The Journal of Systems & Software 204 (2023) 111789

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A catalogue of game-specific anti-patterns based on GitHub and Game Development Stack Exchange☆

Vartika Agrahari, Shriram Shanbhag *, Sridhar Chimalakonda, A. Eashaan Rao

Research in Intelligent Software & Human Analytics (RISHA) Lab, Department of Computer Science & Engineering, Indian Institute of Technology Tirupati, India

ARTICLE INFO

Article history:
Received 3 August 2022
Received in revised form 17 April 2023
Accepted 15 June 2023
Available online 22 June 2023

Keywords:
Games
Anti-patterns
Catalogue
Thematic analysis

ABSTRACT

With the ever-increasing use of games, game developers are expected to write efficient code and support several aspects such as security, maintainability, and performance. However, the need for frequent updates in game development may lead to quick-fix solutions and bad coding practices. Though unintentional, these bad practices may lead to poor program comprehension and can cause several issues during software maintenance. The quick-fix solutions might lead to technical debts due to the presence of anti-patterns and code smells, which may affect the functional and non-functional requirements of the game. To avoid such instances, game developers may need some guidelines to refer to during the game development process. Thus, to aid developers and researchers, in our previous work, we had presented an initial catalogue of anti-patterns in the domain of game development. To broaden the scope of the catalogue and diversify the instances of anti-patterns, we analyzed additional data from a Q&A platform. We present 15 game-specific anti-patterns based on thematic analysis of 189 *issues*, 892 *commits*, 104 *pull requests* from 100 open-source GitHub game repositories, and 971 *questions* from Game Development Stack Exchange. We see the catalogue as an effort towards improving the development and quality of the games. The catalogue containing a detailed description of every anti-pattern with the *context*, *problem*, *solution*, *example(s)*, and their occurrences on GitHub and Game Development Stack Exchange is available at https://rishalab.github.io/Catalog-of-Game-Antipatterns/.

1. Introduction

The game industry continues to see extending growth in terms of revenue and worldwide usage. For the year 2022, Newzoo forecasts that in terms of revenue, the game industry has crossed $200 billion in the mobile, PC, and console segment.[1] With increasing market demand for games and their widespread usage has posed a challenge for game developers to develop and design better quality games in less time (Kanode and Haddad, 2009; Edholm et al., 2017; McKenzie et al., 2021). This pressing need may sometimes force developers to use quick-fix solutions and can result in the violation of functional and non-functional requirements (Brown et al., 1998; Fowler, 1997).

Developers sometimes deviate from good design choices and coding practices, leading to the adoption of bad practices or anti-patterns in their software (Brown et al., 1998). Even though these choices and practices may not have an immediate impact on the functionality or correctness of the program, they can result in long-term issues related to maintainability, security, performance, and increased technical debt (Kermansaravi et al., 2021; Borowa et al., 2021). Existing literature consists of several research studies regarding anti-patterns and code smells (Palomba et al., 2014; Hübener et al., 2022; Lin et al., 2021; Guamán et al., 2022). Brown et al. (1998) explain anti-patterns by stating that they are like misfits to a problem that should be prevented and avoided to lead to a better solution. Despite the existence of several studies focusing on anti-patterns and code smells (Silva et al., 2016; Palomba et al., 2014; Fowler, 1997), understanding and analyzing the presence of these anti-patterns and smells in games is still largely unexplored in the literature (Kanode and Haddad, 2009; Edholm et al., 2017; Politowski et al., 2021).

Games need to be considered a distinct domain over other software as it involves components such as AI simulations, camera movements, players' actions, game mechanics, and particle effects (Murphy-Hill et al., 2014). They deal with real-time constraints and a continuous rendering process. In addition, due

☆ Editor: Raffaela Mirandola.

* Correspondence to: Indian Institute of Technology Tirupati, Yerpedu – Venkatagiri Road, Yerpedu Post, Tirupati District, Andhra Pradesh, 517619, India.

E-mail addresses: cs18m016@iittp.ac.in (V. Agrahari), cs20s503@iittp.ac.in (S. Shanbhag), ch@iittp.ac.in (S. Chimalakonda), cs21d002@iittp.ac.in (A.E. Rao).

[1] https://newzoo.com/insights/articles/games-market-revenues-will-pass-200-billion-for-the-first-time-in-2022-as-the-u-s-overtakes-china

https://doi.org/10.1016/j.jss.2023.111789
0164-1212/

to high popularity, developers tend to introduce new features frequently to engage the players, which might lead to unstable requirement instability, insufficient test automation, and time pressure (Borowa et al., 2021). These may enable quicker releases, but they may also bring several quality issues with them. Studies related to games have been seen from multiple perspectives in the literature. For instance, games are look through the lens of psychological and social aspects (Lee and Peng, 2006), educational (Giessen, 2015) and their behavioral (Anderson and Bushman, 2001) effects on users. Nevertheless, there is less research on the software quality and bad practices in game development (Nystrom, 2014; Björk and Holopainen, 2006; Borrelli et al., 2020). Thus, there is a need to discover bad practices in game development, which involves aspects that are different from traditional software development practices.

Researchers have created catalogues within software engineering for domains such as android (Carvalho et al., 2019), energy patterns (Cruz and Abreu, 2019; Shanbhag et al., 2022), architecture (Garcia et al., 2009), software metrics (Dalla Palma et al., 2020), anti-patterns in multi-language systems (Abidi et al., 2019), emphasizing the need for domain-specific catalogues. Following the tradition, this study aims to extend the domain-specific catalogue by creating one for game anti-patterns. We selected two sources of information, i.e., GitHub and Game Development StackExchange[2] to study the instance for game anti-patterns.

GitHub hosts a significant number of open-source game repositories (Kalliamvakou et al., 2014), with approximately 188 K repositories related to games. These repositories contain valuable software artifacts such as *commits*, *issues*, and *pull requests*, which can be leveraged to gain insights into the challenges faced by users and developers, and their potential solutions. For instance, Cruz and Abreu (2019) have analyzed these artifacts to propose a catalogue of energy patterns in Android and iOS projects. While feedback and surveys do provide valuable insights from developers, these artifacts provide better and more comprehensive information about the difficulties developers face and how they affect the usability and performance of software (Kalliamvakou et al., 2014). Thus, taking cues from earlier studies we considered the text corpus of *commits*, *issues*, and *pull requests* to find the existence of game-specific anti-patterns in the game repositories.

The second source we considered for our study is *"Game Development Stack Exchange"*, a question-and-answer site for independent and professional game developers. The aim of this platform is to provide support in game development activity, with experts answering any questions related to issues faced by game developers. We selected this platform considering that it has over 134 K posts related to game development and is a part of the Stack Exchange[3] network. The data from this platform is also used in other game development-related studies (Santos et al., 2018; Kamienski and Bezemer, 2021). Therefore, we believe in finding some instances of game anti-patterns through this discussion platform.

This paper proposes a catalogue of game-specific anti-patterns that could serve as a checklist for game developers during the development process. Cataloguing could help in prioritizing the development process and resources. Language-specific anti-patterns and detection techniques may not be sufficient to handle bad practices in games (Fard and Mesbah, 2013). Unlike the existing literature which primarily focuses on source code and other artifacts (Garcia et al., 2009; Abidi et al., 2019), we wish to leverage the potential of the text corpus of GitHub's *commits*, *issues*, *pull requests* and Game Development Stack Exchange's *Q&A* to answer the research question:

What are the most prevalent anti-patterns in the context of games?

We applied a systematic methodology comprising of data collection followed by the commonly used thematic analysis (Fereday and Muir-Cochrane, 2006; Petersen et al., 2008) on 2156 text data records of consisting of *issues*, *commits*, *pull requests* from GitHub and *questions* from Game Development Stack Exchange to propose a catalogue of fifteen game-specific anti-patterns. We describe the methodology in detail in Section 2 and document the catalogue on our website as a reference for the developers.

The primary contributions of our study are:

- A catalogue of fifteen game-specific anti-patterns along with a detailed discussion of each anti-pattern. The detailed description is available online at : https://rishalab.github.io/Catalog-of-Game-Antipatterns/.
- A public-domain dataset consisting of total 2156 records, with 892 *commits*, 189 *issues*, 104 *pull requests* mined from 100 open-source games and 971 *questions* with accepted answer from Game Development Stack Exchange.

It is noteworthy that this paper extends our previous conference version (Agrahari and Chimalakonda, 2022). In particular, we improved the previous version as follows:

- We performed a thematic analysis of 971 *questions* with accepted answer from Game Development Stack Exchange.
- We discovered five additional game-specific anti-patterns from our thematic analysis.
- We found 397 additional occurrences in Game Development Stack Exchange data for four anti-patterns from the catalogue in our previous version.

The remainder of the paper is structured as follows. Section 2 focuses on the methodology used to obtain the catalogue. Detailed catalogue of game-specific anti-patterns is presented in Section 3 with threats to validity in Section 4. Discussion in Section 5 is followed by related work in Section 6, and eventually the paper ends with conclusion and future work in Section 7.

2. Methodology

We applied a two phase process consisting of data collection followed by thematic analysis to arrive at the catalogue. Thematic Analysis is a commonly used methodology used by researchers to analyze patterns by utilizing qualitative analysis (Fereday and Muir-Cochrane, 2006; Petersen et al., 2008). Researchers have used this qualitative method for categorizing energy patterns (Cruz and Abreu, 2019), agile challenges (Gregory et al., 2015), awareness interpretation for collaborative computer games (Teruel et al., 2016), and so on. Researchers also performed thematic analysis to gain insights based on user reviews for disaster apps (Tan et al., 2020). Thus, leveraging the thematic analysis process fits our context of catalogue creation. We curated a dataset of *commits*, *issues*, and *pull requests* from 100 open-source GitHub game repositories of different genres, such as board game, puzzle, arcade, and so on. We also collected a dataset of *questions* from Game Development Stack Exchange, a popular Q&A platform for professional game developers. The methodology, shown in Fig. 1, consisted of two phases:

- Collection of the dataset in the form of *commits*, *issues*, *pull requests*, from GitHub and *questions* from Game Development Stack Exchange.
- Execution of thematic analysis on the collected data to obtain game-specific anti-patterns.

[2] https://gamedev.stackexchange.com/
[3] https://stackexchange.com/

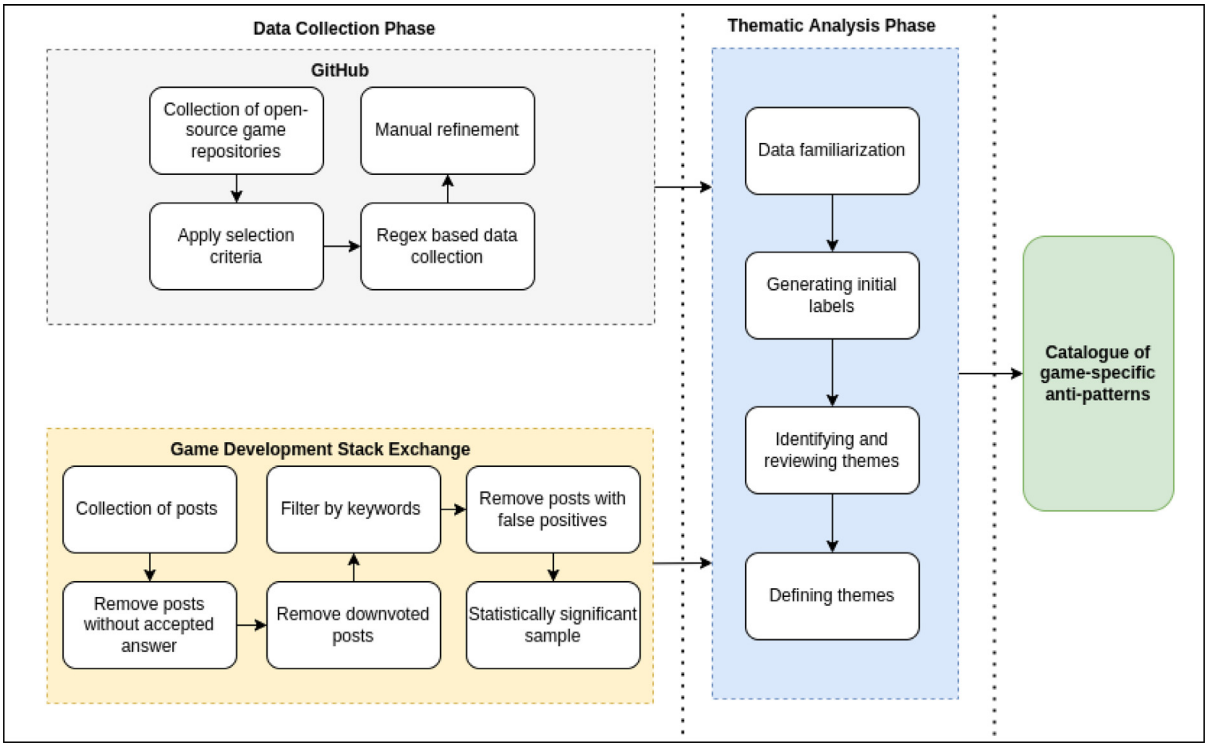

Fig. 1. Methodology used to obtain catalogue of game-specific anti-patterns.

2.1. GitHub data collection

We followed a four-step process to collect the dataset of text records for thematic analysis in the process of generating the catalogue for game-specific anti-patterns.

2.1.1. Collection of open-source game repositories

We started collecting the game repositories from a popular GitHub repository[4] that provides awesome list of desktop games. The GitHub repository[4] listing the popular games itself has the stargazers count of approximately 18.2 K. We gathered the repositories and browsed each game to inspect if it is game by checking its README as there were non-game repositories and forked repositories of the original source. Thus, we ended up with a list of URLs of 229 game repositories. There were a total of 15 genres of games spread across 19 programming languages. The three major languages in which these games were developed are *JavaScript*, *C++*, and *C* with 92, 45, and 23 games respectively. The games present in this list were either *browser-based* (can be played instantaneously in the browser with no need for installation) or *native*, which requires installation.

2.1.2. Apply selection criteria

Although we collected links of 229 games, we observed that there were games with zero star count, zero forks, and with size of less than 1 MB. Therefore, for thematic analysis we decided to resort to a subset of these games repositories. We proposed the selection criteria below and filtered 100 games out of 229 game repositories. We took the subset of the total games, which is significant enough and can represent the whole dataset as elaborated in below description. We created the subset by combining random games and the games having more stargazers count. Thus, mathematically subset can be shown:

Selection Rule
If $\boldsymbol{A}$ = *Whole Dataset*,
then $\mathbf{B} \subset \mathbf{A}$, where,
$\boldsymbol{B}$ = *0.5*(top starred games(**A**))+*
0.5(random sample of remaining dataset after selecting top starred games(**A**))*

- **First Half :** Half of the subset determined for thematic analysis contains the top games sorted according to the stargazers count in descending order. We did this to make sure that we include the games which are most popular among the developer community (Borges and Valente, 2018; Ray et al., 2014). Also, the games having more stargazers count are generally open-source games which are not developed by a single developer, but a team of them (Klug and Bagrow, 2016). Thus, we consider and analyze these games for our catalogue. We selected a total of 50 top starred games from the 229 game repositories. The metadata of the half subset is mentioned below:
  - Average stargazers count: 2320.9
  - Average forks count: 816.46
  - 22 games out of 50 are developed in the *C++* language.
- **Second Half :** Another half of the subset belongs to the random sample taken from the remaining links after the selection of top starred games. We did so to ensure that we also take the randomized sample of the whole dataset. To confirm that the random sample is statistically significant, we resorted to measuring the confidence level and confidence interval of the selected subset. The confidence interval represents the range of data that incorporates the true value of the unknown population parameter. In other words, if we construct an infinite number of the independent sample using a confidence interval, then the confidence level will

[4] https://github.com/leereilly/games

Table 1
Metadata of GitHub's 100 selected games.

Features	Statistics
Average stargazers count	1214
Average forks count	431
Open issues	92
#genres	13
#programming languages	15
# browser-based games	56
# native games	44

correspond to the proportion that contains the true value of the parameter. Therefore, based on stargazers count as a metric, the confidence level of the selected subset is 95% with a confidence interval in the range of 70-135 (Kitchenham et al., 2002). A confidence interval of 95% states that if we consider random samples of the same sample size for 100 times, then 95 out of 100 contains the true but unknown mean in the interval of 70–135. A total of 50 games were selected randomly from the set of 179 games (deducting top starred 50 games from 229 games). The metadata of random games selected is mentioned below.

- Average stargazers count: 102.5
- Average forks count: 43.2
- 35 games out of 50 were developed in *JavaScript* language.

We observe that selected games in the subset majorly written in *C++* and *JavaScript*. Further, among the 100 games selected, 44 games are *native* games while the remaining 56 games are *browser-based*. The metadata of the game chosen for the GitHub repositories are shown in Table 1.

2.1.3. Regular expression based gathering of data

Based on the 100 games selected for analysis, we made a regular expression based search in *commits*, *issues*, and *pull requests* to find out the subjects of our potential interests (Cruz and Abreu, 2019; Bao et al., 2016). We searched for various words that can possibly correspond to game-specific anti-patterns. The search words along with the rationale for selecting those words are given in Table 2. The regular expression used for the search was:

```
.*(performance|efficiency|delay|lag|
usability|refactor|code smell|anti-
pattern|bad|issue|bug|defect|flaw|
fault|problem|energy|battery|power|
freeze|crash|hang|glitch|control).*
```

We used GitHub API v3.[5] and PyGithub[6] to mine *commits*, *pull requests*, and *issues* For *pull requests* and *issues*, GitHub API v3 documentation[7] states that -*"GitHub's REST API v3 considers every pull request an issue, but not every issue is a pull request"*. For this reason, we identified *issues* out of *get_issues (state='all')* function, which returns *issues* as well as *pull requests*. Another function *get_pulls (state='all')* returns a list of all *pull requests*. We utilized the results of both functions to categorize and separate the *issues* and *pull requests* from the text data. We combined the title, body, and comments of all the *issues* and *pull requests* to perform regular expression matching. For commits, we included text records that

Table 2
Rationale behind words chosen for Data Collection.

Terms	Rationale
Performance, Efficiency, Delay, Lag	To trace the records having issues related to performance of games.
Usability	To seek the problem related to usability.
Refactor, Code Smell, Anti-Pattern	Problem related to bad practices in code and their refactoring.
Bad, Issue, Bug, Defect, Flaw, Fault, Problem	To track the problems which causes some issue during the gameplay. We made it general to track any kind of issue in games.
Energy, Battery, Power	Problems related to energy efficiency of game.
Freeze, Crash, Hang, Glitch	To track the issues related to the user interface of game.
Control	Issues related to the control system of games.

were merged into the default branch of the GitHub repository. As a result, we extracted a total of 1989 records from 100 games, consisting of 1245 *commits*, 523 issues, and 221 *pull requests*. It can be seen that the count of *commits* is higher compared to *issues* or *pull requests*. This could be attributed to the fact that the number of *commits* is generally higher than *pull requests* and *issues* in most repositories. Some of the repositories have no *issues* or *pull requests*; still, they have a lot of *commits*. For example, Game-off-2013,[8] is popular among the developer community with 590 forks so far and contains 368 *commits* but no *issues* or *pull requests*.

2.1.4. Manual refinement

To validate that the subjects we mined are relevant and fit our interest, we manually inspected the data collected to separate the false positives (Cruz and Abreu, 2019). An example false positive we encountered in one of the *issues* is: *"We had a rocking day for gameplay balancing. I think it's pretty good for now, let's create separate issues for any remaining tweaks"*.[9] We observe that the text data does not specify any anti-patterns in games.

We followed two strategies for manual refinement:
1. Examine the matched text data with the regular expression and find its relevance in the context of games.
2. Annalyze the entire thread of the matched record by visiting its GitHub page and examining the comments and the meaning of the conversation. This can help us identify and remove records that do not discuss problems related to games, but rather focus on other non-consequential topics. For example, we found an issue, we came across an issue where the developers were discussing updates and contributions to the repository, rather than addressing any problems related to the game.[10]

Thus after manual refinement of all records, we arrived at a dataset of 1185 text records with 892 *commits*, 189 *issues*, and 104 *pull requests*.

Data format: Our dataset contains fields such as *username*, *name of the game*, *URL*, *text* (contains commit_message, if the record is of the commit, otherwise contains the body of *issues* or *pull requests*), matching text with regular expressions, and a column named *full_content* which contains the combined text data of head, body, and comments in case of *issues* or *pull requests*.

5 https://developer.github.com/v3
6 https://pygithub.readthedocs.io
7 https://developer.github.com/v3/issues
8 https://github.com/redbluegames/game-off-2013
9 https://github.com/lostdecade/onslaught_arena/issues/9
10 https://github.com/KeenSoftwareHouse/SpaceEngineers/issues/584

We made the column *full_content* to analyze the full-text data related to any *issues* and *pull requests*.

2.2. Game development stack exchange data collection

Stack Exchange provides users with the access to the data from their Q&A websites through their data explorer.[11] We used the data explorer to collect the posts from their Game Development forum. The data explorer had a limit of 50,000 posts per requests. We worked around this by splitting the requests to collect posts between smaller time intervals and combining the results. At the end of this step, we ended up with a total of 134,229 posts from April 2014 (start of Game Development Stack Exchange) to July 2022. We preprocessed the data through the following steps.

2.2.1. Removal of posts without an accepted answer

Stack Exchange lets the users answer questions posted on their Q&A forums. Once a question has answers, the questioner can "accept" an answer that helped them resolve their issue. In order to maintain the quality of the data used for our analysis, we filtered out the posts that did not have an accepted answers. Through this, we attempted to ensure that we only considered the posts that had a resolution validated by the questioner. At the end of this step, we had 28,509 posts.

2.2.2. Removal of downvoted posts

Stack Exchange lets the users vote on the posts in their Q&A sites. The number of votes on a post can be used as an indication of its usefulness. The score of a post is defined as the difference between the number of upvotes and the number of downvotes. If a post has a higher number of downvotes than the number of upvotes, it might indicate a lack of usefulness or quality of the post. Hence, we filtered out the questions containing a negative score to ensure the quality of the data. At the end of this step, we had a list of 27,372 posts.

2.2.3. Keyword based filtering

To find the subjects of potential interest, we applied keyword based filtering on the dataset obtained in the previous step. We used the same set of keywords as described in Table 2. We looked for occurrences of these keywords in both the title and the body of the questions. Upon filtering, we obtained a set of 13,269 posts.

2.2.4. Removal of posts with false positive terms

Upon initial inspection of the posts, we found several false positive matches in our dataset. To give a few examples, the keyword "lag" matches with terms like "village" and "lagoon", the keyword "hang" matches with "change". To eliminate these false positive matches, we selected a statistically significant sample of 374 posts with a confidence of 0.95 and a margin of error of 0.05. We manually inspected the sample to look for false positive matches and compile a list of terms that may lead to false positive matches. The terms obtained are as follows

> *coefficient*, *flag*, *lagoon*, *village*, *plague*, *galaga*, *assemblage*, *stalagmites*, *badge badminton*, *badmouth*, *empower*, *firepower*, *powepoint*, *superpower*, *powerless*, *change*, *powershell*

We then filtered out the posts that contain the false positive matches to obtain a dataset of 9374 posts using an automated approach.

[11] https://data.stackexchange.com/

2.2.5. Selection of a sample

Due to a large number of posts, We selected a statistically significant sample of 971 posts with a confidence level of 99.9% and a margin of error of 0.05. It is also noteworthy that "post" in Stack Exchange sites can refer to both a question and answers. Since this sample only contained posts with an accepted answer, the sample could be considered as a dataset of *questions* containing an accepted answer.

2.3. Thematic analysis

To curate the catalogue, we resorted to the commonly used thematic analysis approach (Fereday and Muir-Cochrane, 2006; Petersen et al., 2008) and identified game-specific anti-patterns gathered from *issues*, *commits*, *pull requests*, and *questions*. In total, two researchers and one volunteer were involved in the whole process of thematic analysis. We followed the approach of thematic analysis by implementing below four steps on our dataset:

- *Data Familiarization:* We thoroughly analyzed each record of our dataset and inspected the text data related to *issues*, *commits*, *pull requests* and *questions*. We observed the title, body, and comments (column full_content) of all records and discussed it with co-author and a fellow volunteer.
- *Generating Initial Labels:* Based on records of *commits*, *issues*, *pull requests* and *questions*, we started giving initial codes to the dataset. We initialized some themes such as usability, performance, and many more on an abstract note. We divided the process into several iterations supported by rigorous discussions among both the authors and a fellow volunteer.
- *Identifying and Reviewing Themes:* After analyzing all the instances of our dataset, we discussed and reviewed them to find the relevance of the themes in the context of games. We divided them into subcategories wherever required and also merged the themes accordingly. We decided to discard the themes which occurred for few instances, i.e., less than three times. We also discarded themes that did not point to an issue or a problem as anti-patterns are supposed to be practices that may have bad consequences.
- *Defining Themes:* In this stage, we made an orderly description of each anti-pattern and its occurrence. Section 3 describes all themes observed for game-specific anti-patterns along with their definition and the occurrences in the dataset.

Thematic analysis was conducted by a team consisting of two authors and a volunteer, who were involved in all stages of the analysis process. The entire process took approximately 260 human hours over a period of 2 months. At times, there were disagreements among the researchers regarding the themes identified. However, these disagreements were resolved by analyzing the themes in the context of games, considering how each theme could potentially impact game development (Shi and Shih, 2015). We also found instances where a single record was falling into multiple themes leading to co-occurrences.

In total, 892 *commits*, 189 *issues*, and 104 *pull requests* and 971 *questions* were analyzed during the process of thematic analysis. This extended version specifically focused on the analysis of 971 *questions*. The *commits*, *issues* and *pull requests* were analyzed in our previous version (Agrahari and Chimalakonda, 2022). Altogether, we obtained fifteen game-specific anti-patterns.

Table 3
Themes identified during thematic analysis of GitHub artifacts. *n* denotes the number of occurrences of the theme in GitHub data.

Pattern no.	Themes	Example	Ref.
1	Beware of vague inheritance from game engines/frameworks (n = 65)	*"Chess.js library returns a location string with only a number and causes the player to be locked with the piece, not letting them move it at all. This code checks if the string is the correct length before allowing it through in hideMoves and showMoves to prevent the error from being thrown, as it runs just fine so long as that faulty object is excluded."*	Link
2	Avoid unwanted movement of game objects (n = 63)	*"Fix automovement toggling on "joystick use" flag"*	Link
3	Ensure balance between game controls and functionality (n = 191)	*"When holding down a key, the keyboard buffer fills, causing the associated behavior to continue afterward for some time. For example, if I hold down the W key for a few seconds, the character continues to move forward for some time after releasing the key. This makes the game very difficult to play. Using keyboard events (key down/key up) instead of buffered keyboard input would eliminate this issue."*	Link
4	Avoid wrong locgic/ invalid moves (n = 200)	*"put everything inside a function also fix bug when AI is making invalid move"*	Link
5	Avoid user interface glitch (n = 403)	*"Drag tool makes graphic glitchy. Whenever I use the frag tool, I think the game lags and the graphics becomes glitchy."*	Link
6	Be cautious of platform dependency (n = 252)	*"Hi, I use a Samsung S4 to try the game and you can see from the screenshot below that the Unicode characters for the left button and the right button are not showed."*	Link
7	Avoid memory leaks (n = 53)	*"Running Particle Clicker in Chrome (36, the current stable version) exhibits a memory leak. Looking at Chrome's task manager, memory for the tab grows by up to 180k per second."*	Link
8	Avoid energy extensive components (n = 7)	*"game over was executing even after game over - no need really. lets save memory power"*	Link
9	Provide offline support (n = 6)	*"As mentioned in issue #24, having a cache manifest will enable the game to work offline, especially useful for mobile devices."*	Link
10	Ensure game security (n = 9)	*"Usernames with spaces fail to authenticate"*	Link

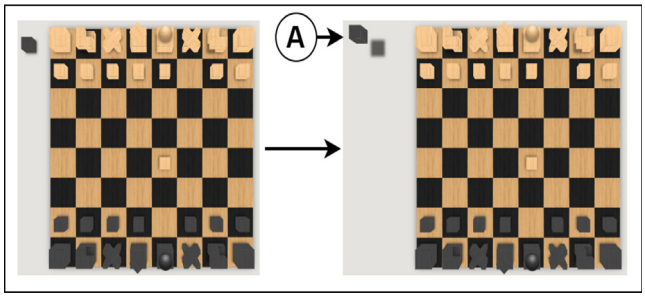

Fig. 2. A scenario of *Unwanted moves* taken from open-source game, *3D Hartwig Chess Set*. Label [A] shows the unwanted movement of jail pieces.

3. Game-specific anti-patterns: Catalogue definition

In this section, we list all the game-specific anti-patterns. We describe each one of them with the following: *context, problem, solution, example(s)* for situation illustrating the occurrence in the case of games and *implications*. For each anti-pattern, we discuss implications for researchers (indicated with the symbol ***R***) and/or practitioners/developers (***P/D***) based on our findings. A detailed discussion on each anti-pattern along with the GitHub link of its occurrences is available online at: https://rishalab.github.io/Catalog-of-Game-Antipatterns/. The themes found in GitHub and Stack Exchange data are presented with examples and references in Table 3 and 4 respectively.

1. **Beware of Vague Inheritance from Game Engines/Frameworks**
   Context: Games are often developed using a framework or game engine that simplifies the task of game developers by providing a set of templates/ modules which prevent them from developing the game from scratch (Lewis and Jacobson, 2002).
   Problem: The anti-patterns inherited can affect functional and non-functional requirements of the game.
   Solution: We should be aware of the anti-pattern present in the game engine/frameworks and should resolve it before using it for the development of the game. Game engines should be examined for the existence of any anti-pattern which can be inherited by the game, and thus should be removed at the time of development.
   Example: Consider a board game using a chess library, that is used for chess piece placement/movement, move generation/validation, and check/checkmate/stalemate detection. Thus, if there exists an issue in any one of the rules of checkmate detection, then it can lead to the wrong detection for the game, that gets inherited from the chess library.
   Implications: **R** can explore this domain more in the context of games and how vague inheritance of anti-patterns could lead to unwanted consequences in game quality. ***P/D*** should keep track of the anti-patterns present in the development environment to avoid unexpected technical debt in the game.
2. **Avoid Unwanted Movement of Game Objects**
   Context: Player moves are one of the important parts of gameplay. Player movements should be logical, consistent and optimized (Korhonen and Koivisto, 2006).
   Problem: If the game does not follow the expected behavior, it causes hindrance in the usability of the game. It does not cause any violation in the game rules, but it makes the game clumsy to play.
   Solution: Developers should keenly focus on this anti-pattern by planning out the object movements in advance which is logical and expected by the end-user.

Table 4
Themes identified during thematic analysis of Game Development Stack Exchange posts. *n* denotes the number of occurrences of the theme in Stack Exchange data.

Pattern no.	Themes	Example	Ref.
5	Avoid user interface glitch (n = 353)	*"so once upon a time I made a really cool character for my game project and he looked great. Then at some unspecific point in time, my character's UV seams got these awful white outlines around them"*	Link
7	Avoid memory leaks (n = 14)	*"the thread running the callback is using 63% CPU and the RAM is constantly rising, slowly (memory leak?). I fixed it up a bit, before RAM used was 2.32 GB."*	Link
8	Avoid energy extensive components (n = 6)	*" What matters? Battery usage, i've played some simple games that used so much battery compared to their complexity and graphics and i don't wanna to create that battery hungry game."*	Link
10	Ensure game security (n = 24)	*'Instead, you should send control input to the server, validate that it is practical (pressing the "attack" button 1000 times in 1 s, for example, is not valid), then update the simulation on the server. The client itself is the main place people can cheat, so programs like Punkbuster can cut down on wallhacks, aimbots, etc. "*	Link
11	Beware of pitfalls while using APIs (n = 51)	*"As you can see from this code, the way it batches things up internally is by checking the TextureID of each item in the _batchItemList and if it has changed, flushes the vertex array and binds a new texture. This approach is pretty typical of how sprite batching works in most 2D engines (from my understanding) and usually turns out okay. However, if you have too many texture switches it can be pretty costly on performance. The usual way to deal with this in a tile based engine is to use a texture atlas."*	Link
12	Follow conventions while distributing the tasks between the client and the server (n = 60)	*"The obvious problem with B is that you can never trust the client. If I depend on the client telling me that the player got hit really hard and that he did, in fact, throw the player somewhere, that's an obvious security hole. The user can make a simple script that blocks any messages with a 'he got hit' header or something, and become invincible."*	Link
13	Take measures to eliminate lag between client and server (n = 125)	*"..the problem that immediately became apparent is that the user's character moved slower when the server was lagging. This meant that chasing enemies were able to easily catch up to the character and kill him.."*	Link
14	Ensure correct computation of vectors (n = 52)	*"Common solutions say that the delta rotation can be calculated by qDelta = qFrom.inverse() * qTo, which I use as well and works fine for most cases. However, I am often running into issues for some specific rotations. Inverting the calculated delta rotations helps for some cases, but doesn't cover all cases."*	Link
15	Beware of legal and licensing issues (n = 38)	*"On OS X and Windows games like this often allow for users to install fan-made databases - which essentially overwrites the game's default data with (fan-made) "real word" data, thus circumventing intellectual property/likeness usage issues for the developer(s). Is this legal?"*	Link

Example: Consider the case of a board game such as chess, which requires grabbing and moving the pieces in the game. Thus, the player should not be allowed to move jail pieces (as shown in Fig. 2). Similarly, there should be a check on moving pieces so that the players do not move the piece, which is not of his/her color. It avoids unnecessary extra work and needless confusion for end-users.
Implications: For researchers ***R***, finding out the frequent occurrences of this anti-pattern and the steps leading to them can be a interesting research direction in the context of game. For developers ***P/D***, it can help in optimizing the game object movements, thus leading to better performance of game.

3. **Ensure Balance Between Game Controls and Functionality**
Context: Game control is an integral part of the gameplay. Having effective, smooth and minimized movements of various devices such as keyboard, joystick, mouse, etc. increases the usability of game (Korhonen and Koivisto, 2006).
Problem: Rough and faulty game controls can have a negative impact on usability factor.
Solution: Optimized game actions along with smooth game controls increase hassle-free gameplay.
Example: In the case of point and click game, there should not be any action related to the keyboard or any other device, as it causes unnecessary control movements for the player.

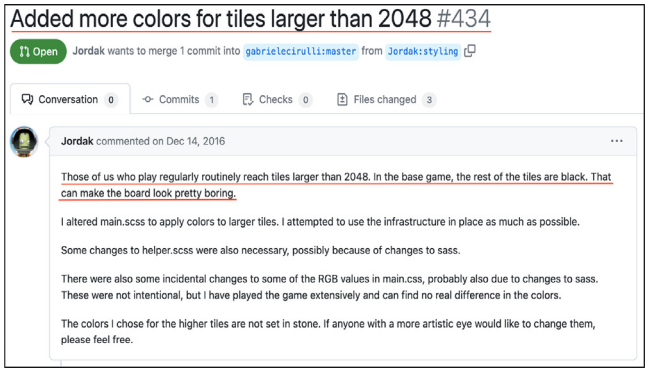

Fig. 3. The screenshot is taken from a GitHub *pull request* of the game *2048*, where a user talks about more colors for tiles as less colors make game boring. Underlined sentences highlights the text discussing about the UI Design of game.

Implications: Researchers ***R*** can explore ways to address this anti-pattern by proposing different ways to optimize game controls with maximum game features coverage while ***P/D*** could provide game with minimum mouse scrolls, keyboard buttons, and other devices.

4. **Avoid Wrong Logic/Invalid Moves**
Context: Game design involves the implementation of many small modules that results in one big module. These small

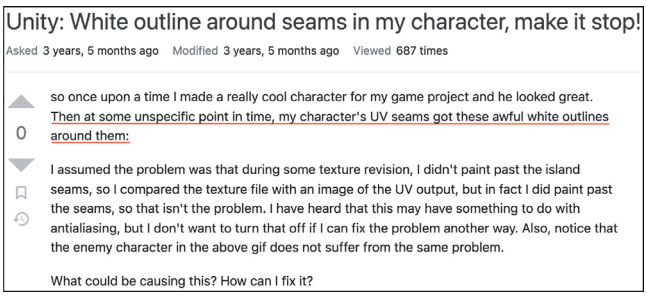

Fig. 4. Example of a Stack Exchange post involving UI glitch.

modules of functionalities should be implemented properly with proper logic; otherwise, it can lead to a faulty game. Invalid moves lead to a violation of game rules and thus creates confusion for users.

Problem: Violation of game rules through invalid moves can destroy the likeness of the game in player's mind as it highlights the loopholes in a game.

Solution: Wrong Logic/Invalid moves should be strictly limited by the developers by carefully defining the control-action of the game. There should be precise "Game Description Language" which can guide the developer on each step (Thielscher, 2010).

Example: In a chess game, if the game design allows illegal moves also in the game, without any objection, then the game may lose its purpose of play.

Implications: This anti-pattern opens up new research direction for researchers ***R*** to explore it deeply and how it affects the gameplay. Further, open-source game developers ***P/D*** should particularly try to avoid and check invalid moves in their game.

5. **Avoid User Interface Glitch**

Context: UI is an essential component of game display that can attract or frustrate the user.

Problem: Bad UI design can lead to a displeasing display of the game. Issues which cause problems in the audio/video of games affects the UI of game and can lead to demotivation in the player for playing the game.

Solution: The UI design should be visually pleasing and should follow the UI heuristics of game (Johnson and Wiles, 2003).

Example 1: The tile size in the board related games should be of average size. It should not be too big or too small. Another instance can be shown in the screenshot (Fig. 3) that discusses tiles of more colors in the game 2048.

Example 2: The post with ID 176381[12] shown in Fig. 4 explains a UI glitch that leads to white outline around seams in the character usually caused by mipmapping as explained in the answer.[13]

Implications: Open-source developers ***P/D*** should specially emphasize on UI of game by conducting UI quality assurance test. They should test the compatibility of game with various devices and should ensure proper layout, content, images and font.

6. **Be Cautious of Platform Dependency**

Context: Games often come with a set of dependencies. Different platforms and versions might be required to execute the game. Often the installation process of a game takes more time than expected due to version problem, error in Makefile, and other reasons.

Fig. 5. Screenshot taken from one a GitHub *issue* of the game *AlienInvansion*, where the user is not able to view the game controls properly on a particular Chrome version and specific mobile phone.

Problem: If the platform dependencies are not stated explicitly, it may cause faulty installation and demotivate users.

Solution: Documentation of the game dependencies must be provided to end-users to ensure hassle-free game play.

Example: Consider a game running smoothly on a Windows laptop, but not on Mac PC. Further, the developers did not provide any information about the same. Another example can be related to the processor requirement of games. All the requirements and dependencies should be documented beforehand so that it does not effect the gaming experience of a user. Fig. 5 shows a screenshot of an *issue* related to platform dependency of game, where the user faces problem in viewing the icons and game controls because of platform dependency.

Implications: All the steps and requirements related to platform dependency of the game should be properly documented involving lower-level details by the developer ***P/D***; thus the user will not have to struggle on the execution of game.

7. **Avoid Memory Leaks**

Context: Games involve different game objects and assets, which consumes a lot of memory. This memory storage should be used wisely; otherwise it can cause memory leak.

Problem: Memory leak causes reduction in available memory for usage, which causes bad performance.

Solution: Memory leak can be reduced by avoiding number of case scenario such as : multiple references to the same object, creating huge object tree, assurance of proper garbage collection of unused variable.

Example 1: Consider an arcade game, where different game objects keep on appearing and disappearing in the gameplay. These objects need to be handled carefully in the memory to avoid a memory leak. Fig. 6 shows a screenshot of game where developer amend the code to remove the problem of memory leak in game.

Example 2: Consider the example from the post[14] where every single loop of the callback function re-creates buffer, reloads texture, and recompiles shader leading to memory leak. It can be avoided by not recreating everything on every loop as shown in Fig. 7.

Implications: To avoid memory leak problem, developers ***P/D*** should free up the unnecessary variables and game objects, and should use the idea of data locality (Nystrom,

12 https://gamedev.stackexchange.com/questions/176381/
13 https://gamedev.stackexchange.com/a/176390
14 https://gamedev.stackexchange.com/questions/120440/

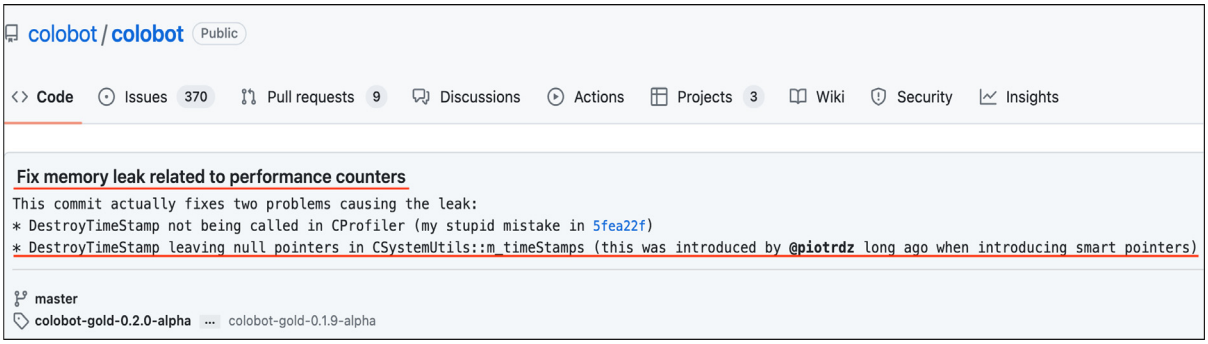

Fig. 6. Screenshot of a GitHub *commit* done by developer to rectify the memory leak problem caused by usage of null pointer in game *Colobot*.

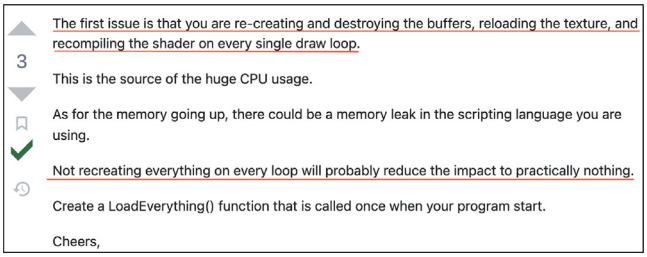

Fig. 7. Stack Exchange post explaining method to avoid memory leaks.

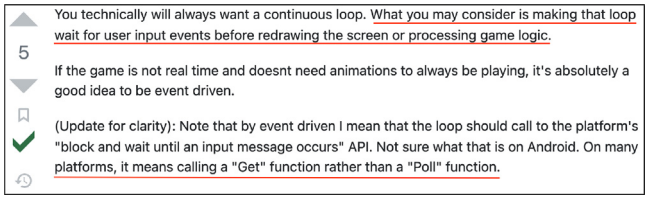

Fig. 8. Example of a Stack Exchange post suggesting method to save energy in an Android game.

2014). Developers should use the object pool where memory can be reused instead of allocating and freeing them every time (Nystrom, 2014).

8. **Avoid Energy Extensive Patterns**
   Context: Sustainable software development has become one of the key requirement of current era. Any game should be sustainable and energy-efficient which consumes optimized power (Capra et al., 2012). It must be designed in a way so that it fulfills the functional and non-functional requirement with minimum power usage.
   Problem: More power consumption leads to faster power drainage.
   Solution:Developers should avoid energy anti-pattern in game (Cruz and Abreu, 2019), and should utilize the process of sustainable software development (Amsel et al., 2011).
   Example 1: Consider a game where graphics and animation in games are active even after game over. This causes unnecessary power consumption.
   Example 2: Consider an input -update-render cycle in an OpenGL Android game. Making the loop wait for user input events before redrawing on screen may save battery and improve energy efficiency as shown in Fig. 8 taken from the post with ID 29753[15]
   Implications: Optimization of energy has been focused by researchers (Cruz and Abreu, 2019; Amsel et al., 2011), however there is need to further explore energy consumption in the context of games more deeply. ***R*** can emphasize on the possible ways on how game can be more interactive with optimized power consumption. Further, practitioners and open-source developers ***P/D*** should follow good practices to reduce the power usage of their of games.
9. **Provide Offline-Support**
   Context: In recent times, an increasing number of games have adopted a browser-based approach, allowing users to play the game directly without the need for installing any software. This eliminates potential prerequisites and simplifies the user experience.
   Problem: Sometimes users get disconnected from the internet in the middle of the game because of which they may lose access to the game.
   Solution: Provide offline support for the game, so that people can save their game to the home screen, and play offline.
   Example: *2048* is a *browser-based* game with offline support, facilitating its users to save the state of the game when there are connection issues.
   Implications: *Offline Support* is the functionality that should be considered by developers ***P/D*** in the case of *browser-based* games for better acceptance across players' community.
10. **Ensure Game Security**
    Context: Security plays an important role in gameplay to protect a user's identity and his/her game achievements. It ensures that untrusted clients do not intrude into the game. It also ensures that players do not take unfair advantage in the game by doing wrong practices to win, such as cheat codes, compromised environment, and so on (Yan and Choi, 2002).
    Problem: Untrusted players can become a threat to the security of the game. Unethical means of achieving the target in gameplay may downgrade the game.
    Solution: Following all the heuristics related to security issues in games (Yan and Choi, 2002) is key solution to this anti-pattern.
    Example 1: Consider an RPG (Role Playing Game), where the user makes improper usage of cheat codes to achieve the target.
    Example 2: In MMO games, a few players might be prone to cheating. A way to handle this is to not trust anything sent by the client and implementing validation of client inputs for practicality on the server side. The example is

15 https://gamedev.stackexchange.com/questions/29753/

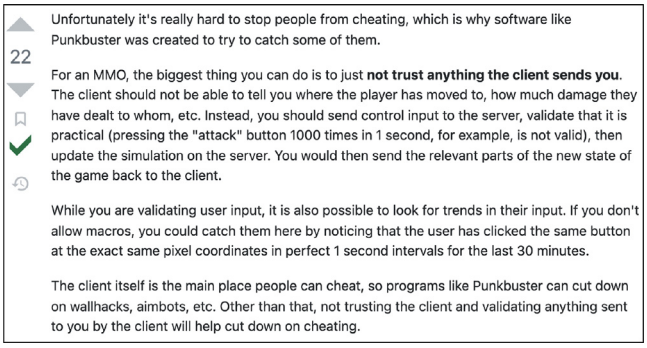

Fig. 9. Stack Exchange post explaining ways to deal with cheating from client's side in MMO games.

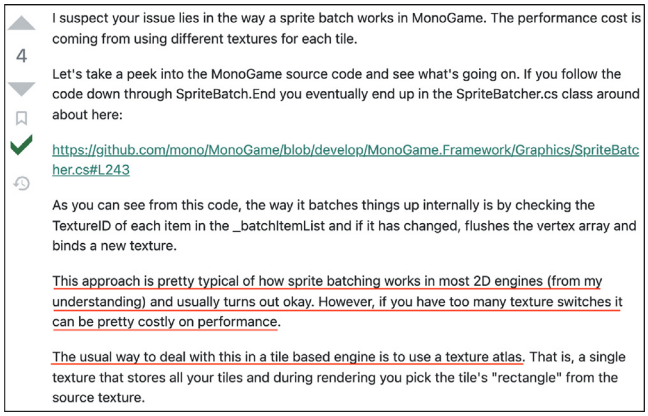

Fig. 10. Example of a Stack Exchange post showing a pitfall while using *SpriteBatch* in *Monogame*.

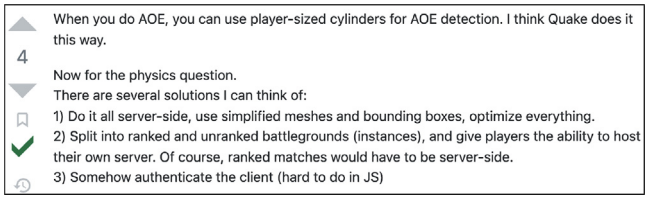

Fig. 11. Stack Exchange post explaining resolution to the server capacity issue without performing the calculations on the client.

taken from the post.[16] shown whose accepted answer is shown in Fig. 9

Implications: Developers ***P/D*** should ensure that there is a proper channel of authentication for players that checks for insecure passwords, multiple logins, sniffed passwords, and so on. They should check for loopholes that can compromise the security of game.

11. **Beware of Common Pitfalls While Using APIs**

Context: The use of game engines/frameworks and external libraries is common in game development (Lewis and Jacobson, 2002). The APIs provided by frameworks/engines and external libraries are used for quick and convenient implementation of the functionalities based on the requirements of the game.

Problem: APIs may be useful in implementing the required functionalities. However, their incorrect or improper usage due to a lack of proper understanding of their usage and working may cause the game to malfunction and lead to inefficiencies. This may impact the game's functionality and the performance.

Solution: This can be avoided by being aware of common pitfalls of while using the APIs and take appropriate steps to deal with them.

Example: Consider the use of *SpriteBatch* in *Monogame* framework to render multiple tiles on screen. Saving each tile in a different texture slows down the rendering on large screens due to *SpiteBatch's* internal handling of texture switches. Being aware of this pitfall and using texture atlas to deal with the issue can resolve the issue of slow rendering. The example is based on the post.[17] shown in Fig. 10

Implications: Developers ***P/D*** should refer to the API documentation and examples to gain an understanding of standard ways to use APIs. They should also develop an understanding of the common pitfalls and their root causes while using the APIs so that they can be avoided.

12. **Follow Conventions While Distributing Tasks Between the Client and the Server**

Context: Networked games require the use of client–server architecture (tong Cai et al., 2002). The servers typically handle multiple clients (tong Cai et al., 2002). This setup has the potential induce several network oriented issues and vulnerabilities.

Problem: Violating the conventions and standard practices while distributing the tasks between the client and the server to deal with the latency or minimize resource usage may leave vulnerable loopholes for exploitation and cheating.

Solution: This can be avoided by adhering to standard practices while distributing the tasks between client and server. The tasks that concern to inputs and changing the visuals should be done on the client while the server should solely be responsible for managing the game state and the logic.

Example: Consider the example from the post[18] shown in Fig. 11 where the developer wants to implement physics and AoE spells in a massively multiplayer online game. The servers lack enough capacity and may cause lag. However, to overcome this issue, the calculations should never be performed on the client side. Instead, techniques such as simplified meshes, bounding boxes, optimizations and having players host their own servers should be used.

Implications: Developers ***P/D*** should adhere standard practices while designing networked games with respect to the tasks performed on the client and the server.

13. **Take Measures to Eliminate Lag and Synchronization Issues Between Client and Server**

Context: Networked games are supported by distributed client–server architecture (tong Cai et al., 2002). Due to this, a constant communication is required between the nodes in the network during the gameplay

Problem: Weak network, disconnections and server latency can induce lag and cause a bad gaming experience. Lack of proper synchronization can lead to variation in game states on client and the server.

Solution: Lag can be handled using methods such as client-side predictive logic and latency minimization techniques. Synchronization can be achieved through use of reliable data transmission protocols.

16 https://gamedev.stackexchange.com/questions/1520/

17 https://gamedev.stackexchange.com/questions/82799/

18 https://gamedev.stackexchange.com/questions/29384/

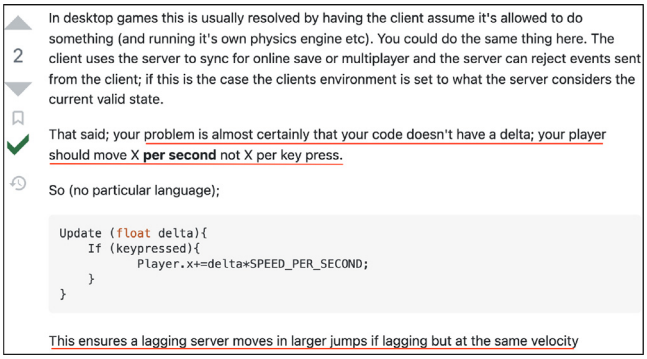

Fig. 12. Answer on Stack Exchange explaining resolution to the issue caused due to server lag.

Example: Consider the example from the post[19] where the user faces the issue of the character moving slower due to server lag. Due to this problem, the character gets caught by the enemies and is killed. This issue can be resolved as shown in Fig. 12 by modifying the logic to move the character based on time duration rather than the number of key-presses.
Implications: Developers ***P/D*** should ensure to account for lag and synchronization issues while designing networked games. Researchers ***R*** can explore the best practices in dealing with network related issues in games.

14. **Ensure Correct Computation of Vectors**
Context: Many games involve movements of objects. Vectors are fundamental in representing the aspects of objects such as position, direction, velocity, acceleration, momentum and so on.
Problem: Incorrect computation while updating the vectors associated with the objects can cause incorrect movements and produce unstable results.
Solution: This can be avoided by ensuring the use of correct formulas from mathematics and physics domain to update the vectors associated with the object.
Example: Consider the example from the post[20] shown in Fig. 13 where the programmer wants to compute the relative quaternion rotation of an object. Use of incorrect formula produces unexpected results for some specific rotations. This can be overcome by using the correct vector update formula as provided in the accepted answer[21]
Implications: Developers ***P/D*** should gain adequate understanding of vector mathematics and refer to accurate formulas when developing games that involve movement of objects.
15. **Beware of Legal and Licensing Issues**
Context: Modern game development involves leveraging third party products to reduce the workload on the developers. Components such as music, images, characters, fonts and trademarks among others are used as third party components. Many games may also require the storage of user data.
Problem: The external components may come with patents, copyright, license, and intellectual property issues. Games that have to adhere to data privacy laws (Bennett and Oduro-Marfo, 2018) with their user data which can vary based on country and region.

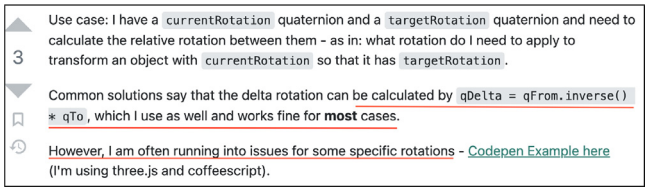

Fig. 13. Example of a Stack Exchange post with incorrect vector computation.

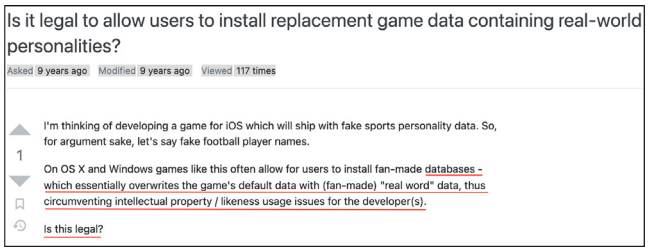

Fig. 14. Example of a Stack Exchange post asking about possible legal issues with the game.

Solution: To deal with the issues related to legal matters, a legal expert should be consulted before releasing the game.
Example: An instance of this pattern can be seen in the post[22] shown in Fig. 14. The developer asks about the implications of allowing users to have names of real-world personalities in the game.
Implications: Game Developers ***P/D***, especially when aiming to commercialize their games must consult legal experts to deal with aspects such as licensing, copyright, intellectual property and laws concerning data privacy.

4. Threats to validity

In this section, we discuss the potential threats to our research study, and how far we can generalize the catalogue.

4.1. Internal validity

We performed regular expression based search to find the *issues*, *commits*, and *pull requests* that may contain anti-pattern. There is the possibility of missing some relevant text data if they are not covered by the regular expression. We attempted to minimize this threat by performing an exhaustive manual search to collect the list of keywords that may correspond to anti-patterns. We considered the English language primarily, assuming it is the most commonly spoken language among developer communities. As part of our analysis, we only considered Stack Exchange *questions* that had an accepted answer. We acknowledge that there may be cases where questioners do not accept an answer even if it helped them resolve the issue. However, we made a deliberate decision to exclude unaccepted answers to maintain the quality and reliability of our data, as there is no way to ascertain the validity of such answers.

During the manual filtering of GitHub artifacts, we made efforts to eliminate all cases of false positives to the best of our ability. For Stack Exchange data, due to the large number of posts, we automated the removal of posts containing false positive matches based on manual inspection. However, it is possible that a few false positives may have been inadvertently included in our

19 https://gamedev.stackexchange.com/questions/114633/
20 https://gamedev.stackexchange.com/questions/143430/
21 https://gamedev.stackexchange.com/a/144693
22 https://gamedev.stackexchange.com/questions/72695

dataset, either due to errors during manual deletion or because some false positive matches may have gone unnoticed during manual inspection. Nevertheless, we argue that any such false positives were discarded during the thematic analysis.

Due to the substantial size of the Stack Exchange posts even after the removal of false positives, we opted to analyze a sample of 971 *questions* for thematic analysis. However, we acknowledge that sampling may pose a potential threat of the sample not being fully representative of the entire population. To mitigate this risk, we employed measures to ensure statistical significance using a high confidence level in our sample selection and randomized the selection to minimize bias.

We chose *commits*, issues, and *pull requests* from GitHub and *questions* from Game Development Stack Exchange to analyze the commonly occurring anti-patterns faced by end-users and developers. Nonetheless, there is a possibility that other methodologies could have resulted in different set of anti-patterns. To the best of our efforts, we obtain the catalogue by in-depth analysis and believe that it is factual. But, since it also involved manual interpretation, there is a possibility of human error.

4.2. External validity

The game repositories analyzed in this study exclusively consist of open-source ones available on GitHub. Commercial or paid games that do not have publicly accessible repositories may exhibit different characteristics and issues. Some issues related to non open-source games may also be covered in Stack Exchange questions, as developers often use this platform for assistance regardless of whether they are working on open-source or non open-source projects. However, the anti-patterns identified in this study can be applied to any game, irrespective of its license.

The Game Development Stack Exchange is a general forum for game development. Hence, the nature of the data may be different from game engine specific forums such as Unity Forum.[23] There is a scope for inclusion of data from framework/engine specific forum as well. Regardless, the anti-patterns obtained can be useful irrespective of game engine used for development.

5. Discussion

In this paper, we attempted to present a catalogue of fifteen game-specific anti-patterns obtained through the process of thematic analysis of artifacts from GitHub and Game Development Stack Exchange. We believe that this catalogue might help developers in minimizing the bad practices during game development. Although some of the anti-patterns we discussed are available in the literature, but in the context of games, we present this catalogue with the intention of aggregating all anti-patterns at one place. While we understand that there are a few broad categories in our catalogue, we kept it wide, as they are already discussed in detail in the literature.

From the results of the thematic analysis, we could observe that there were a few patterns observed in GitHub data that could not be observed in Game Development Stack Exchange data and vice versa. This could be due to the difference in the nature of the platforms and the users using them. On GitHub, the *issues*, *commits* and *pull requests* are very specific to their projects. On Q&A forums, the discussions tend to be more general as there is limited scope to contextualize the *questions* based on projects due to the nature of the platform. As an example, in the *Avoid Unwanted Movement of Game Objects* pattern, most occurrences in the catalogue are commits to fix issues that allowed users to make unwanted movements in the game based on issues raised by the users/testers. As issues regarding a game from end user/tester are unlikely to be raised on a programming Q&A forum, we do not find instances of that pattern on Game Development Stack Exchange.

We also observe that out of the 971 *questions* analyzed, only 723 could be classified under an anti-pattern. The remaining *questions* were under the themes that were discarded during the thematic analysis. However, in the case of GitHub data, all 1185 records were classified under one or more anti-patterns. We suspect that the nature of Q&A platform may be the reason for a high discard of themes from Game Development Stack Exchange as it has a higher chance of novice level and general questions. It should also be noted that the frequency of occurrences vary across the anti-patterns with a few patterns such as *"Avoid energy extensive components"* having a relatively low number of occurrences. However, we have included them in the catalogue as the importance of an anti-pattern should not solely be determined by its frequency of occurrence. This also aligns with catalogues in other domains obtained using data from GitHub and Stack Overflow (Cruz and Abreu, 2019; Valenzuela-Toledo and Bergel, 2022).

We believe that this catalogue can serve as a valuable guide for game developers to avoid the listed anti-patterns during game development. While it can be beneficial for all game developers, it may be particularly useful for open-source game developers who may lack guidance on best practices to avoid during game development. Paid and commercial game developers typically have dedicated development teams, but they can still benefit from the catalogue by taking appropriate actions to avoid mistakes. Additionally, researchers and practitioners can use the dataset and catalogue for further empirical studies. We recognize the need to continuously explore anti-patterns and bad practices in depth to strengthen the dataset and catalogue for future research.

6. Related work

Game development has gained a fair amount of attention from researchers since the past decade. Game based software development require distinctive discussion from the research point of view (Murphy-Hill et al., 2014; Wesley and Barczak, 2016). Possible reasons for the difference in game development with that to the traditional software development is in the context of various performance and real-time factors being involved, such as memory allocation and de-allocation issues, rendering process, and graphical components (Nystrom, 2014; Politowski et al., 2016; Kanode and Haddad, 2009). Due to this, researchers have studied factors that differentiate game development from the regular software development (Murgia et al., 2014; Pascarella et al., 2018). As an example, Murphy-Hill et al. (2014) conducted a survey study with the game and non-game developers to find the substantial differences between the two, and found that the video game developers require a more creative mind, good knowledge of maths, and performance tuning in comparison to non-game developers. Similarly, Pascarella et al. (2018) studied 60 open-source projects to differentiate between games and non-games and analyzed that project organization, developers skills, automated testing, code reuse, and many other aspects differ in both the domains. In a preliminary work, Khanve (2019) has shown that the code-specific bad practices or code smells in games can be different from those of code smells in other software by manually analyzing the violation of game programming patterns in eight *JavaScript* games and concluded that games need to be handled separately in terms of code smells.

Given the unique nature of game development compared to conventional software development, efforts have been made to investigate the challenges and issues specific to the game industry. Petrillo et al. (2008) proposed four main categories for issues

[23] https://forum.unity.com/

related to computer game development by analyzing game postmortems: *Schedule Problems, Budget Problems, Quality Problems, Management Problems,* and *Business related problems*. Developers' postmortem reports have been utilized to create a dataset that captures the software problems encountered by them (Politowski et al., 2020). The effects of time pressure over the quality of games (Borg et al., 2019) has also been studied. The study revealed that the teams rely on an ad-hoc approach to develop and share contextual similarities to software startups.

Along with the game-related problems, researchers also proposed solutions to a few of these problems. Varvaressos et al. (2017) proposed automated runtime bug finding for video games based on *game loop*. They experimented on six real-world games and tracked their bugs based on the games' bug database. Researchers propose various heuristics and measures to analyze the usability and friendly user interface of games (Korhonen and Koivisto, 2006). Truelove et al. (2021) proposed a taxonomy of bug types in games help developers identify the aspects of game development that could benefit from greater attention. Albaghajati and Ahmed (2022) have tried the approach of using co-evolutionary genetic algorithms to automatically detect bugs in video games. In a recent study, Borrelli et al. proposed a set of game-specific bad smells in Unity Projects (Borrelli et al., 2020) with the aim of helping developers avoid them.

Despite the large availability of data on platforms such as GitHub and Stack Exchange, the above mentioned studies have not utilized them to study the issues and practices of game developers. There exist a number of empirical studies that rely on GitHub data in other domains such as release engineering (Joshi and Chimalakonda, 2019), code quality (Ray et al., 2014), software evolution (Silva et al., 2016), and so on. They have been utilized to study software developer practices (Cruz and Abreu, 2019; Moura et al., 2015). Likewise, the popular sites from Stack Exchange network such as Stack Overflow has also been used to study developer practices and challenges (Shanbhag et al., 2022; Zhang et al., 2019; Uddin et al., 2021). GitHub and Stack Exchange are a vast source of information that could be considered to gain a better understanding of problems faced by game developers. Our study takes a step in that direction. Both the sources have been used in development of catalogue of anti-patterns in domains such as mobile development (Cruz and Abreu, 2019), deep learning (Shanbhag et al., 2022). Along a similar line, our work aims to propose a catalogue of anti-patterns in game development domain.

In our previous work (Agrahari and Chimalakonda, 2022), we had proposed a game anti-pattern catalogue by analyzing GitHub artifacts. However, it did not include the analysis of data from Q&A platforms. Hence, we extend our catalogue with the data from Game Development Stack Exchange in this paper.

7. Conclusion and future work

We proposed a catalogue of fifteen anti-patterns in the context of games using a dataset of *commits*, *issues*, *pull requests* from 100 popular open-source games available on GitHub and 971 *questions* from Game Development Stack Exchange. We analyzed the textual data in the dataset, performed thematic analysis and presented the catalogue of game-specific anti-patterns. Among the list of proposed anti-patterns, *Avoid User Interface Glitch* is the most frequent anti-pattern followed by *Be Cautious of Platform Dependency*. Since the catalogue proposed is prevailing in the context of games, it can help game developers make informed decisions towards improving quality of games during development.

We see this work as a step towards further empirical research in the largely under-explored domain of games. We plan to extend this research work by deep diving into game-specific code smells. We also plan to explore automatic and semi-automatic approaches and tools for detection of game-specific anti-patterns and code smells specifically leveraging advances in Artificial Intelligence, Machine Learning and Natural Language Processing.

The catalogue is based on artifacts available of a public Q&A forum and a code hosting platform. In the future, we plan to conduct a comprehensive survey involving game developers to analyze the usefulness of our catalogue and its impact of the development process. This would help us further validate the anti-patterns obtained from our analysis.

CRediT authorship contribution statement

Vartika Agrahari: Methodology, Software, Data curation, Formal analysis, Investigation, Writing – original & draft, Validation. **Shriram Shanbhag:** Software, Data curation, Formal analysis, Investigation, Writing – original & draft. **Sridhar Chimalakonda:** Conceptualization, Supervision, Project administration. **A. Eashaan Rao:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Abidi, M., Khomh, F., Guéhéneuc, Y.-G., 2019. Anti-patterns for multi-language systems. In: Proceedings of the 24th European Conference on Pattern Languages of Programs. pp. 1–14.

Agrahari, V., Chimalakonda, S., 2022. A catalogue of game-specific anti-patterns. In: 15th Innovations in Software Engineering Conference. pp. 1–10.

Albaghajati, A., Ahmed, M., 2022. A co-evolutionary genetic algorithms approach to detect video game bugs. J. Syst. Softw. 188, 111261.

Amsel, N., Ibrahim, Z., Malik, A., Tomlinson, B., 2011. Toward sustainable software engineering: NIER track. In: 2011 33rd International Conference on Software Engineering. ICSE, IEEE, pp. 976–979.

Anderson, C.A., Bushman, B.J., 2001. Effects of violent video games on aggressive behavior, aggressive cognition, aggressive affect, physiological arousal, and prosocial behavior: A meta-analytic review of the scientific literature. Psychol. Sci. 12 (5), 353–359.

Bao, L., Lo, D., Xia, X., Wang, X., Tian, C., 2016. How android app developers manage power consumption?-an empirical study by mining power management commits. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories. MSR, IEEE, pp. 37–48.

Bennett, C., Oduro-Marfo, S., 2018. GLOBAL privacy protection: Adequate laws, accountable organizations and/or data localization? In: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers. pp. 880–890.

Björk, S., Holopainen, J., 2006. Games and design patterns. Game Des. Read. 410–437.

Borg, M., Garousi, V., Mahmoud, A., Olsson, T., Stalberg, O., 2019. Video game development in a rush: A survey of the global game jam participants. IEEE Trans. Games.

Borges, H., Valente, M.T., 2018. What's in a GitHub star? understanding repository starring practices in a social coding platform. J. Syst. Softw. 146, 112–129.

Borowa, K., Zalewski, A., Saczko, A., 2021. Living with technical debt—A perspective from the video game industry. IEEE Softw. 38 (06), 65–70. http://dx.doi.org/10.1109/MS.2021.3103249.

Borrelli, A., Nardone, V., Di Lucca, G.A., Canfora, G., Di Penta, M., 2020. Detecting video game-specific bad smells in unity projects. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 198–208.

Brown, W.H., Malveau, R.C., McCormick, H.W., Mowbray, T.J., 1998. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons, Inc.

Capra, E., Francalanci, C., Slaughter, S.A., 2012. Is software "green"? Application development environments and energy efficiency in open source applications. Inf. Softw. Technol. 54 (1), 60–71.

Carvalho, S.G., Aniche, M., Veríssimo, J., Durelli, R.S., Gerosa, M.A., 2019. An empirical catalog of code smells for the presentation layer of android apps. Empir. Softw. Eng. 24 (6), 3546–3586.
Cruz, L., Abreu, R., 2019. Catalog of energy patterns for mobile applications. Empir. Softw. Eng. 1–27.
Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A., 2020. Towards a catalogue of software quality metrics for infrastructure code. J. Syst. Softw. 110726.
Edholm, H., Lidström, M., Steghöfer, J.-P., Burden, H., 2017. Crunch time: The reasons and effects of unpaid overtime in the games industry. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track. ICSE-SEIP, IEEE, pp. 43–52.
Fard, A.M., Mesbah, A., 2013. Jsnose: Detecting javascript code smells. In: 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation. SCAM, IEEE, pp. 116–125.
Fereday, J., Muir-Cochrane, E., 2006. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. Int. J. Qual. Methods 5 (1), 80–92.
Fowler, M., 1997. Refactoring: Improving the design of existing code. In: 11th European Conference. Jyväskylä, Finland.
Garcia, J., Popescu, D., Edwards, G., Medvidovic, N., 2009. Toward a catalogue of architectural bad smells. In: International Conference on the Quality of Software Architectures. Springer, pp. 146–162.
Giessen, H.W., 2015. Serious games effects: an overview. Procedia Soc. Behav. Sci. 174, 2240–2244.
Gregory, P., Barroca, L., Taylor, K., Salah, D., Sharp, H., 2015. Agile challenges in practice: a thematic analysis. In: International Conference on Agile Software Development. Springer, pp. 64–80.
Guamán, D., Pérez, J., Valdiviezo-Diaz, P., Canas, N., 2022. Estimating the energy consumption of software components from size, complexity and code smells metrics. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 1456–1459.
Hübener, T., Chaudron, M.R., Luo, Y., Vallen, P., van der Kogel, J., Liefheid, T., 2022. Automatic anti-pattern detection in microservice architectures based on distributed tracing. In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, IEEE, pp. 75–76.
Johnson, D., Wiles, J., 2003. Effective affective user interface design in games. Ergonomics 46 (13–14), 1332–1345.
Joshi, S.D., Chimalakonda, S., 2019. RapidRelease-A dataset of projects and issues on github with rapid releases. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 587–591.
Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D., 2014. The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, pp. 92–101.
Kamienski, A., Bezemer, C.-P., 2021. An empirical study of Q&A websites for game developers. Empir. Softw. Eng. 26 (6), 1–39.
Kanode, C.M., Haddad, H.M., 2009. Software engineering challenges in game development. In: 2009 Sixth International Conference on Information Technology: New Generations. IEEE, pp. 260–265.
Kermansaravi, Z.A., Rahman, M.S., Khomh, F., Jaafar, F., Guéhéneuc, Y.-G., 2021. Investigating design anti-pattern and design pattern mutations and their change-and fault-proneness. Empir. Softw. Eng. 26 (1), 1–47.
Khanve, V., 2019. Are existing code smells relevant in web games? an empirical study. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 1241–1243.
Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. 28 (8), 721–734.
Klug, M., Bagrow, J.P., 2016. Understanding the group dynamics and success of teams. Royal Soc. Open Sci. 3 (4), 160007.
Korhonen, H., Koivisto, E.M., 2006. Playability heuristics for mobile games. In: Proceedings of the 8th Conference on Human-Computer Interaction with Mobile Devices and Services. pp. 9–16.
Lee, K.M., Peng, W., 2006. What do we know about social and psychological effects of computer games? A comprehensive review of the current literature. Play. Video Games Motiv. Responses Consequences 327–345.
Lewis, M., Jacobson, J., 2002. Game engines. Commun. ACM 45 (1), 27.
Lin, T., Fu, X., Chen, F., Li, L., 2021. A novel approach for code smells detection based on deep leaning. In: EAI International Conference on Applied Cryptography in Computer and Communications. Springer, pp. 171–174.
McKenzie, T., Morales-Trujillo, M., Lukosch, S., Hoermann, S., 2021. Is agile not agile enough? A study on how agile is applied and misapplied in the video game development industry. In: 2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering. ICGSE, IEEE, pp. 94–105.
Moura, I., Pinto, G., Ebert, F., Castor, F., 2015. Mining energy-aware commits. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, pp. 56–67.
Murgia, A., Tourani, P., Adams, B., Ortu, M., 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 262–271.
Murphy-Hill, E., Zimmermann, T., Nagappan, N., 2014. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 1–11.
Nystrom, R., 2014. Game Programming Patterns. Genever Benning.
Palomba, F., De Lucia, A., Bavota, G., Oliveto, R., 2014. Anti-pattern detection: Methods, challenges, and open issues. In: Advances in Computers, Vol. 95. Elsevier, pp. 201–238.
Pascarella, L., Palomba, F., Di Penta, M., Bacchelli, A., 2018. How is video game development different from software development in open source? In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories. MSR, IEEE, pp. 392–402.
Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering. EASE 12, pp. 1–10.
Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C., 2008. Houston, we have a problem... a survey of actual problems in computer games development. In: Proceedings of the 2008 ACM Symposium on Applied Computing. pp. 707–711.
Politowski, C., Fontoura, L., Petrillo, F., Guéhéneuc, Y.-G., 2016. Are the old days gone? A survey on actual software engineering processes in video game industry. In: Proceedings of the 5th International Workshop on Games and Software Engineering. pp. 22–28.
Politowski, C., Petrillo, F., Ullmann, G.C., de Andrade Werly, J., Guéhéneuc, Y.-G., 2020. Dataset of video game development problems. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 553–557.
Politowski, C., Petrillo, F., Ullmann, G.C., Guéhéneuc, Y.-G., 2021. Game industry problems: An extensive analysis of the gray literature. Inf. Softw. Technol. 106538.
Ray, B., Posnett, D., Filkov, V., Devanbu, P., 2014. A large scale study of programming languages and code quality in github. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp. 155–165.
Santos, R.E., Magalhães, C.V., Capretz, L.F., Correia-Neto, J.S., da Silva, F.Q., Saher, A., 2018. Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–10.
Shanbhag, S., Chimalakonda, S., Sharma, V.S., Kaulgud, V., 2022. Towards a catalog of energy patterns in deep learning development. In: The International Conference on Evaluation and Assessment in Software Engineering 2022. pp. 150–159.
Shi, Y.-R., Shih, J.-L., 2015. Game factors and game-based learning design model. Int. J. Comput. Games Technol. 2015.
Silva, D., Tsantalis, N., Valente, M.T., 2016. Why we refactor? confessions of github contributors. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 858–870.
Tan, M.L., Prasanna, R., Stock, K., Doyle, E.E., Leonard, G., Johnston, D., 2020. Modified usability framework for disaster apps: a qualitative thematic analysis of user reviews. Int. J. Disaster Risk Sci. 11, 615–629.
Teruel, M.A., Navarro, E., González, P., López-Jaquero, V., Montero, F., 2016. Applying thematic analysis to define an awareness interpretation for collaborative computer games. Inf. Softw. Technol. 74, 17–44.
Thielscher, M., 2010. A general game description language for incomplete information games. In: Twenty-Fourth AAAI Conference on Artificial Intelligence.
tong Cai, W., Xavier, P., Turner, S.J., Lee, B.-S., 2002. A scalable architecture for supporting interactive games on the internet. In: Proceedings 16th Workshop on Parallel and Distributed Simulation. IEEE Computer Society, p. 60.
Truelove, A., de Almeida, E.S., Ahmed, I., 2021. We'll fix it in post: what do bug fixes in video game update notes tell us? In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 736–747.
Uddin, G., Sabir, F., Guéhéneuc, Y.-G., Alam, O., Khomh, F., 2021. An empirical study of iot topics in iot developer discussions on stack overflow. Empir. Softw. Eng. 26, 1–45.
Valenzuela-Toledo, P., Bergel, A., 2022. Evolution of GitHub action workflows. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 123–127.
Varvaressos, S., Lavoie, K., Gaboury, S., Hallé, S., 2017. Automated bug finding in video games: A case study for runtime monitoring. Comput. Entertain. (CIE) 15 (1), 1–28.
Wesley, D., Barczak, G., 2016. Innovation and Marketing in the Video Game Industry: Avoiding the Performance Trap. Routledge.
Yan, J.J., Choi, H.-J., 2002. Security issues in online games. Electron. Libr.
Zhang, T., Gao, C., Ma, L., Lyu, M., Kim, M., 2019. An empirical study of common challenges in developing deep learning applications. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 104–115.

Vartika Agrahari is a Machine Learning Engineer in FactSet Systems India. She holds a Masters Degree in Computer Science from Indian Institute of Technology Tirupati. Her research interests include Empirical Software Engineering and Human Computer Interaction.

Shriram Shanbhag is an MS student in the Department of Computer Science & Engineering at Indian Institute of Technology Tirupati, India. He is pursuing his thesis in Energy Efficient Machine learning. His research interests include Empirical Software Research, Green Software Engineering, and Software Engineering for Artificial Intelligence.

Sridhar Chimalakonda is an Assistant Professor in the Department of Computer Science & Engineering at Indian Institute of Technology Tirupati, India. He received his Ph.D. and MS by Research in Computer Science & Engineering from International Institute of Information Technology - Hyderabad, India. He leads the Research in Intelligent Software and Human Analytics (RISHA) Lab which primarily works in the area of Software Engineering, and specifically towards empirically and qualitatively assessing quality, reuse, architecture and evolution of a broad range of software systems (such as mobile, web, games and so on). He is passionate about the massive potential of technology for improving the quality of education and automation in educational technologies.

A Eashaan Rao is a Ph.D. candidate in the Department of Computer Science at the Indian Institute of Technology Tirupati, India. He is a researcher with a strong interest in using AI to enhance software engineering practices, particularly in the area of software maintenance. His research focuses on improving the maintenance cycle of software development by addressing software bugs aiming to contribute to the development of more efficient and effective software engineering practices.