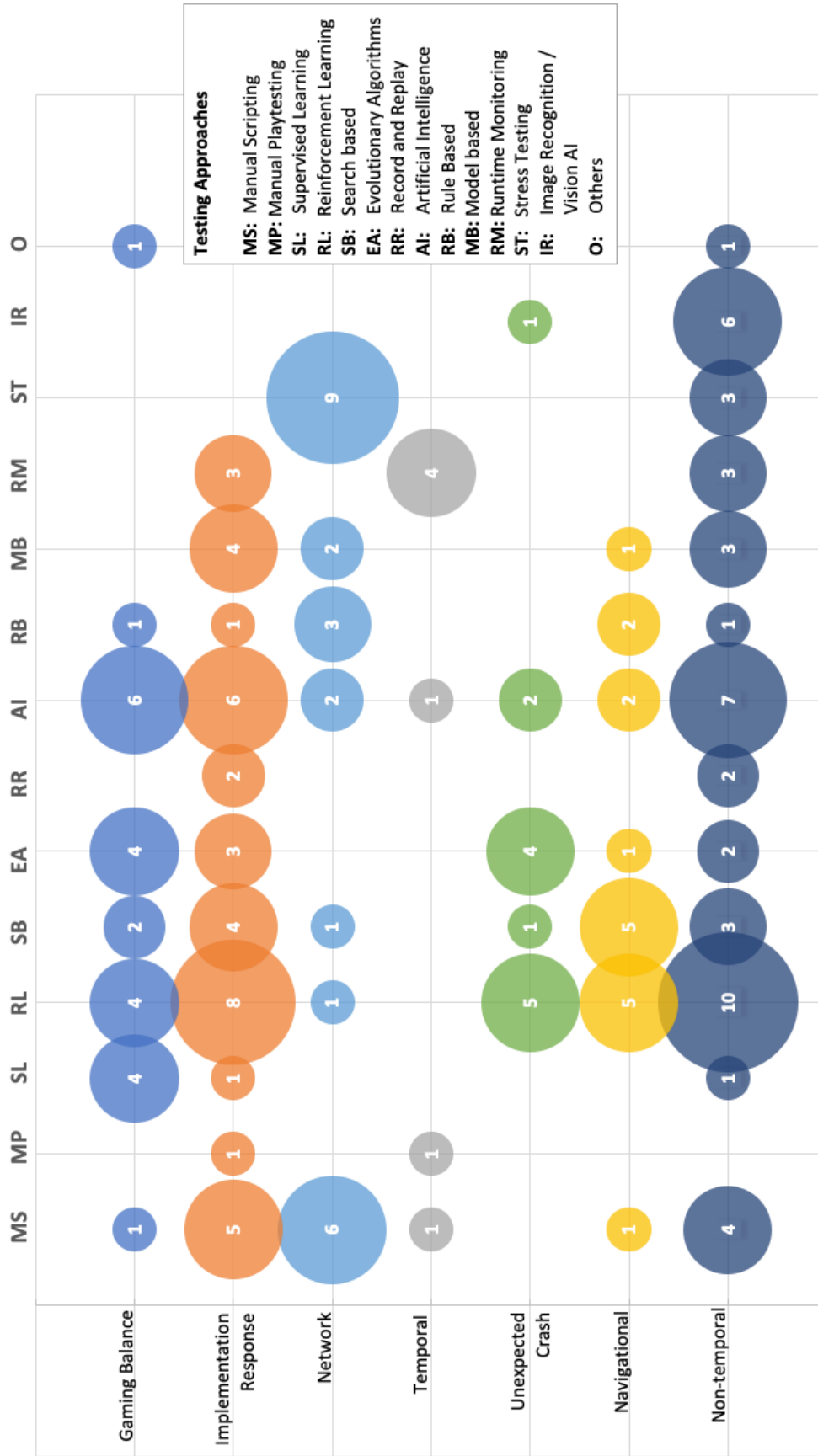


Figure 6: Illustrates the types of game testing techniques used for identification of game bug.

6.2 Construct Threats Validity

We used the findings from the MLR when we refined the questionnaire following many discussions among the authors, in order to offset difficulties with construct validity. Another threat to construct validity was whether the survey participants correctly understood the identified game bug categories. To achieve this, we supplied detailed descriptions of the identified categories, for each of the relevant questions. To Further ensure objectivity of the participants, we assured all participants that their identities would be kept anonymous.

6.3 Conclusions Threats Validity

To validate the treatment’s dependability, all studies were thoroughly examined by at least two authors to eliminate bias in data extraction, which can lead to inaccurate conclusions. We also evaluated the quality of included studies which confirmed that most of them are of good quality based on the quality assessment criteria. Any disagreements over the retrieved data were settled by consensus among the authors. To ensure data traceability, the provided graphs and tables are created from the extracted data in a spreadsheet.

7 Related Work

In this section, we discuss related literature, highlighting the differences from each study. Politowski et al. [3] and Washburn et al. [41] analyzed postmortems from the video-game development industry, with a focus on management and production aspects, and best practices and difficulties in the development process, respectively. Our analysis of postmortems from *Gamesutra.com* focuses on implementation faults resulting in gameplay bugs, with extended and specialized categories, including Graphics/sound issues and Game mechanics/system issues. Politowski et al. [2] surveyed the game testing practices in the industry and found that manual playtesting and intrinsic knowledge are the most commonly used methods. Albaghajati et al. [38] conducted a survey of academic literature to provide a framework for game testing approaches. Redavid et al. [42] presented an overview of game testing practices from a software engineering perspective. We also surveyed academic and grey literature, with a focus on identifying limitations of existing automated game testing approaches and classifying bugs encountered by game players. Our proposed taxonomy provides guidance for playtesters and researchers in creating robust automated game testing approaches. Other related studies include diagnostic taxonomies of game failures from usability perspective [9], challenges specific to network computer games [29], and a bug database for comparative study of testing techniques [43].

We have extended the taxonomy presented in [7] after an MLR to identify and classify commonly appearing bugs that result in game failures even after launch. We have added major classifications of Navigational bugs, Network problems, and Gaming Balance and further expanded the non-temporal implementation failures class to include sound issues, unexpected crashes and System response issues. We have also added to other classes described in detail in the following sections. We have validated our taxonomy from professionals.

8 Conclusion

This paper proposed detailed game bug taxonomy and discussed essential characteristics of the existing game testing approaches. The taxonomy is based on comprehensive analysis of game bugs and existing game testing approaches by conducting systematic review of white literature and grey literature. The classification of bugs into different categories is performed by thoroughly analyzing the existing classification schemes for bugs and game-testing approaches. The taxonomy provides classification for **63** different types of **implementation faults** found 189 different sources of relevant literature. These 63 faults are classified into eight high-level categories i.e. Gaming Balance, Implementation response, Network, Sound, Temporal, Unexpected Crash, Navigational and Non-Temporal Faults. Five out of 63 faults have been observed and modified from existing literature and **15** categories are used from existing literature as is.

While **43** categories are newly added as a result of meticulous MLR process. We validated the proposed taxonomy by conducting survey involving different game industry practitioners such as game developers, game testers, and game players. The results of the survey ratify the validity of bug taxonomy. We also performed correlation analysis between the defined metrics, which show moderately positive correlation between severity and priority and weakly positive correlation between frequency and priority of the bugs.

We observed that manual approaches toward game testing are still dominating in the industry. There is very little work done for automated detection of sound bugs in games. Conversely, game balancing is the most studied topic in recent literature. Furthermore, most game testing techniques are specialized and depend on specific platforms. Like other sub-areas of testing, the issue of automated oracle also exists in game testing. Most recent approaches in game testing are incorporating machine learning techniques. In future, we plan to focus on installation and deletion faults as well as development and deployment platform compatibility faults.

References

- [1] T. Dobrilova, *How much is the gaming industry worth in 2021?* Dec. 2021. [Online]. Available: <https://techjury.net/blog/gaming-industry-worth/#gref>.
- [2] 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)*, IEEE, 2021, pp. 90–99.
- [3] 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*, vol. 134, p. 106538, 2021.
- [4] C. P. Schultz and R. D. Bryant, *Game testing: All in one*. Mercury Learning and Information, 2016.
- [5] M. Felderer and A. Beer, “Using defect taxonomies for testing requirements,” *IEEE Software*, vol. 32, no. 3, pp. 94–101, 2014.
- [6] A. Marchetto, F. Ricca, and P. Tonella, “An empirical validation of a web fault taxonomy and its usage for web testing,” *Journal of Web Engineering*, pp. 316–345, 2009.
- [7] 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*, 2010.
- [8] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, “Taxonomy of real faults in deep learning systems,” in *ICSE ’20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, IEEE Computer Society, 2020, pp. 1110–1121, ISBN: 9781450371216.
- [9] B. Aytemiz and A. M. Smith, “A diagnostic taxonomy of failure in videogames,” *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, 2020.
- [10] Y. Wang, M. V. Mäntylä, Z. Liu, J. Markkula, and P. Raulamo-jurvanen, “Improving test automation maturity: A multivocal literature review,” *Software Testing, Verification and Reliability*, vol. 32, no. 3, e1804, 2022.
- [11] R. J. Adams, P. Smart, and A. S. Huff, “Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies,” *International Journal of Management Reviews*, vol. 19, no. 4, 2017.
- [12] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, “Deep learning for video game playing,” *IEEE Transactions on Games*, vol. 12, 2020.

- [13] M. U. Khan, S. Sherin, M. Z. Iqbal, and R. Zahid, "Landscaping systematic mapping studies in software engineering: A tertiary study," *Journal of Systems and Software*, vol. 149, pp. 396–436, 2019.
- [14] A. Albaghajati and M. Ahmed, "A co-evolutionary genetic algorithms approach to detect video game bugs," *Journal of Systems and Software*, vol. 188, p. 111 261, 2022.
- [15] F. D. Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "Ai-based playtesting of contemporary board games," in *ACM International Conference Proceeding Series*, vol. Part F130151, Association for Computing Machinery, 2017.
- [16] M. Schatten, I. Tomicic, B. O. Duric, and N. Ivkovic, "Towards an agent-based automated testing environment for massively multi-player role playing games," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics(MIPRO)*, IEEE.
- [17] M. Schatten, B. O. Đurić, and I. Tomićić, "Towards an application programming interface for automated testing of artificial intelligence agents in massively multi-player on-line role-playing games," in *Central European Conference on Information and Intelligent Systems*.
- [18] G. Lovreto, A. T. Endo, P. Nardi, and V. H. S. Durelli, "Automated tests for mobile games: An experience report," in *17th Brazilian Symposium on Computer Games*.
- [19] F. De, M. Silva, I. Borovikov, and K. Zaman, "Exploring gameplay with ai agents," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.
- [20] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis," *FDG*,
- [21] Y. Zhou, H. Zhang, X. Huang, S. Yang, M. A. Babar, and H. Tang, "Quality assessment of systematic reviews in software engineering: A tertiary study," in *Proceedings of the 19th international conference on evaluation and assessment in software engineering*, 2015, pp. 1–14.
- [22] G. Tebes, D. Peppino, P. Becker, G. Matturro, M. Solari, and L. Olsina, "A systematic review on software testing ontologies," in *International conference on the quality of information and communications technology*, Springer.
- [23] C. Politowski, F. Petrillo, G. C. Ullmann, J. De Andrade Werly, and Y. G. Guéhéneuc, "Dataset of video game development problems," *Proceedings-2020 IEEE/ACM 17th International Conference on Mining Software Repositories, MSR 2020*, 2020.
- [24] S. S. Bajwa, X. Wang, A. Nguyen Duc, and P. Abrahamsson, "Failures to be celebrated: An analysis of major pivots of software startups," *Empirical Software Engineering*, vol. 22, 2017.
- [25] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Identifying gameplay videos that exhibit bugs in computer games," *Empirical Software Engineering*, 2019.
- [26] N. A. Butt, S. Sherin, A. A. Jilani, M. U. Khan, and M. Z. Iqbal, *Data for Deriving and Evaluating a Detailed Taxonomy of Game Bugs*, version 2.0.0, Oct. 2023. DOI: [10.5281/zenodo.8425685](https://doi.org/10.5281/zenodo.8425685). [Online]. Available: <https://doi.org/10.5281/zenodo.8425685>.
- [27] I. ISO and N. IEC, "Iso/iec," *IEEE International Standard-Systems and software engineering-Vocabulary-24765-2017*, pp. 1–541, 2017.
- [28] J. Schell, *The Art of Game Design: A book of lenses*. CRC press, 2008.
- [29] L. Mangia and R. Paiano, "Challenges for network computer games," in *Proceedings of the IADIS International Conference*, IADIS, 2004.

- [30] R. Zubek, *Elements of game design*. MIT Press, 2020.
- [31] M. Unterkalmsteiner, T. Gorschek, A. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt, “Evaluation and measurement of software process improvement—a systematic literature review,” *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 398–424, 2011.
- [32] T. Punter, M. Ciolkowski, B. Freimut, and I. John, “Conducting on-line surveys in software engineering,” in *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, IEEE, 2003, pp. 80–88.
- [33] S. Brown, *Likert scale examples for surveys*, 2010.
- [34] X. Zhao, X. Xia, P. S. Kochhar, D. Lo, and S. Li, “An empirical study of bugs in build process,” in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1187–1189.
- [35] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, “Bug report severity level prediction in open source software: A survey and research opportunities,” *Information and software technology*, vol. 115, pp. 58–78, 2019.
- [36] P. Chakraborty, R. Shahriyar, A. Iqbal, and A. Bosu, “Understanding the software development practices of blockchain projects: A survey,” in *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, 2018, pp. 1–10.
- [37] S. Imtiaz and N. Ikram, “Framework for task allocation in global software development,” *IEEE Access*, vol. 8, pp. 206 235–206 247, 2020.
- [38] A. M. Albaghajati and M. A. K. Ahmed, “Video game automated testing approaches: An assessment framework,” *IEEE Transactions on Games*, 2020.
- [39] L. Briand and Y. Labiche, “Empirical studies of software testing techniques: Challenges, practical strategies, and future research,” *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–3, 2004.
- [40] D. L. Lough, *A taxonomy of computer attacks with applications to wireless networks*. Virginia Polytechnic Institute and State University, 2001.
- [41] M. Washburn, P. Sathiyarayanan, M. Nagappan, T. Zimmermann, and C. Bird, ““what went right and what went wrong”: An analysis of 155 postmortems from game development,” in *IEEE/ACM 38th IEEE International Conference on Software Engineering Companion*.
- [42] C. Redavid and A. Farid, “An overview of game testing techniques,” 2011.
- [43] Z. Li, Y. Wu, L. Ma, X. Xie, Y. Chen, and C. Fan, “Gbgallery: A benchmark and framework for game testing,” *Empirical Software Engineering*, vol. 27, no. 6, pp. 1–27, 2022.

Selected Studies

- [S1] J. Hernández Bécares, L. Costero Valero, and P. P. Gómez Martín, “An approach to automated videogame beta testing,” *Entertainment Computing*, vol. 18, 2017.
- [S2] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, “An automated model based testing approach for platform games,” in *2015 ACM/IEEE MODELS 2015-Proceedings*.
- [S3] J. Pfau, A. Liapis, G. N. Yannakakis, and R. Malaka, “Dungeons and replicants ii: Automated game balancing across multiple difficulty dimensions via deep player behavior modeling,” *IEEE Transactions on Games*, 2022.

- [S4] S. Shirzadehhajimahmood, I. Prasetya, F. Dignum, and M. Dastani, “An online agent-based search approach in automated computer game testing with model construction,” in *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*, 2022, pp. 45–52.
- [S5] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslen, “Augmenting automated game testing with deep reinforcement learning,” *IEEE Conference on Computational Intelligence and Games, CIG*, vol. 2020-August, 2020.
- [S6] Y. Zheng, X. Xie, T. Su, *et al.*, “Wuji : Automatic online combat game testing using evolutionary deep reinforcement learning,” in *34th ACM/IEEE International Conference on Automated Software Engineering*, pp. 772–784.
- [S7] M. Schatten, B. O. Duric, I. Tomicic, and N. Ivkovic, “Automated mmorpg testing—an agent-based approach,” in *International conference on practical applications of agents and multi-agent system*, Y. Demazeau, P. Davidsson, J. Bajo, and Z. Vale, Eds., vol. 10349, Springer International Publishing.
- [S8] J. Pfau, J. D. Smeddinck, and R. Malaka, “Automated game testing with icarus: Intelligent completion of adventure riddles via unsupervised solving,” *CHI PLAY 2017 Extended Abstracts-Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, pp. 153–163, 2017.
- [S9] K. Chang, B. Aytemiz, and A. M. Smith, “Reveal-more: Amplifying human effort in quality assurance testing using automated exploration,” in *2019 IEEE Conference on Games(CoG)*, vol. 2019-Augus.
- [S10] R. Ferdous, F. Kifetew, D. Prandi, I. S. W. B. Prasetya, S. Shirzadehhajimahmood, and A. Susi, “Search-based automated play testing of computer games: A model-based approach,” 2021.
- [S11] K. Chen, Y. Li, Y. Chen, C. Fan, Z. Hu, and W. Yang, “Glib: Towards automated test oracle for graphically-rich applications,” in *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Association for Computing Machinery(ACM).
- [S12] C. J. Schaefer and H. Do, “Model-based exploratory testing: A controlled experiment,” in *Proceedings-IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014*, IEEE Computer Society, pp. 284–293.
- [S13] P. Lima, E. Guerra, N. S. Pereira, and P. Meirelles, “Towards automated playtesting in game development,” in *Proceedings of SB*.
- [S14] P. García-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, “Automated playtesting in collectible card games using evolutionary algorithms: A case study in hearth-stone,” *Knowledge-Based Systems*, vol. 153, 2018.
- [S15] L. K. Chen, Y. H. Chen, S. F. Chang, and S. C. Chang, “A long/short-term memory based automated testing model to quantitatively evaluate game design,” *Applied Sciences(Switzerland)*, vol. 10, no. 19, 2020.
- [S16] S. Stahlke, A. Nova, and P. Mirza-Babaei, “Artificial playfulness: A tool for automated agent-based playtesting,” in *Conference on Human Factors in Computing Systems-Proceedings*.
- [S17] Z. Song, “An automated framework for gaming platform to test multiple games,” in *ICSE-Companion*.

- [S18] S. Varvaressos, K. Lavoie, A. B. Massé, S. Gaboury, and S. Hallé, “Automated bug finding in video games: A case study for runtime monitoring,” *Proceedings-IEEE 7th International Conference on Software Testing, Verification and Validation, ICST 2014*, pp. 143–152, 2014.
- [S19] J. Tuovenen, M. Oussalah, and P. Kostakos, “Mauto: Automatic mobile game testing tool using image-matching based approach,” *The Computer Games Journal*, vol. 8, 2019.
- [S20] S. Ariyurek, A. Betin-Can, and E. Surer, “Automated video game testing using synthetic and humanlike agents,” *IEEE Transactions on Games*, vol. 13, no. 1, pp. 50–67, 2019.
- [S21] S. Ariyurek, A. Betin-Can, and E. Surer, “Enhancing the monte carlo tree search algorithm for video game testing,” in *IEEE Conference on Computational Intelligence and Games, CIG*.
- [S22] M. R. Taesiri, M. Habibi, and M. Fazli, “A video game testing method utilizing deep learning,” *The CSI Journal on Computer Science and Engineering*, vol. 17, no. 2, pp. 26–33, 2020.
- [S23] Y. Choi, H. Kim, C. Park, and S. Jin, “A case study: Online game testing using massive virtual player,” *Communications in Computer and Information Science*, vol. 256 CCIS, 2011.
- [S24] C. Guerrero-Romero, S. M. Lucas, and D. Perez-Liebana, “Using a team of general ai algorithms to assist game design and testing,” in *2018 IEEE Conference on Computational Intelligence and Games(CIG)*, IEEE, pp. 1–8.
- [S25] C. S. Cho, K. M. Sohn, C. J. Park, and J. H. Kang, “Online game testing using scenario-based control of massive virtual users,” *International Conference on Advanced Communication Technology, ICACT*, vol. 2, 2010.
- [S26] S. Liu, L. Chaoran, L. Yue, *et al.*, “Automatic generation of tower defense levels using pcg,” in *International Conference on the Foundations of Digital Games, ICST*.
- [S27] C. S. Cho, D. C. Lee, K. M. Sohn, C. J. Park, and J. H. Kang, “Scenario-based approach for blackbox load testing of online game servers,” *Proceedings-2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2010*, pp. 259–265, 2010.
- [S28] A. Liaqat, M. A. Sindhu, and G. F. Siddiqui, “Metamorphic testing of an artificially intelligent chess game,” *IEEE Access*, vol. 8, 2020.
- [S29] W.-k. Chen, C.-h. Liu, and P.-h. Chen, “A game framework supporting automatic functional testing for games,” *Applied Computing and Information Technology*, vol. 727, 2016.
- [S30] A. Nantes, R. Brown, and F. Maire, “A framework for the semi-automatic testing of video games,” *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008.
- [S31] Y. Wu, Y. Chen, X. Xie, B. Yu, C. Fan, and L. Ma, “Regression testing of massively multi-player online role-playing games,” in *Proceedings-2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020*, Institute of Electrical and Electronics Engineers Inc., pp. 692–696.
- [S32] M. Mozgovoy and E. Pyshkin, “Unity application testing automation with appium and image recognition,” in *Communications in Computer and Information Science*, vol. 779, Springer, Cham, pp. 139–150.

- [S33] B. Chan, J. Denzinger, D. Gates, K. Loose, and J. Buchanan, “Evolutionary behavior testing of commercial computer games,” *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, 2004.
- [S34] M. Shaker, N. Shaker, and J. Toglius, “Evolving playable content for cut the rope through a simulation-based approach,” in *Proceedings / the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AAAI Press, pp. 72–78.
- [S35] Y. Shin, J. Kim, K. Jin, and Y. B. Kim, “Playtesting in match 3 game using strategic plays via reinforcement learning,” *IEEE Access*, vol. 8, 2020.
- [S36] G. Xiao, F. Southey, R. C. Holte, and D. Wilkinson, “Software testing by active learning for commercial games,” *Proceedings of the National Conference on Artificial Intelligence*, vol. 2, pp. 898–903, 2005.
- [S37] H. Zhao, J. Sun, and G. Hu, “Study of methodology of testing mobile games based on ttcn-3,” in *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, IEEE.
- [S38] M. P. Krieger and A. D. Brucker, “Ocl-based runtime monitoring of jvm hosted applications,” vol. 44, 2011.
- [S39] T. Ahumada and A. Bergel, “Reproducing bugs in video games using genetic algorithms,” in *IEEE Games, Multimedia, Animation and Multiple Realities Conference(GMAX)*, á, ISBN: 9781728161471.
- [S40] P. Gutiérrez-Sánchez, M. A. Gómez-Martín, P. A. González-Calero, and P. P. Gómez-Martín, “Reinforcement learning methods to evaluate the impact of ai changes in game design,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 17, no. 1, 2021.
- [S41] A. Nantes, R. Brown, and F. Maire, “Measuring visual consistency in 3d rendering systems,” in *Proceedings of the Thirty-Third Australasian Computer Science Conference*, pp. 43–52.
- [S42] S. N. Stahlke and P. Mirza-Babaei, “Ustesting without the user: Opportunities and challenges of an ai-driven approach in games user research,” *Computers in Entertainment*, vol. 16, no. 2, 2018.
- [S43] X. Li, D. Zhou, L. Zhang, and Y. Jing, “Human-like ui automation through automatic exploration,” in *International Conference on Big Data and Artificial Intelligence*, ICST.
- [S44] S. Varvaressos, D. Vaillancourt, S. Gaboury, A. B. Massé, and S. Hallé, “Runtime monitoring of temporal logic properties in a platform game,” in *International Conference on Runtime Verification*.
- [S45] A. Kumar, S. Kumar Purushothaman, V. Singh, A. Bharti, and N. Kashyap, “Multi-game automation approach for cocos-2dx based card game,” in *International Conference on Computing, Communication and Networking Technologies(ICCNT)*.
- [S46] B. Wilkins, C. Watkins, and K. Stathis, “A metric learning approach to anomaly detection in video games,” in *IEEE Conference on Computational Intelligence and Games, CIG*, vol. 2020-August, IEEE Computer Society, pp. 604–607.
- [S47] A. Kazmi, A. Fatima, A. Idris, and S. Hussain, “Adaptive usage statistical testing for 3d gaming applications,” *2018 International Conference on Computing, Electronic and Electrical Engineering, ICE Cube 2018*, pp. 1–6, 2018.

- [S48] C. Paduraru and M. Paduraru, “Automatic difficulty management and testing in games using a framework based on behavior trees and genetic algorithms,” in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, vol. 2019-November, Institute of Electrical and Electronics Engineers Inc., pp. 170–179.
- [S49] E. J. Powley, S. Colton, S. Gaudl, R. Saunders, and M. J. Nelson, “Semi-automated level design via auto-playtesting for handheld casual game creation,” in *IEEE Conference on Computational Intelligence and Games*.
- [S50] Y. Jung, B.-H. Lim, K.-H. Sim, *et al.*, “Venus: The online game simulator using massively virtual clients,” in *Asian Simulation Conference*.
- [S51] M. Morosan and R. Poli, “Lessons from testing an evolutionary automated game balancer in industry,” in *IEEE Games, Entertainment, Media Conference(GEM)*, ISBN: 9781538663042.
- [S52] C. Bell and M. Goadrich, “Automated playtesting with recycled cardstock,” *Game and Puzzle Design*, 2016.
- [S53] A. Toumi, J. Gutierrez, and M. Wooldridge, “A tool for the automated verification of nash equilibria in concurrent games,” in *International Colloquium on Theoretical Aspects of Computing*, M. Leucker, C. Rueda, and F. D. Valencia, Eds., vol. 9399, Springer International Publishing, pp. 583–594.
- [S54] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. P. Popović, “Evaluating competitive game balance with restricted play,” in *Artificial Intelligence and Interactive Digital Entertainment Conference*.
- [S55] M. Preuss, T. Pfeiffer, V. Volz, and N. Pflanzl, “Integrated balancing of an rts game: Case study and toolbox refinement,” in *IEEE Conference on Computational Intelligence and Games(CIG)*.
- [S56] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, “Exploring game space of minimal action games via parameter tuning and survival analysis,” *IEEE Transactions on Games*, vol. 10, no. 2, 2018.
- [S57] M. Morosan and R. Poli, “Automated game balancing in ms pacman and starcraft using evolutionary algorithms,” in *European Conference on the Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds., vol. 10199, Springer International Publishing.
- [S58] C. Paduraru, M. Paduraru, and A. Stefanescu, “Rivergame-a game testing tool using artificial intelligence,” pp. 422–432, 2022.
- [S59] C. Lu, R. Georgescu, and J. Verwey, “Go-explore complex 3d game environments for automated reachability testing,” *IEEE Transactions on Games*, 2022.
- [S60] R. Tufano, S. Scalabrino, L. Pascarella, E. Aghajani, R. Oliveto, and G. Bavota, “Using reinforcement learning for load testing of video games,” pp. 2303–2314, 2022.
- [S61] Y. Zhao, E. Tang, H. Cai, X. Guo, X. Wang, and N. Meng, “A lightweight approach of human-like playtest for android apps,” pp. 309–320, 2022.
- [S62] P. Feldmeier and G. Fraser, *Neuroevolution-based generation of tests and oracles for games*, 2022.
- [S63] G. Liu, M. Cai, L. Zhao, *et al.*, “Inspector: Pixel-based automated game testing via exploration, detection, and investigation,” pp. 237–244, 2022.
- [S64] D. A. DeLaurentis, J. H. Panchal, A. K. Raz, *et al.*, “Toward automated game balance: A systematic engineering design approach,” pp. 1–8, 2021.
- [S65] P. M. Negrao, *Automated playtesting in videogames*, 2020.

- [S66] Sriram and Varun, *Automated playtesting of platformer games using reinforcement learning*, Generic, 2019.
- [S67] D. Loubos, *Automated testing in virtual worlds*, 2018. [Online]. Available: https://doi.org/10.475/123_4.
- [S68] S. R. Dandey, *An automated testing tool for the virtual cell game*, 2013.
- [S69] B. Steffan Derek Hooper, *Automated testing and validation of computer graphics implementations for cross-platform game development*, 2017.
- [S70] A. Nikas, *Automated gui testing for games using pseudo-dsl*, 2015.
- [S71] H. Kljajic and O. Karlsson, *Applying automated testing in an existing client-server game a pursuit for fault localization in quake 3*, 2015.
- [S72] A. Sestini, L. Gisslén, J. Bergdahl, K. Tollmar, and A. D. Bagdanov, *Ccpt: Automatic gameplay testing and validation with curiosity-conditioned proximal trajectories*, 2022.
- [S73] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, *Improving playtesting coverage via curiosity driven reinforcement learning agents*, Mar. 2021.
- [S74] N. John and J. Gow, *Adversarial behaviour debugging in a two button fighting game*, Generic, 2021.
- [S75] A. Jonasson and J. Nilsson, *Using artificial intelligence for gameplay testing on turn-based games*, 2018.
- [S76] C. Lewis and J. Whitehead, “Repairing games at runtime or, how we learned to stop worrying and love emergence,” *IEEE Software*, vol. 28, no. 5, 2011.
- [S77] A. Zook, E. Fruchter, and M. O. Riedl, “Automatic playtesting for game parameter tuning via active learning,” 2019.
- [S78] M. Zuo, L. Schick, M. Gombolay, and N. Gopalan, “Efficient exploration via first-person behavior cloning assisted rapidly-exploring random trees,” *arXiv preprint arXiv:2203.12774*, 2022.

Selected Postmortems

- [PM1] *Activisions heavy gear 2*, 1999. [Online]. Available: <https://bit.ly/3rqbeBL>.
- [PM2] *An indie-style experiment at a aaa studio: Insomniacs slow down, bull*, 2015. [Online]. Available: <https://bit.ly/34egfol>.
- [PM3] *Another one bites the dust: Dynasty feud postmortem*, 2018. [Online]. Available: <https://bit.ly/3IZig6A>.
- [PM4] *Appy entertainments spellcraft school of magic*, 2012. [Online]. Available: <https://bit.ly/3IZigU8>.
- [PM5] *Baldurs gate ii the anatomy of a sequel*, 2001. [Online]. Available: <https://bit.ly/3IZihaE>.
- [PM6] *Blizzards diablo ii*, 2000. [Online]. Available: <https://bit.ly/3HsdMoJ>.
- [PM7] *Blue fangs zoo tycoon 2 marine mania*, 2007. [Online]. Available: <https://bit.ly/3Lc075Q>.
- [PM8] *Blue tongue softwares jurassic park: Operation genesis*, 2003. [Online]. Available: <https://bit.ly/34xVWSq>.

- [PM9] *Collaborating on an indie sensation-cuphead*, 2018. [Online]. Available: <https://bit.ly/3uqcAyC>.
- [PM10] *Disney onlines toontown*, 2003. [Online]. Available: <https://bit.ly/3Grf167>.
- [PM11] *Dreamforges sanitarium*, 1998. [Online]. Available: <https://bit.ly/3rpbhxK>.
- [PM12] *Dreamworks interactives trespasser*, 1999. [Online]. Available: <https://bit.ly/3rqWp24>.
- [PM13] *Drinkbox studios guacamelee!* 2013. [Online]. Available: <https://bit.ly/34lqfvU>.
- [PM14] *Ensemble studios age of empires ii: Age of kings*, 2000. [Online]. Available: <https://bit.ly/350fMd0>.
- [PM15] *Freeverses marathon 2: Durandal*, 2007. [Online]. Available: <https://bit.ly/34fmus0>.
- [PM16] *Freeverses top gun for iphone*, 2009. [Online]. Available: <https://bit.ly/3ooA1xB>.
- [PM17] *Games kitchens wireless pets*, 2002. [Online]. Available: <https://bit.ly/3op8p26>.
- [PM18] *Hero-u: Rogue to redemption postmortem*, 2019. [Online]. Available: <https://bit.ly/3opx3zt>.
- [PM19] *Ion storms deus ex*, 2000. [Online]. Available: <https://bit.ly/3AXGj2P>.
- [PM20] *Ironclad/stardocks sins of a solar empire*, 2008. [Online]. Available: <https://bit.ly/3oo9M0Y>.
- [PM21] *Irrational games system shock 2*, 1999. [Online]. Available: <https://bit.ly/3Hx4TtZ>.
- [PM22] *Lucas arts star wars starfighter*, 2001. [Online]. Available: <https://bit.ly/3Hr9QEet>.
- [PM23] *Mcmillen and himsls the binding of isaac*, 2012. [Online]. Available: <https://bit.ly/3GEbced>.
- [PM24] *Micro forte s fallout tactics*, 2001. [Online]. Available: <https://bit.ly/34h3yJs>.
- [PM25] *Mind controls oasis*, 2005. [Online]. Available: <https://bit.ly/3umseuz>.
- [PM26] *Mode 7 games frozen synapse*, 2012. [Online]. Available: <https://bit.ly/3ooWVLY>.
- [PM27] *Multitude fireteam*, 1999. [Online]. Available: <https://bit.ly/3gnnyMX>.
- [PM28] *Nayantaras star chamber*, 2003. [Online]. Available: <https://bit.ly/3onQgl0>.
- [PM29] *Ngames chop suey kung fu*, 2001. [Online]. Available: <https://bit.ly/3G1DR7n>.
- [PM30] *Ninjabees a kingdom for keflings*, 2009. [Online]. Available: <https://bit.ly/3sgtM6P>.
- [PM31] *Operation flashpoint*, 2001. [Online]. Available: <https://bit.ly/3GqF1yw>.
- [PM32] *Over the top games nyxquest kindred spirits*, 2010. [Online]. Available: <https://bit.ly/3rsH9lk>.
- [PM33] *Pangalores knightly adventure*, 2013. [Online]. Available: <https://bit.ly/3Lc07mm>.
- [PM34] *Pipeworks softwares deadliest warrior*, 2010. [Online]. Available: <https://bit.ly/3seWRzE>.
- [PM35] *Pixelogics the italian job*, 2002. [Online]. Available: <https://bit.ly/3J3JwRu>.
- [PM36] *Poncho-a postmortem*, 2017. [Online]. Available: <https://bit.ly/3GtfRiR>.
- [PM37] *Postcard from gdc europe 2005: The first generation of the next generation: A project gotham racing 3 postmortem*, 2005. [Online]. Available: <https://bit.ly/3rrcSDg>.
- [PM38] *Postmortem sniper studios crazy taxi fare wars*, 2007. [Online]. Available: <https://bit.ly/3uFEA11>.

- [PM39] *Postmortem: E-line media and upper one games never alone*, 2015. [Online]. Available: <https://bit.ly/3rocBRE>.
- [PM40] *Postmortem: Presto studios star trek: Hidden evil*, 1999. [Online]. Available: <https://bit.ly/34lqfMq>.
- [PM41] *Prestos whacked!* 2002. [Online]. Available: <https://bit.ly/3AXCWJo>.
- [PM42] *Raven softwares heretic ii*, 1999. [Online]. Available: <https://bit.ly/3roKNMJ>.
- [PM43] *Realtime worlds crackdown*, 2009. [Online]. Available: <https://bit.ly/3LbMaXh>.
- [PM44] *Redstorms rainbow six*, 2000. [Online]. Available: <https://bit.ly/3rmDTI2>.
- [PM45] *Rpublique*, 2016. [Online]. Available: <https://bit.ly/3uoqCAB>.
- [PM46] *Saber interactives timeshift*, 2008. [Online]. Available: <https://bit.ly/35HToS1>.
- [PM47] *Sierra studios gabriel knight 3*, 2000. [Online]. Available: <https://bit.ly/3goEkeI>.
- [PM48] *Sony bend studios uncharted: Golden abyss*, 2012. [Online]. Available: <https://bit.ly/3uqcJSG>.
- [PM49] *Stardock entertainment and ironclad games sins of a solar empire: Rebellion*, 2012. [Online]. Available: <https://bit.ly/34bNF7c>.
- [PM50] *Stardocks galactic civilizations 2: Dread lords*, 2006. [Online]. Available: <https://bit.ly/3smLB4m>.
- [PM51] *Stardocks the political machine*, 2004. [Online]. Available: <https://bit.ly/35HTpFT>.
- [PM52] *Starting from scratch: Haemimont games tropico 5 postmortem*, 2014. [Online]. Available: <https://bit.ly/3J8n21P>.
- [PM53] *Student postmortem: 6msofts romeo and juliet*, 2005. [Online]. Available: <https://bit.ly/3L916Ft>.
- [PM54] *Surreal software drakan order of the flame*, 1999. [Online]. Available: <https://bit.ly/3G18uKp>.
- [PM55] *Team meats super meat boy*, 2011. [Online]. Available: <https://bit.ly/3GqF1P2>.
- [PM56] *The chinese rooms amnesia: A machine for pigs*, 2014. [Online]. Available: <https://bit.ly/3utZZKz>.
- [PM57] *The game design of surrealsthe suffering*, 2004. [Online]. Available: <https://bit.ly/3oqHt21>.
- [PM58] *Tom clancys splinter cell*, 2003. [Online]. Available: <https://bit.ly/3scvXbD>.
- [PM59] *Torpex games' schizoid*, 2008. [Online]. Available: <https://bit.ly/3J3va3k>.
- [PM60] *Ty the tasmanian tiger 2 for the younger market*, 2005. [Online]. Available: <https://bit.ly/3IWQhnY>.
- [PM61] *Vampire the masquerade redemption*, 2000. [Online]. Available: <https://bit.ly/3upjuDX>.
- [PM62] *Zachtronics industries spacechem*, 2012. [Online]. Available: <https://bit.ly/3sj0CUr>.

Selected Talk Sessions

- [T1] B. Foddy, *Designing with physics-bend the physics engine to your will(game developers conference(gdc))*, 2015. [Online]. Available: <https://bit.ly/3sjxTio>.
- [T2] R. Saint-Pierre, *Time where it matters-friction free bug reporting(game developers conference(gdc))*, 2013. [Online]. Available: <https://bit.ly/3Gp7dSr>.

- [T3] B. Anguelov and B. Sunzhine-Hill, *Managing the movement-getting your animation behaviors to behave better(game developers conference(gdc))*, 2013. [Online]. Available: <https://bit.ly/3ooeSdL>.
- [T4] M. Madej, *Challenges of designing multiplayer games(digital dragons)*, 2014. [Online]. Available: <https://bit.ly/3opMg3x>.
- [T5] B. Żywicznyński, *Upgrading the production toolbelt from this war of mine to frostpunk(digital dragons)*, 2018. [Online]. Available: <https://bit.ly/3J7j6hL>.

Selected Web Sources

- [W1] *Game-breaking glitches*, (W). [Online]. Available: <https://bit.ly/3okWTF2>.
- [W2] *Game breaking bug*, (W). [Online]. Available: <https://bit.ly/3GkZu7W>.
- [W3] *Metroid dread has a game-breaking bug: Here's how to avoid it*, (N). [Online]. Available: <https://bit.ly/3GqiJgy>.
- [W4] *10 infamous game breaking bugs in famous video games*, (B). [Online]. Available: <https://bit.ly/3unigcz>.
- [W5] *5 worst game-breaking bugs of all time*, (B). [Online]. Available: <https://bit.ly/3IWRbRo>.
- [W6] *Pokémon unite politely asks players to stop using game-breaking bug*, (B). [Online]. Available: <https://bit.ly/3HrOLvG>.
- [W7] *Halo infinite has a save game-breaking bug that needs to be fixed*, (B). [Online]. Available: <https://bit.ly/3skm23T>.
- [W8] *Destiny 2 game-breaking glitch surfaces, bungie starts handing out bans*, (B). [Online]. Available: <https://bit.ly/3goNjwo>.
- [W9] *Game breaking bug in apex legends makes players invulnerable while reviving*, (N). [Online]. Available: <https://bit.ly/3sguMrB>.
- [W10] *New world's latest patch has brought 2 game-breaking bugs to healing and the auction house*, (N). [Online]. Available: <https://bit.ly/3gqo6Sh>.
- [W11] *Assassin's creed: Valhalla has game-breaking bug four months after release despite recent update*, (N). [Online]. Available: <https://bit.ly/3GEc5n3>.
- [W12] *Call of duty: Warzone's biggest game-breaking bug is back*, (B). [Online]. Available: <https://bit.ly/3LcPb9Q>.
- [W13] *'cyberpunk 2077' update introduced a game-breaking bug*, (B). [Online]. Available: <https://engt.co/3oqjpfz>.
- [W14] *Three game-breaking bugs still present in the game*, (B). [Online]. Available: <https://bit.ly/32W0505>.
- [W15] *Vanguard players demand fix to 'game-breaking' dead drop glitch*, (B). [Online]. Available: <https://bit.ly/3ATWyy4>.
- [W16] *Game breaking bug on war!* (F). [Online]. Available: <https://bit.ly/3gkuAC4>.
- [W17] *New world invincibility exploit discovered and it's game-breaking*, (N). [Online]. Available: <https://bit.ly/3GvyRNE>.
- [W18] *"game-breaking" apex legends bug is making ash's phase ultimate useless*, (N). [Online]. Available: <https://bit.ly/3AVdsMC>.

- [W19] *Possible game-breaking bug discovered in fallout 4*, (N). [Online]. Available: <https://bit.ly/3umttKf>.
- [W20] *Game-breaking bug in soma safe mode(epic store)*, (F). [Online]. Available: <https://bit.ly/3AUIvs9>.

Selected Video Sources

- [V1] *This is the most game breaking bug i've ever seen-(youtube)*. [Online]. Available: <https://bit.ly/3GlF7r7>.
- [V2] *10 recent worst game breaking glitches-(youtube)*. [Online]. Available: <https://bit.ly/3AYTJvq>.
- [V3] *Top 10 worst game breaking glitches-(youtube)*. [Online]. Available: <https://bit.ly/3AYTKzu>.
- [V4] *Top 10 worst game breaking glitches-(youtube)*. [Online]. Available: <https://bit.ly/3Lb60Sa>.
- [V5] *New world game breaking bug needs to be fixed asap!-(youtube)*. [Online]. Available: <https://bit.ly/34xq40y>.
- [V6] *Massive game breaking glitch in fortnite!-(youtube)*. [Online]. Available: <https://bit.ly/3LcPqlg>.
- [V7] *Blizzard has to fix this game breaking bug now!-(youtube)*. [Online]. Available: <https://bit.ly/3rzRfAZ>.
- [V8] *5 game breaking glitches that bricked your console — fact hunt — larry bundy jr-(youtube)*. [Online]. Available: <https://bit.ly/3AWESSk>.
- [V9] *Fallout 4 game-breaking bug discovered-gs news update-(youtube)*. [Online]. Available: <https://bit.ly/34lr0dg>.
- [V10] *Top 5 game breaking bugs and exploits in overwatch-(youtube)*. [Online]. Available: <https://bit.ly/3uFG8sb>.
- [V11] *New world game breaking bug deletes hours of gameplay!-(youtube)*. [Online]. Available: <https://bit.ly/3omVy08>.
- [V12] *New game breaking bug is truly awful... maybe don't play ranked rn-pokemon unite-(youtube)*. [Online]. Available: <https://bit.ly/3LbNkSD>.
- [V13] *Wtf new game breaking bug-unlimited gold abuse glitch-7.29 patch dota 2-(youtube)*. [Online]. Available: <https://bit.ly/3gtJXrL>.
- [V14] *Fallout 76's latest update adds in a major game breaking bug-(youtube)*. [Online]. Available: <https://bit.ly/3ro7aSI>.
- [V15] *The most game breaking bug of all time-(youtube)*. [Online]. Available: <https://bit.ly/3L8Q1UV>.
- [V16] *Season 11 gamebreaking bug #954-(youtube)*. [Online]. Available: <https://bit.ly/3B0DjCT>.
- [V17] *The “game breaking” bugs that closed classic-(youtube)*. [Online]. Available: <https://bit.ly/3Gr8rwo>.
- [V18] *Game breaking bug gets you banned in destiny 2!-(youtube)*. [Online]. Available: <https://bit.ly/3GpP0Ju>.

- [V19] *There is a game breaking glitch in sword and shield...-(youtube)*. [Online]. Available: <https://bit.ly/3J3wovs>.
- [V20] *Minecraft's most game breaking glitches...-(youtube)*. [Online]. Available: <https://bit.ly/36pvj37>.
- [V21] *Is it worse post 1.1 patch?-game breaking bugs and glitches-cyberpunk 2077(down on the street)-(youtube)*. [Online]. Available: <https://bit.ly/3BbbZCh>.
- [V22] *Bug that crashed lcs game! taliyah/karthus gamebreaking bug investigation!-(youtube)*. [Online]. Available: <https://bit.ly/3siXxno>.
- [V23] *Game-breaking glitch discovered in paper mario: The origami king. here's how to avoid it!-(youtube)*. [Online]. Available: <https://bit.ly/3HtfyWE>.
- [V24] *The wheelchair paladin [gamebreaking bug]-(youtube)*. [Online]. Available: <https://bit.ly/3sj2yff>.

Appendix

A Quality Assessment

This appendix contains the table with detailed quality assessment of selected academic studies.

Table A.1: Quality Assessment of Selected Academic Literature

ID	QC1	QC2	QC3	QC4	QC5	QC6	QC7	Quality Score
S1	✓	✓	✓	✓	✓	✓	X	6
S2	✓	✓	✓	✓	✓	✓	✓	7
S3	✓	✓	✓	✓	✓	✓	✓	7
S4	✓	✓	✓	✓	✓	✓	✓	7
S5	✓	✓	✓	✓	✓	✓	✓	7
S6	✓	✓	✓	✓	✓	✓	✓	7
S7	✓	✓	✓	X	✓	✓	X	5
S8	✓	✓	✓	✓	✓	✓	✓	7
S9	✓	✓	✓	X	✓	✓	X	5
S10	✓	✓	✓	X	✓	✓	X	5
S11	✓	X	X	✓	✓	✓	✓	5
S12	✓	✓	✓	✓	✓	✓	✓	7
S13	✓	✓	✓	✓	✓	✓	X	6
S14	✓	✓	✓	X	✓	✓	X	5
S15	✓	✓	✓	X	✓	✓	X	5
S16	✓	✓	✓	X	X	X	X	3
S17	✓	✓	✓	X	X	X	X	3
S18	✓	✓	✓	✓	✓	✓	✓	7
S19	✓	✓	✓	✓	✓	✓	X	6
S20	✓	✓	✓	✓	✓	✓	✓	7
S21	✓	✓	✓	✓	✓	✓	✓	7
S22	✓	✓	✓	✓	✓	✓	✓	7
S23	✓	✓	✓	X	✓	✓	X	5
S24	✓	✓	✓	✓	✓	✓	X	6
S25	✓	✓	✓	✓	✓	✓	✓	7
S26	✓	✓	✓	✓	✓	✓	X	6
S27	✓	✓	✓	X	✓	✓	✓	6
S28	✓	✓	✓	✓	✓	✓	✓	7
S29	✓	✓	✓	✓	✓	✓	X	6
S30	✓	✓	✓	✓	✓	✓	✓	7
S31	✓	✓	✓	✓	✓	✓	✓	7
S32	✓	✓	✓	✓	✓	✓	X	6
S33	✓	✓	✓	X	✓	✓	X	5
S34	✓	✓	✓	✓	✓	✓	✓	7
S35	✓	✓	✓	✓	✓	✓	X	6
S36	✓	✓	✓	✓	✓	✓	X	6
S37	✓	✓	✓	✓	✓	✓	X	6
S38	✓	✓	✓	✓	✓	✓	✓	7
S39	✓	✓	✓	✓	✓	✓	✓	7
S40	✓	✓	✓	✓	✓	✓	X	6
S41	✓	X	X	✓	✓	✓	✓	5
S42	✓	✓	X	X	✓	✓	X	4
S43	✓	✓	✓	X	✓	✓	X	5
S44	✓	✓	✓	✓	✓	✓	✓	7
S45	✓	✓	✓	✓	✓	✓	X	6
S46	✓	X	X	✓	✓	✓	✓	5
S47	✓	✓	✓	✓	X	X	X	4
S48	✓	✓	✓	X	✓	✓	X	5
S49	✓	✓	✓	X	✓	✓	X	5
S50	✓	✓	✓	X	✓	✓	X	5
S51	✓	✓	✓	✓	✓	✓	X	6
S52	✓	✓	✓	✓	✓	✓	X	6
S53	✓	✓	✓	✓	✓	✓	X	6
S54	✓	✓	✓	✓	✓	✓	X	6
S55	✓	✓	✓	✓	✓	✓	X	6
S56	✓	✓	✓	✓	✓	✓	X	6
S57	✓	✓	✓	✓	✓	✓	X	6
S58	✓	✓	✓	✓	✓	✓	✓	7
S59	✓	✓	✓	✓	✓	✓	✓	7
S60	✓	✓	✓	✓	✓	✓	✓	7
S61	✓	✓	✓	✓	✓	✓	X	6

Table A.1 continued from previous page

ID	QC1	QC2	QC3	QC4	QC5	QC6	QC7	Quality Score
S62	✓	✓	✓	✓	✓	✓	✓	7
S63	✓	✓	✓	✓	✓	✓	✓	7
S64	✓	✓	✓	✓	✓	✓	✓	7
S65	✓	✓	✓	✓	✓	✓	✓	7
S66	✓	✓	✓	X	✓	✓	X	5
S67	✓	✓	✓	✓	✓	✓	X	6
S68	✓	✓	✓	✓	✓	✓	X	6
S69	✓			✓	✓	✓	✓	5
S70	✓	✓	✓	✓	✓	✓	✓	7
S71	✓	✓	✓	✓	✓	✓	✓	7
S72	✓	✓	✓	✓	✓	✓	✓	7
S73	✓	✓	✓	✓	✓	✓	✓	7
S74	✓	✓	✓	X	✓	✓	✓	6
S75	✓	✓	✓	X	✓	✓	✓	6
S76	✓	✓	✓	✓	✓	✓	✓	7
S77	✓	✓	✓	X	✓	X	X	4
S78	✓	✓	✓	X	✓	X	X	4
%	100	94.87	93.59	74.36	96.16	93.59	48.72	85.9

B Taxonomy Details

Table B.1: Sources for the derived game bugs main-tier categories

Bug Categories	Source Type	Sources
Gaming Balance	Academic	S3, S15, S26, S33, S34, S35, S36, S40, S48, S49, S51, S52, S53, S54, S55, S56, S57, S64, S74, S77
	Postmortems	PM1, PM2, PM14, PM16, PM20, PM24, PM25, PM26, PM29, PM30, PM32, PM34, PM51, PM53, PM56, PM57, PM62
	Talks	N/A
	Web	W1, W2, W9, W17
	Video	V7, V16
Implementation Response	Academic	S1, S2, S6, S8, S12, S13, S18, S20, S21, S24, S29, S31, S33, S38, S39, S43, S44, S49, S58, S61, S62, S63, S65, S66, S68, S70, S72, S75, S76
	Postmortems	PM4, PM7, PM12, PM21, PM22, PM23, PM39, PM42, PM45, PM47, PM54
	Talks	T1, T2, T4
	Web	W1, W2, W4, W5, W10, W13, W14, W18, W20
	Video	V2, V3, V5, V10, V13, V20
Network	Academic	S12, S23, S25, S27, S37, S45, S50, S60, S71
	Postmortems	PM3, PM6, PM10, PM14, PM17, PM21, PM27, PM28, PM31, PM43, PM46, PM49, PM52
	Talks	T4
	Web	W2
	Video	V2, V14
Sound	Academic	S58
	Postmortems	PM35, PM41
	Talks	T3
	Web	W2
	Video	N/A
Temporal	Academic	S18, S24, S38, S44, S76
	Postmortems	PM18
	Talks	T1
	Web	W2, W8
	Video	V3
Unexpected Crash	Academic	S6, S8, S11, S20, S31, S48, S67, S73, S75
	Postmortems	PM4, PM8, PM9, PM30, PM33, PM36, PM40, PM44, PM47, PM50, PM55
	Talks	T2, T5
	Web	W1, W2, W3, W4, W19
	Video	V2, V3, V4, V6, V10, V22
Navigational	Academic	S4, S5, S9, S20, S21, S34, S39, S59, S63, S65, S66, S70, S72, S73, S78
	Postmortems	PM5, PM11, PM61
	Talks	T3
	Web	W1, W2, W4
	Video	V2, V10
Non-Temporal	Academic	S1, S2, S6, S7, S8, S11, S13, S18, S19, S20, S21, S22, S24, S25, S28, S30, S32, S38, S39, S41, S43, S44, S45, S46, S49, S58, S60, S61, S62, S63, S65, S66, S67, S69, S70, S72, S73, S74, S75, S76
	Postmortems	PM2, PM8, PM11, PM12, PM13, PM15, PM19, PM21, PM23, PM37, PM38, PM47, PM48, PM49, PM54, PM56, PM58, PM59, PM60
	Talks	T1, T2, T3
	Web	W1, W2, W4, W5, W6, W8, W10, W11, W12, W15, W16
	Video	V1, V2, V3, V4, V7, V8, V10, V11, V12, V15, V16, V17, V18, V19, V23

Table B.2: Descriptions and examples of specialized new game bug categories

Categories	Parent Category	Descriptions	Game Example	Ref
Player Stuck Fault	Non Temporal Fault	Player is unable to progress in the game either due to inability to move from location or inability to complete a critical task.	In Guacamelee, the player could get locked in the fight arena.	PM13
Accelerated Response Fault	Temporal Fault	Consequence of an action is accelerated giving unfair advantage or disadvantage especially in real time games	In King's Quest IV, when Rosella is in the ogre's house and must reach the door before he catches her, the ogre travels across the screen in seconds.	W2
Delayed Response Fault	Temporal Fault	There is an unexpected delay in response to an action or a scenario. It is especially critical in real-time games.	In King's Quest IV, at some random intervals the response to Rosella and other characters would lag	W2

Table B.2 continued from previous page

Categories	Parent Category	Descriptions	Game Example	Ref
Network Lag	Network Faults	Game play experience is impacted by lagging response in network games due to network speed constraints and Packet size constraints	In Fireteam, lag would result in the players shots missing, by the time the shot has been fired the intended target has gone around a corner	PM27
Synchronization	Network Faults	Different game states are visible to different players in a multi-player game. A game event starts according to an incorrect or different time zone.	In Sins of a Solar Empire: Rebellion, partway through a match, players would find inconsistencies between the game states for different players	PM49
Connection Fault	Network Faults	Player is unable to connect with the game server.	In Fallout 76, if multiple mannequins were dressed in same outfit it resulted in the players being logged out and unable to establish a connection again.	V14
Network Overload	Network Faults	Game play experience of player is negatively impacted once number of connected players exceeds a specified amount.	<i>Wireless Pets could not even handle 10,000 users playing it simultaneously</i>	PM17
Packet Loss	Network Faults	A variety of packet loss or corruption problems that happen between game servers and client applications due to unreliable or unsustainable connection especially required for MMORPGs.	In Toontown, a variety of packet loss problems were encountered	PM10
Unreachable Locations	Navigational Faults	A player is unable to reach a location in game or a position on screen that should be accessible.	In Rastan on the Commodore 64, on the second level, it is impossible to make a jump over a flaming pit over two ropes to continue with the game progression	W2
Unexpected Paths	Navigational Faults	A player can reach a location in game or a position on screen that should be inaccessible.	In Fallout 77, players were able to reach places and levels that were still being developed.	V2
Too Easy	Gaming Balance	Players find it too easy to play a game. It is especially critical if a particular set of parameters allows for unfair advantages in multiplayer games.	In Hearthstone if a player plays <i>jerry rig carpenter</i> , it reduces the mana value needed to play cards to zero which is equivalent to handing a free win	V7
Too Hard	Gaming Balance	Players find it too difficult to finish the game. This includes difficulty in finishing a task due to obscure design.	In Fallout Tactics, there are some missions where the whole map depends on finding one key in an obscure place.	PM24
Unwinnable	Gaming Balance	Players are unable to finish the game especially if it is due to an overly competent AI or convoluted game logic or game constraints such as small timers or impossible to obtain in-game items.	In the Political Machine, the AI was too competent. When player got up to George Washington to play. The opponent AI won every state.	PM51
Corrupted Frame	Invalid Graphical Representation	Corrupted frames are visible.	<i>Elder Scrolls V: Skyrim</i> , had a number rendering problems that resulted in corrupted frames	V4
Slow Loading	Invalid Graphical Representation	Frame loading rate is too slow	<i>Mighty No.9</i> , also fell victim to slow frame rate	V8
Extra Game Asset	Invalid Graphical Representation	An in-game element that should not be displayed is visible.	In Jak X: Combat Racing, has a fault known as <i>saving glitch</i> where the save icon hangs on the game	W1
Missing Game Asset	Invalid Graphical Representation	An in-game element that should be displayed is not present.	In Donkey Kong Country II, if a player picks up the barrel as it breaks, then the avatar is seen carrying an invisible barrel	V3
Abnormal Text	Invalid Graphical Representation	Text appears at wrong location or is incomprehensible or may cover a characters or other objects.	In Pokémon Red, Blue, Green and Yellow, the text describing 'MISSINGNO' pokémon was often glitchy.	V3
Incorrect Visual State Transition	Invalid Graphical Representation	An in-game element transitions to an incorrect state	In Slow Down, Bull, Annette the bull catcher was notorious for not transitioning properly through her various tell states	PM2
Incorrect Sound	Sound Faults	The sound behind an animation is incorrect or does not match the circumstances	In Whacked, missing sound resources led to either sounds missing or wrong sounds being played behind animations	PM41

Table B.2 continued from previous page

Categories	Parent Category	Descriptions	Game Example	Ref
Corrupted Sound	Sound Faults	The sound behind an animation is corrupted or warped.	In Final Zone II, a buzzing sound would start after the intro cutscene and continue throughout the game.	W2
Unsynchronized Sound	Sound Faults	The sound behind an animation is not correctly synchronized especially critical for Rhythm games.	In Portable Black Squares and Clazziquai, Edition's music became unsynchronized.	W2
Too Short Sound	Sound Faults	The sound ends before the animation completes	In Whacked, the sound of pendulum swing cut off before the animation ended	PM41
Run Away Sound	Sound Faults	The sound continues even after the animation completes	In The Italian Job, due to lack of sound resources available, the engines sounds looped very badly	PM35
Multiple Action Inconsistency	Action	An in-game action only appears faulty after it is combined with another action.	In The Legend of Zelda: The Wind Waker, there is a glitch where if a player jumps and attacks on the back of the chest in the Ghost Ship, the game will crash	W1
Repeated Action Inconsistency	Action	An in-game action only appears faulty after it is taken repeatedly.	<i>Psychonauts</i> had a quirk where a player became unable to use double jump in a specific level	W2
Game Freeze	Implementation Response Fault	Game stops responding to any user action making progress impossible	In <i>Majora's Mask</i> , the game freezes when the player pushes the open icon button that appears on an underwater chest in Termina Field	W1
Unresponsive Action	Implementation Response Fault	Player performs an in-game action successfully but the consequence or result of that action does not appear.	In <i>Cyberpunk 2077</i> , when Takemura calls, the player takes the call but the NPC does not go away on end call.	V21
Incorrect Reward / Punishment	Implementation Response Fault	An in-game action results in less or more reward compared to expected, conversely less or more punishment than expected is also possible.	In Super Mario Bros. Deluxe for Game Boy Color, if a player earns the Yoshi Medal at the same time as the Red Coins or High Score medals in Challenge mode, the player only receives the Yoshi medal, and the others are lost forever.	W2
Incorrect Response Fault	Implementation Response Fault	Player performs an in-game action successfully but the consequence or result of that action is incorrect.	<i>Final Fantasy V</i> had a weapon (the Chicken Knife), that would sometimes make the player run away from battle instead of doing an attack	W2
Collision Detection Fault	Implementation Response Fault	all instances of faulty collision detection	In PUBG, a faulty location on a specific map allowed player avatars to disappear into the map texture	V2
Crash During Demo	Unexpected Crash	Game crashes unexpectedly during demo or practice sequence animation.	<i>PONCHO</i> has a crash bug in the game demo.	PM36
Crash During Non-Play Moments	Unexpected Crash	Game crashes unexpectedly during game Non-Play Moments.	In Nuclear Throne on windows, game crashes during introduction sequence of Loop Boss, Throne II.	W1
Crash At Start Up	Unexpected Crash	Game crashes unexpectedly during game start up sequence.	Save file corruption results in the game crash at start up fault in Prey.	V2
Crash After Action	Unexpected Crash	Game crashes unexpectedly after an in-game action is executed.	In Mighty No.9, game unexpectedly crashed when player fired a weapon.	V8
Crash At Shut-down	Unexpected Crash	Game crashes unexpectedly during game shut down sequence.	In Super Meat Boy, the game crash during shut down.	PM55
Faulty Pits	Unexpected Paths	Players can navigate over a pit through which they should fall	In Super Mario, Mario did not fall through a pit.	S2
Faulty Obstacles	Unexpected Paths	Players can navigate across obstacles against which they should collide.	In Mass Effect, Matriarch Benezia's use of Biotic powers would toss the main character through the wall	W2

Table B.3: Contains the detailed categorization of selected studies based on testing techniques versus bug category.

	Gaming Balance	Impl. Response	Network	Temporal	Unexpected Crash	Navigational	Non-temporal
Manual scripting	S36	S13, S24, S29, S65, S68	S23, S27, S37, S45, S50, S71	S24		S65	S24, S32, S45, S65
Manual Playtesting		S18		S18			
Supervised Learning	S3, S36, S64, S77	S43					S43
Reinforcement learning	S15, S26, S36, S40	S6, S20, S21, S31, S62, S63, S66	S60		S6, S20, S31, S67, S73	S5, S20, S63, S73	S6, S20, S21, S60, S62, S63, S66, S67, S73
Search-based	S40, S48	S2, S20, S21, S43	S12		S48	S9, S20, S21, S59, S78	S20, S21, S43
Evolutionary Algorithms	S33, S48, S51, S57	S6, S33, S39			S6, S33, S39, S48	S39	S6, S39
Record & Replay		S1, S29					S1, S32
AI	S34, S49, S52, S54, S55, S56	S8, S24, S49, S58, S61, S72, S75	S45, S54	S24	S8, S75	S4, S34	S24, S30, S45, S49, S58, S61, S72, S74
Game grammar / game rules / VGD	S34	S70	S25, S27, S37			S34, S70	S25
Model based		S1, S2, S12, S38	S12, S37			S4	S1, S2, S38
Runtime monitoring		S38, S44, S76		S18, S38, S44, S76			S38, S44, S76
Stress testing			S12, S23, S25, S27, S37, S45, S50, S60, S71				S25, S45, S60
Computer Vision					S11		S22, S32, S41, S46, S69, S70
Others	S53						S28

C Survey Questionnaire Details

Table C.1: Survey Questionnaire

Q#	Survey Question	Type of Answers				
		Single Answer from a list	Multiple Answers from a list	Free text field	Numeric field	Likert Scale
1	Your Name			✓		
2	Name of your organization			✓		
3	Your Job Title			✓		
4	Which game do you play most often?			✓		
5	Which of the following is your area of expertise?		✓			
6	How long have you been playing video games? (in years)				✓	
7	How long have you been developing video games? (in years)				✓	
8	How long have you been testing video games? (in years)				✓	
9	What genre of video games do you play?		✓			
10	How often do you encounter Gaming Balance Faults while playing video games?					✓
11	How would you rate severity of Gaming Balance Faults while playing video games?					✓
12	How would you prioritize the need to fix Gaming Balance Faults encountered while playing video games?					✓
13	What type of Gaming Balance Faults have you encountered while playing video games?		✓			
14	Which of the Gaming Balance sub-categories do you find unnecessary and why?		✓	✓		
15	How often do you encounter Implementation Response Faults while playing video games?					✓
16	How would you rate severity of Implementation Response Faults while playing video games?					✓
17	How would you prioritize the need to fix Implementation Response Faults encountered while playing video games?					✓
18	What type of Implementation Response Faults have you encountered while playing video games?		✓			
19	Which of the Implementation Response Faults sub-categories do you find unnecessary and why?		✓	✓		
20	How often do you encounter Network related Faults while playing video games?					✓
21	How would you rate severity of Network Faults while playing video games?					✓
22	How would you prioritize the need to fix Network Faults encountered while playing video games?					✓
23	What type of Network Faults have you encountered while playing video games?		✓			
24	Which of the Network related sub-categories do you find unnecessary and why?		✓	✓		
25	How often do you encounter Sound Faults while playing video games?					✓
26	How would you rate severity of Sound Faults while playing video games?					✓
27	How would you prioritize the need to fix Sound Faults encountered while playing video games?					✓
28	What type of Sound Faults have you encountered while playing video games?		✓			
29	Which of the Sound Faults sub-categories do you find unnecessary and why?		✓	✓		
30	How often do you encounter Temporal Faults while playing video games?					✓
31	How would you rate severity of Temporal Faults while playing video games?					✓
32	How would you prioritize the need to fix Temporal Faults encountered while playing video games?					✓
33	What type of Temporal Faults have you encountered while playing video games?		✓			

Table C.1 continued from previous page

Q#	Survey Question	Type of Answers				
		Single Answer from a list	Multiple Answers from a list	Free text field	Numeric field	Likert Scale
34	Which of the Temporal Faults sub-categories do you find unnecessary and why?		✓	✓		
35	How often do you encounter Unexpected Crash Faults while playing video games?					✓
36	How would you rate severity of Unexpected Crash Faults while playing video games?					✓
37	How would you prioritize the need to fix Unexpected Crash Faults encountered while playing video games?					✓
38	What type of Unexpected Crash Faults have you encountered while playing video games?		✓			
39	Which of the Unexpected Crash sub-categories do you find unnecessary and why?		✓	✓		
40	How often do you encounter Navigational Faults while playing video games?					✓
41	How would you rate severity of Navigational Faults while playing video games?					✓
42	How would you prioritize the need to fix Navigational Faults encountered while playing video games?					✓
43	What type of Navigational Faults have you encountered while playing video games?		✓			
44	Which of the Navigational sub-categories do you find unnecessary and why?		✓	✓		
45	How often do you encounter Non-Temporal Faults while playing video games?					✓
46	How would you rate severity of Non-Temporal Faults while playing video games?					✓
47	How would you prioritize the need to fix Non-Temporal Faults encountered while playing video games?					✓
48	What type of Non-Temporal Faults have you encountered while playing video games?		✓			
49	Which of the Non-Temporal Faults sub-categories do you find unnecessary and why?		✓	✓		
50	How often do you encounter Action Faults while playing video games?					✓
51	How would you rate severity of Action Faults while playing video games?					✓
52	How would you prioritize the need to fix Action Faults encountered while playing video games?					✓
53	What type of Action Faults have you encountered while playing video games?		✓			
54	Which of the Action Faults sub-categories do you find unnecessary and why?		✓	✓		
55	How often do you encounter Invalid Graphical Representation Faults while playing video games?					✓
56	How would you rate severity of Invalid Graphical Representation Faults while playing video games?					✓
57	How would you prioritize the need to fix Invalid Graphical Representation Faults encountered while playing video games?					✓
58	What type of Invalid Graphical Representation Faults have you encountered while playing video games?		✓			
59	Which of the Invalid Graphical Representation sub-categories do you find unnecessary and why?		✓	✓		
60	What major faults/errors/failures have you encountered during gameplay in video games other than the ones already classified?			✓		

D Survey Response Visualization

This appendix present visual depiction of raw data from the survey responses used for derivation of conclusions in our paper.

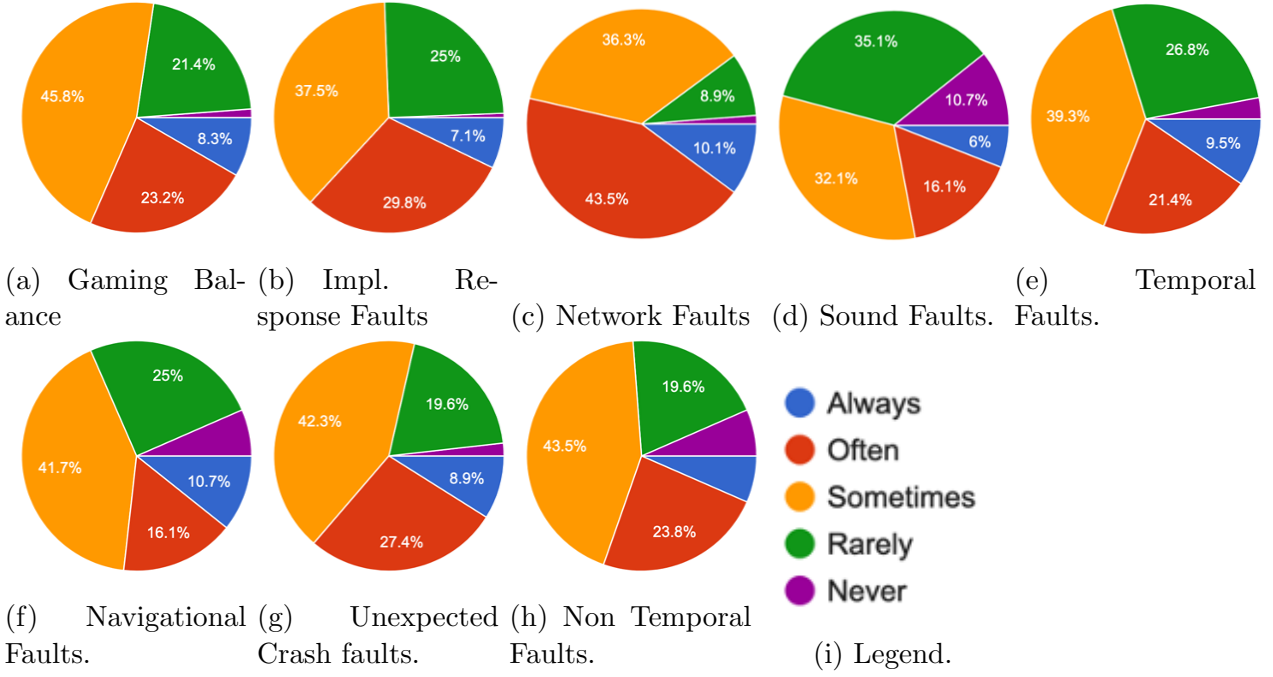


Figure D.1: Frequency of occurrence of game bugs according to survey responses.

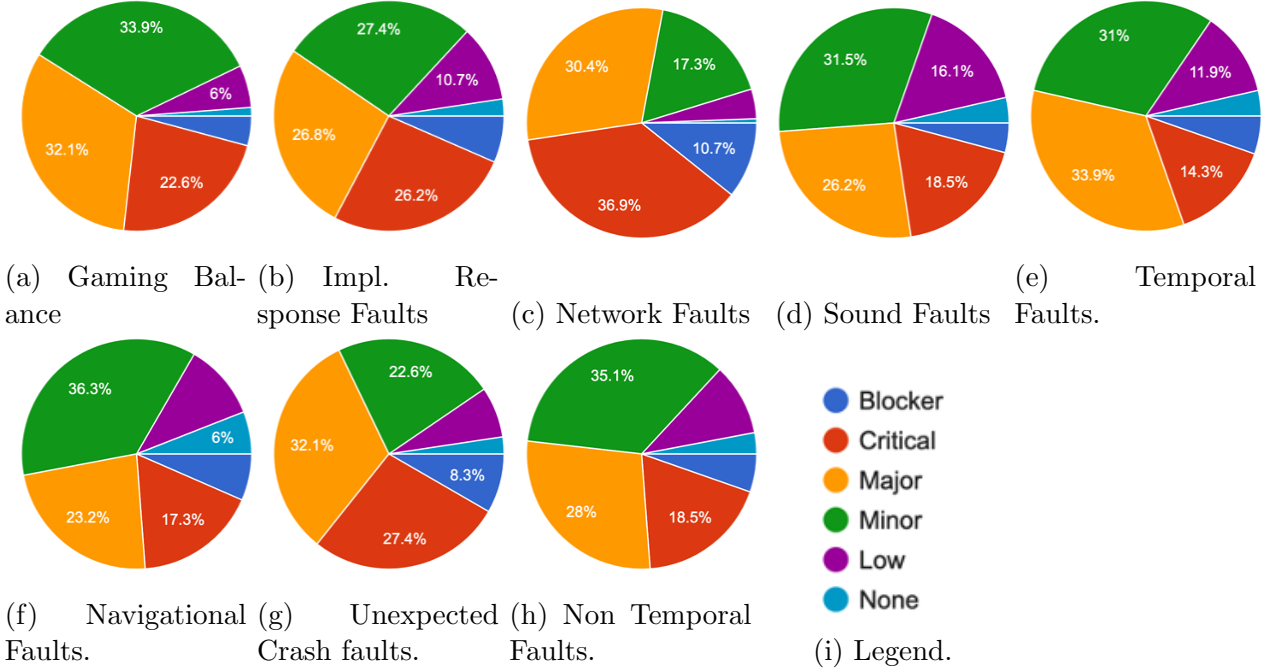


Figure D.2: Severity of game bugs according to survey responses.

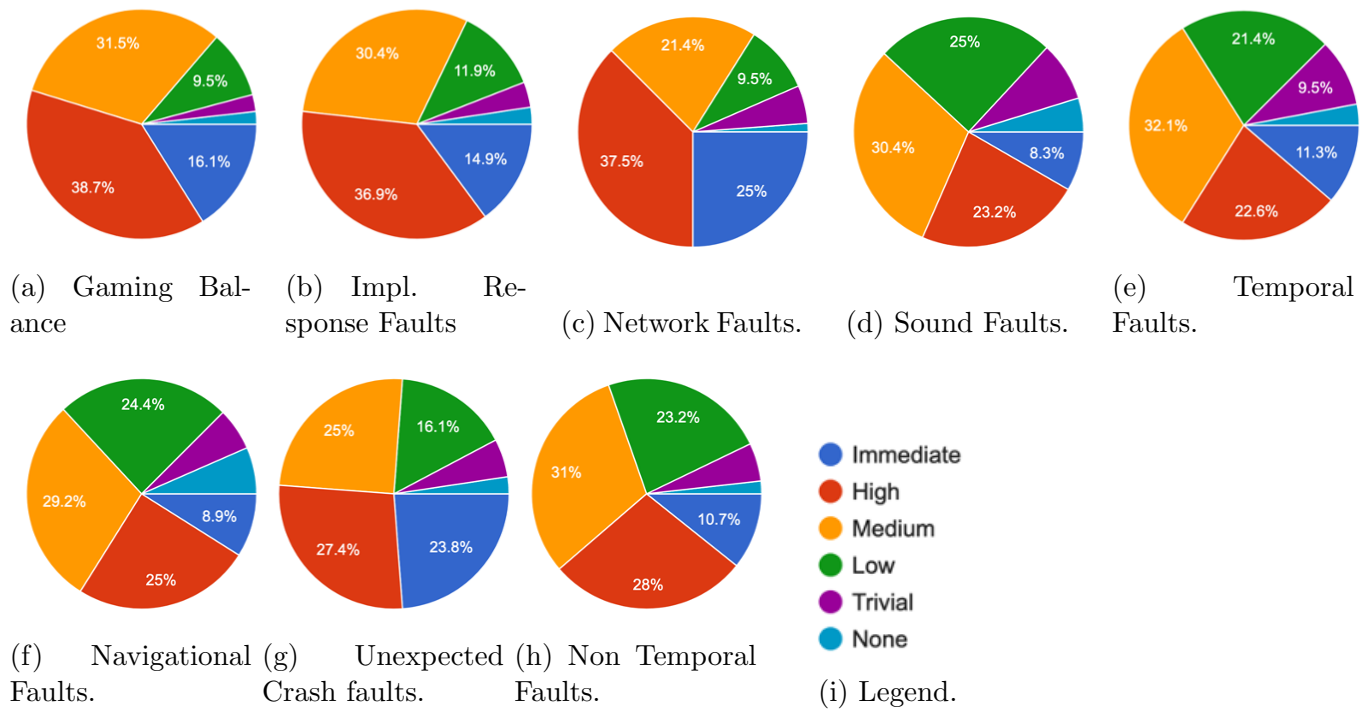


Figure D.3: Bug fixing Priority of game bugs according to survey responses.

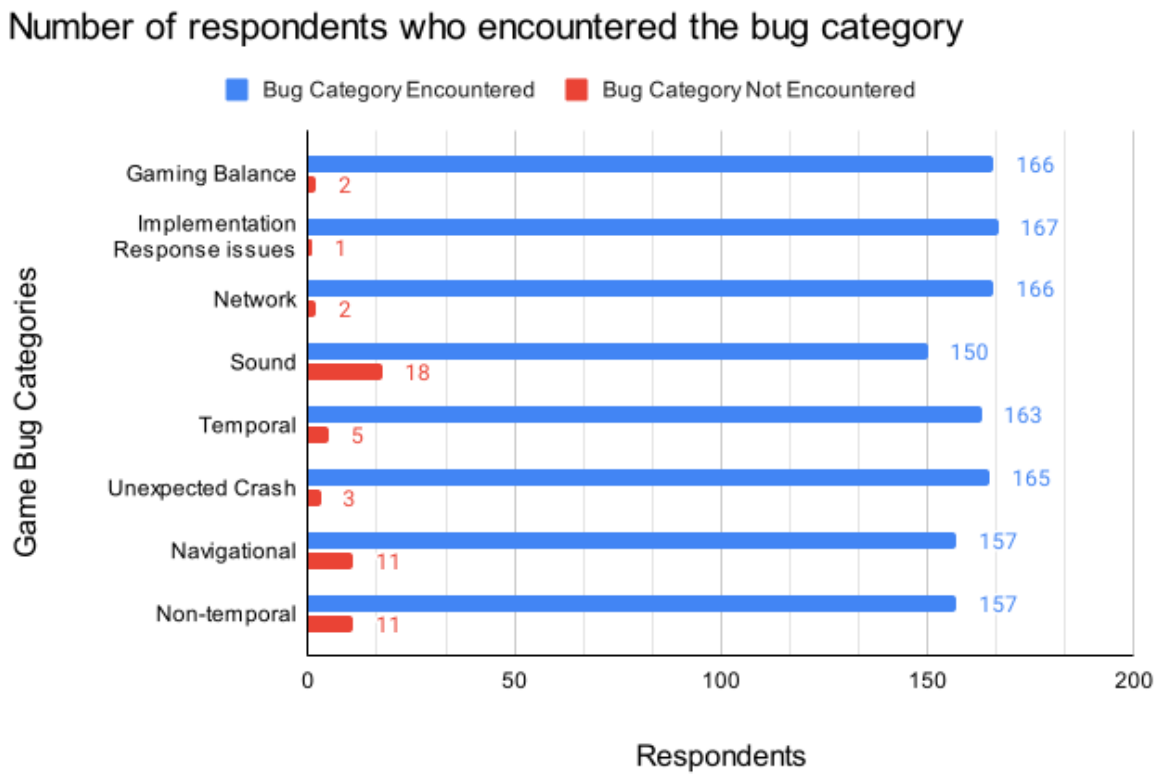


Figure D.5: Respondent Count vs. Game Bug category.

Frequency of occurrence of bug categories

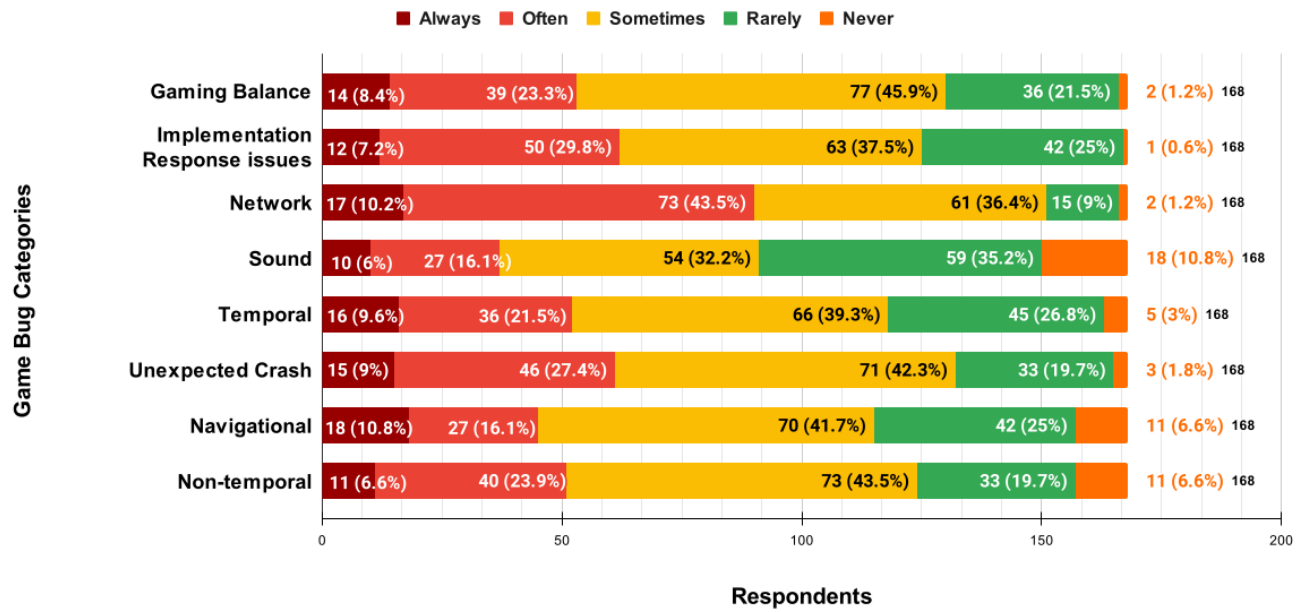


Figure D.4: Frequency of occurrence of bug categories according to survey respondents.

Severity of bug categories

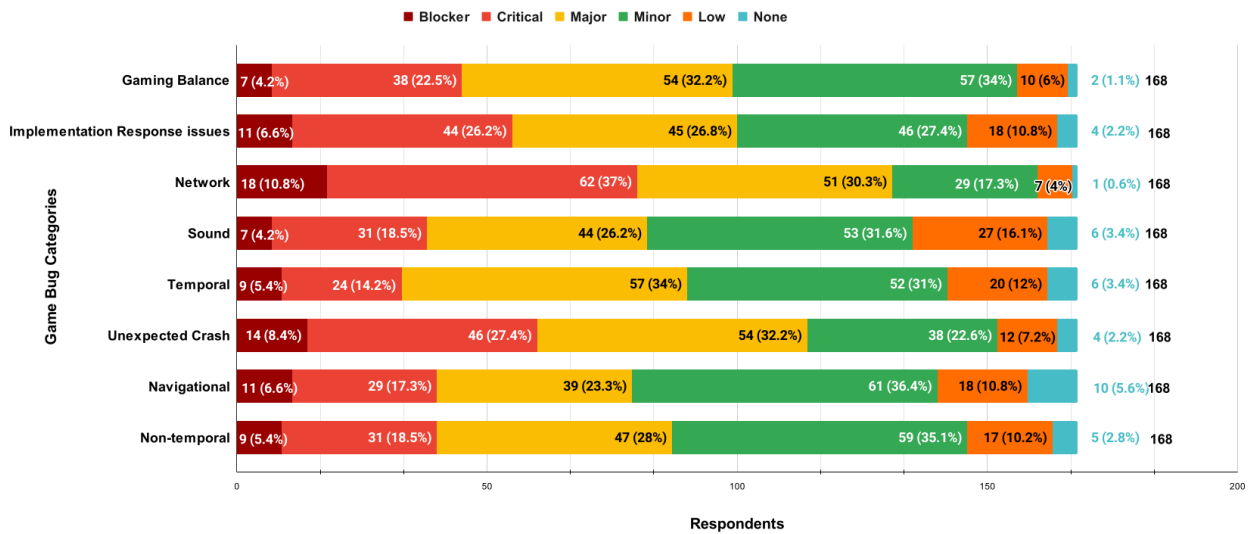


Figure D.6: Severity of bug categories according to survey respondents.

Priority of bug categories

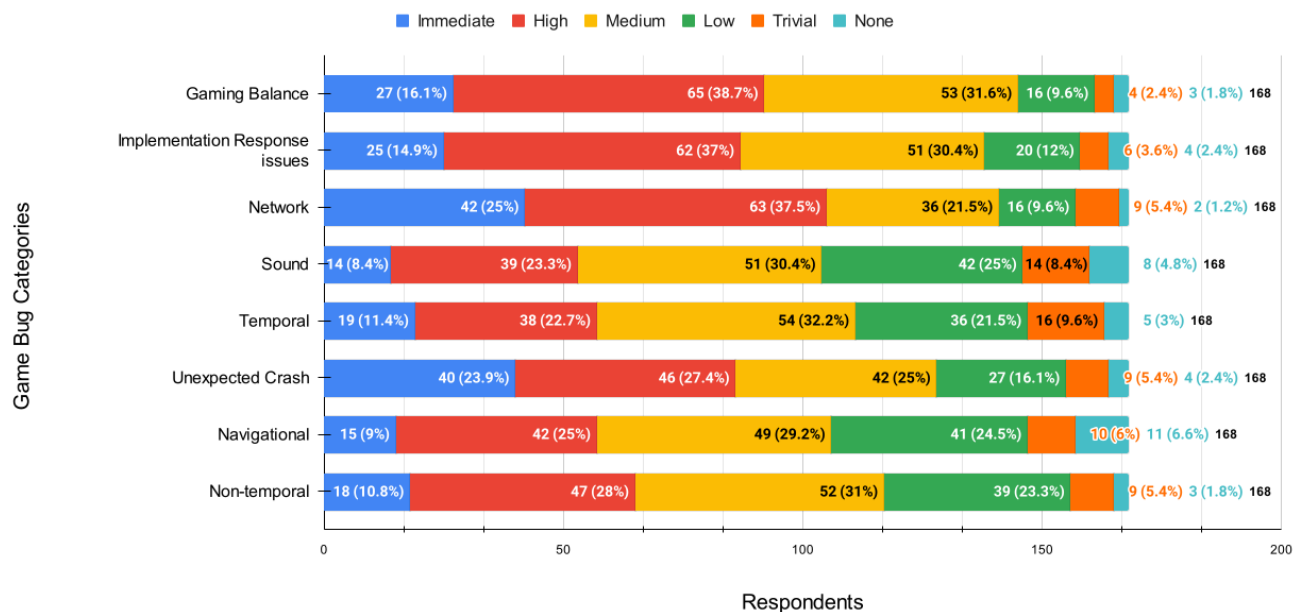


Figure D.7: Priority to fix of bug categories according to survey respondents.

Respondent Count vs. Gameplay Experience (in years)

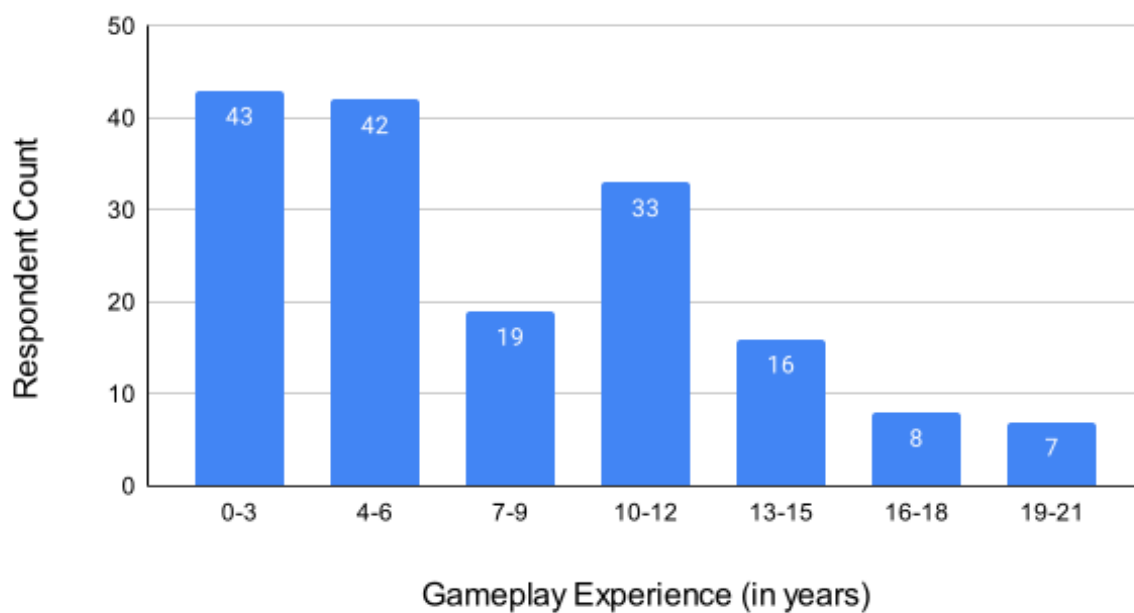


Figure D.8: Respondent Count vs. Gameplay Experience (in years).

Respondent Count vs. Game Testing Experience (in years)

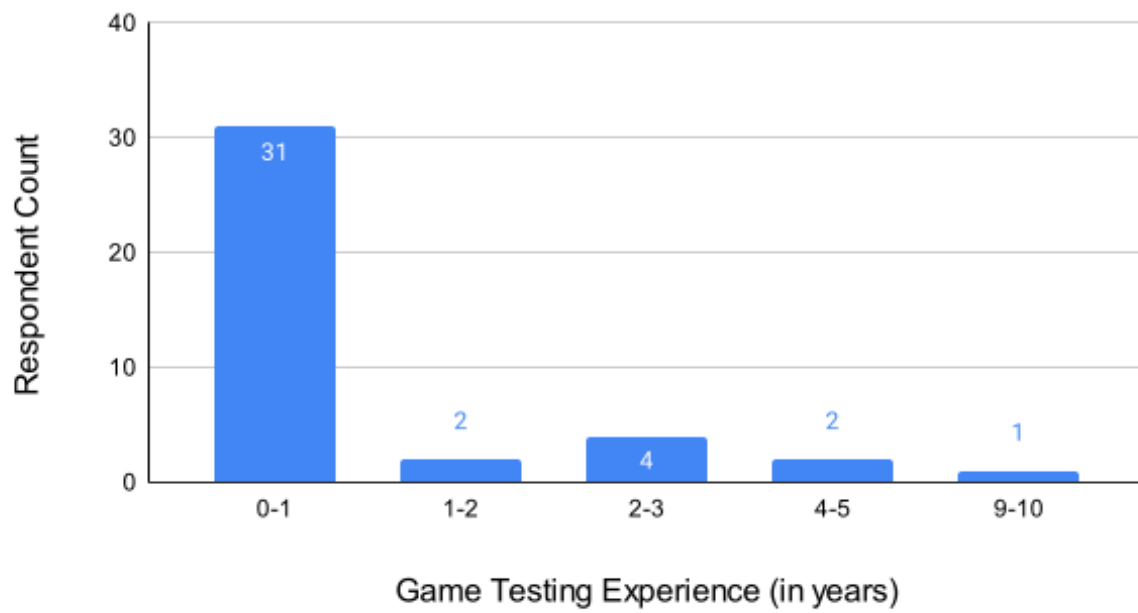


Figure D.9: Respondent Count vs. Game Testing Experience (in years).

Respondent Count vs. Game Development Experience (in years)

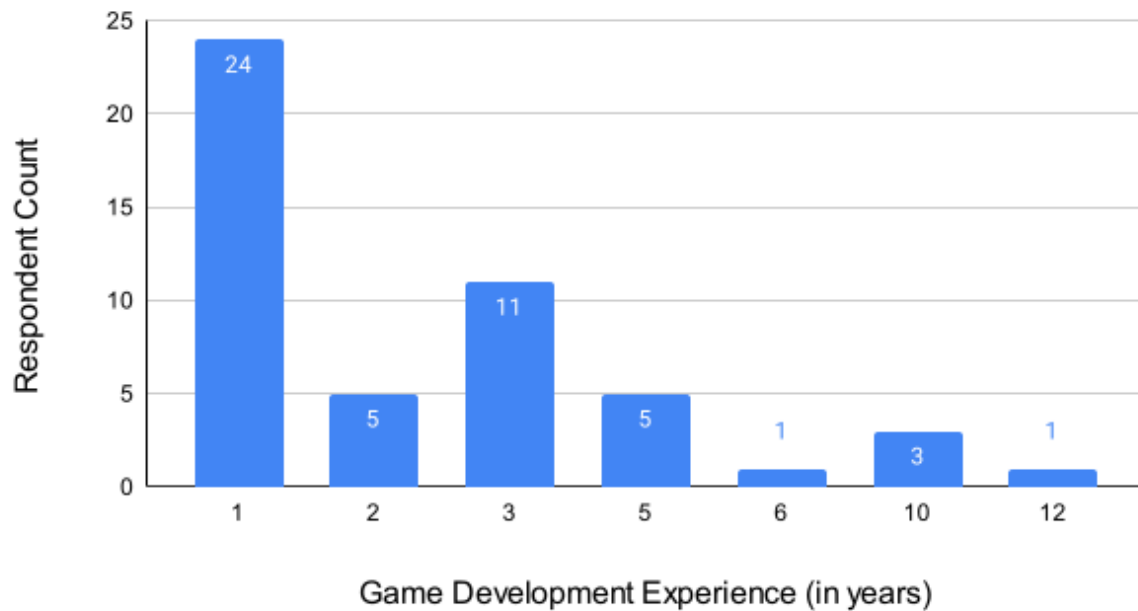


Figure D.10: Respondent Count vs. Game Development Experience (in years).

Most popular genres of video games played by survey respondents

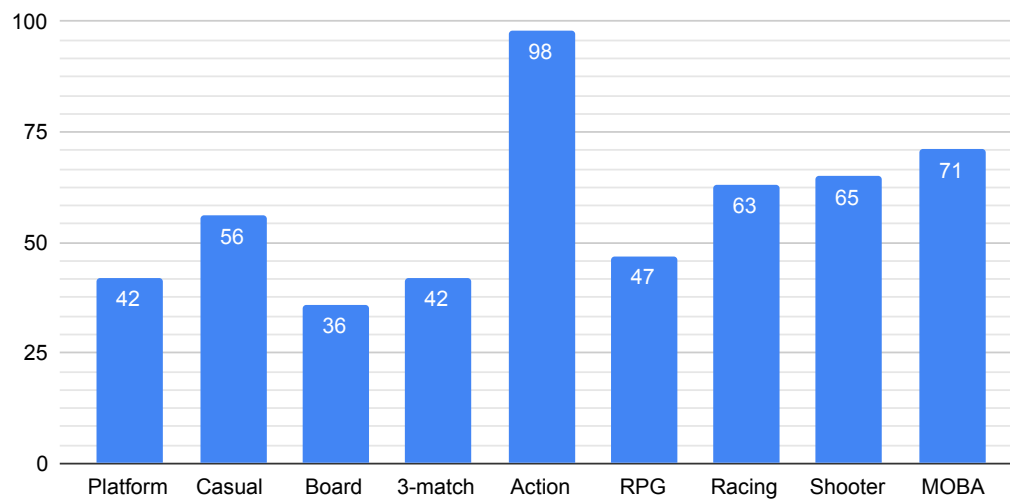


Figure D.11: Most popular genres of video games played by survey respondents.

Games most often played by survey respondents

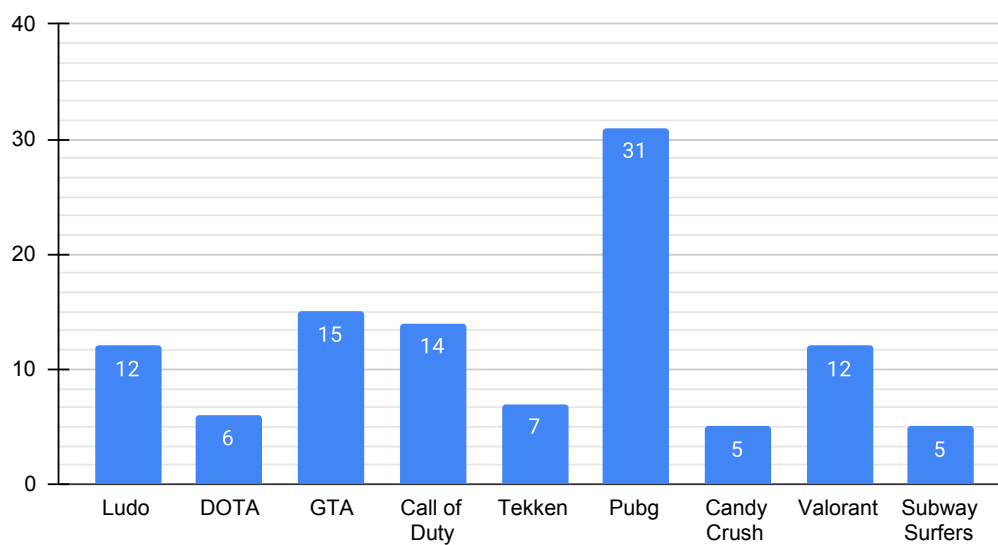
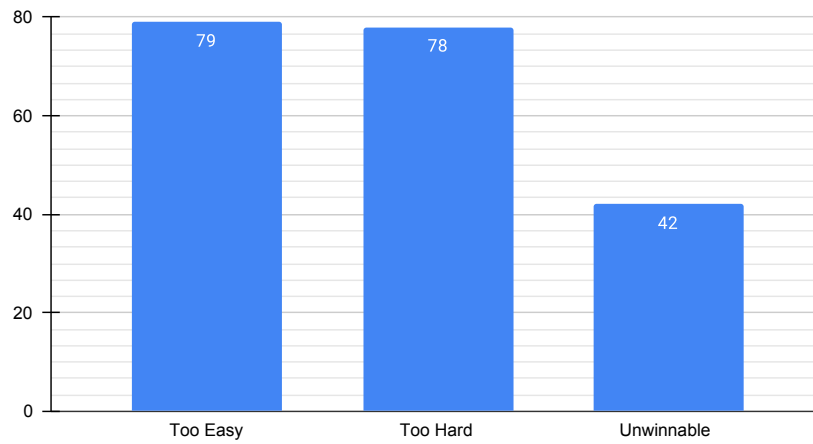


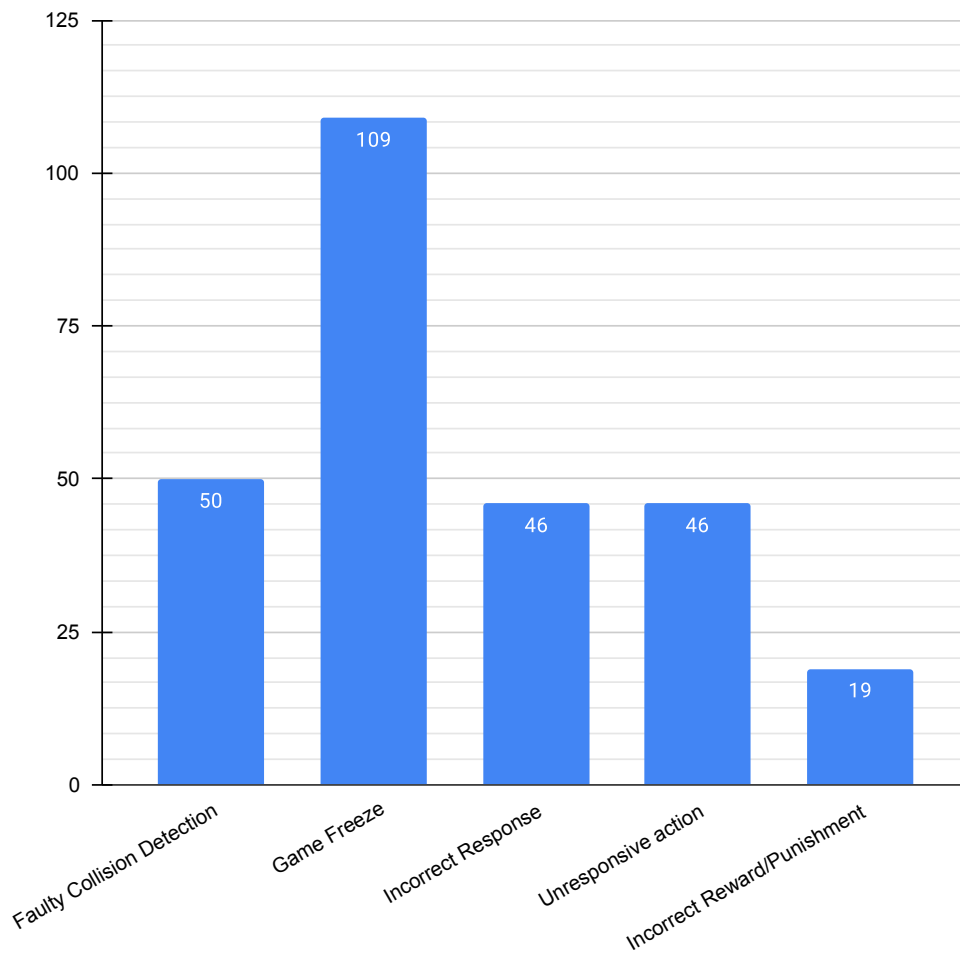
Figure D.12: Games most often played by survey respondents.

Types of Gaming Balance Issues encountered



(a) Gaming Balance

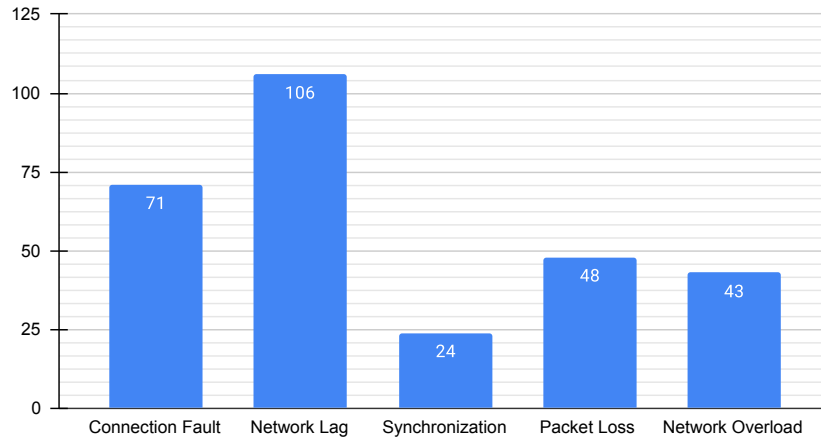
Types of Implementation Response Issues encountered



(b) Implementation Response Faults

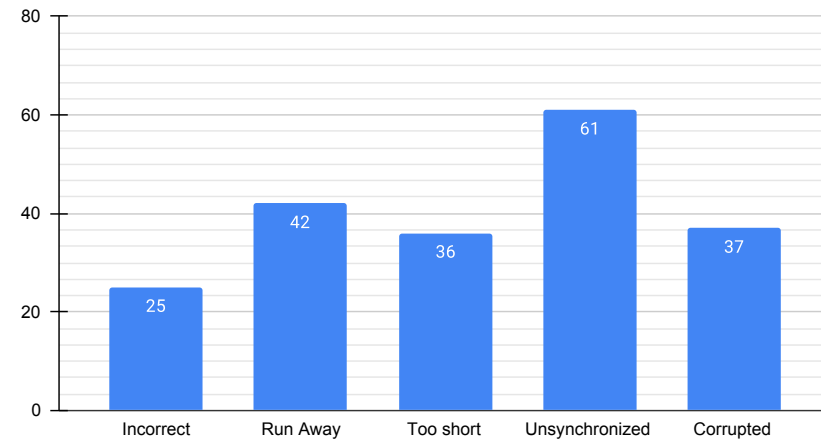
Figure D.13: Types of different bug taxonomy categories encountered according to survey respondents.

Types of Network Issues encountered



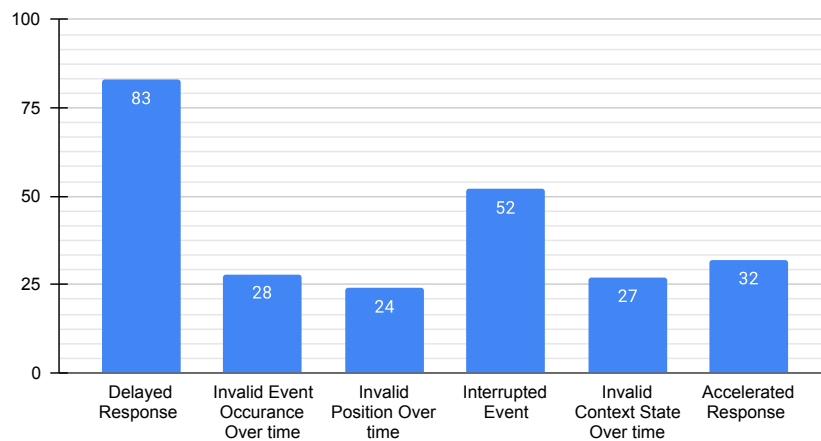
(c) Network Faults

Types of sound Issues encountered



(d) Sound Faults

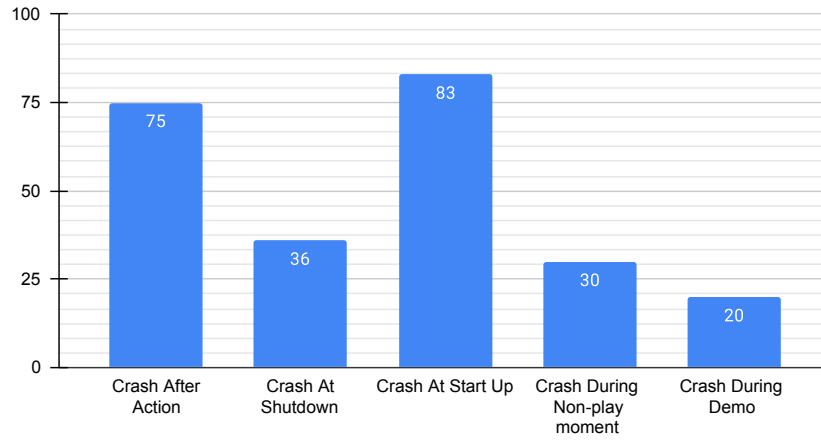
Types of Temporal Issues encountered



(e) Temporal Faults

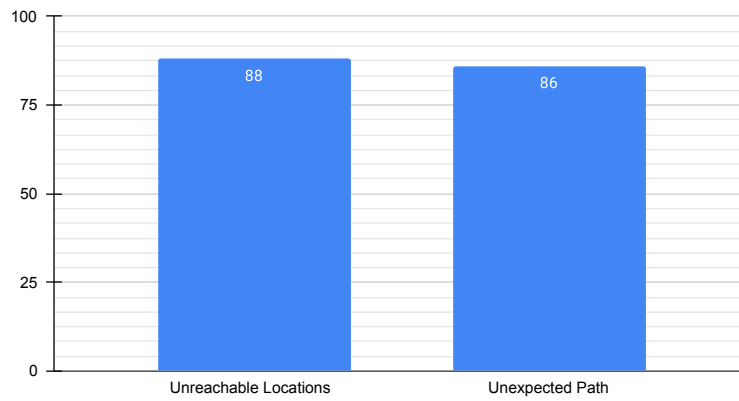
Figure D.13: Types of different bug taxonomy categories encountered according to survey respondents (continued from previous page.).

Types of Crash Issues encountered



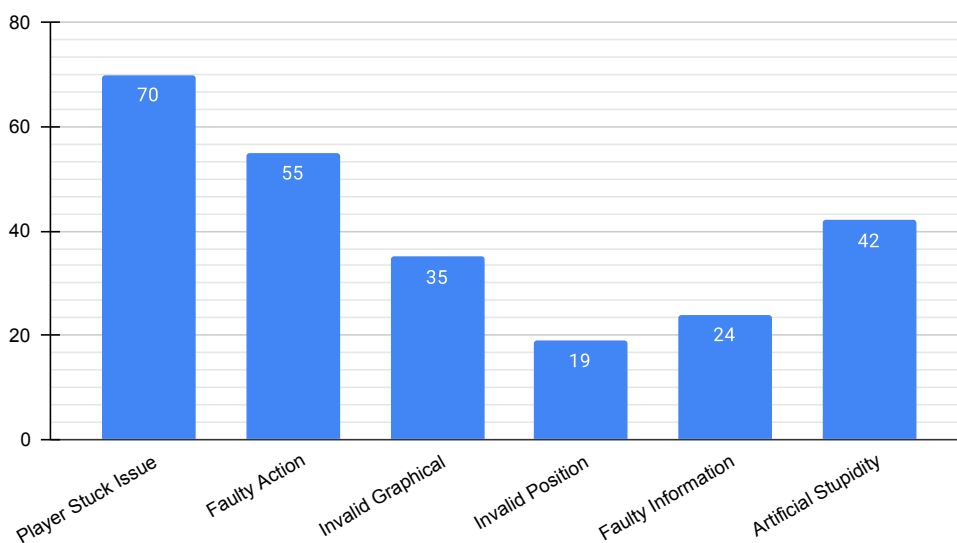
(f) Unexpected Crash

Types of Navigation Issues encountered



(g) Navigational Faults

Types of Non-temporal Issues encountered



(h) Non-Temporal Faults

Figure D.13: Types of different bug taxonomy categories encountered according to survey respondents (continued from previous page).