Department of Computer Science,
National University of Computer and Emerging Sciences,
Islamabad

Automated Bugs Identification Through Early Access Game Review (EAGR) Analytics On Game Distribution Platforms

Ali Shahbaz
20I-2020

Dr. Khubaib Amjad Alam

Version 1.0: FinalTerm_Part1_2021
Prepared by: MSRC Computing

Revision History

Date	Version	Description	Supervisor Signatures
12/14/2021	1.0	Thesis Final Term	Dr. Khubaib Amjad Alam
01/15/21	2.0	Thesis Final Term	Dr. Khubaib Amjad Alam

National University
of computer and emerging sciences

Table of Contents

MS Thesis Final (Part-1)

Abstract

User reviews are considered one of the most important source of information about an app and game. The classification and analysis of reviews in order to extract information has proven to be a considerable difficulty. Game applications receive user input in the form of reviews, which can assist developers in selecting games with improved functionality and extracting relevant information such as user feedback, problems, and descriptions of user experiences linked with existing features. Because of the enormous user base and potential benefits of automated feature and bug extraction, game application review analysis has lately arisen as an active topic of research in software engineering. Recently, several research studies have been conducted to mine and categorize user-reviews into actionable software maintenance requests, including feature requests and bug reports.

However, existing literature have mostly focused on mining functional aspects and analysis of use. Several studies have recently been carried out to mine and categorize user evaluations into actionable software maintenance requests, such as feature requests and bug reports. reviews for positive and negative feedback. To achieve user satisfaction and to survive in the app market, addressing these bugs reports to developers is necessary.

Therefore, in this research we formulate this problem as a Multi-label classification problem and propose a bug classification model using CNN (convolutional neural network). Representative features are used to train a model for bug classification. Use of feature extraction methods to train a classification model may lead to divergent results, which implies the need for a careful selection of these methods. Several recent studies have emphasized that basic state-of-the-art taxonomies for bug classifications into different categories. Therefore, this research employs these taxonomies to provide a tools which takes a user reviews dataset and then identify bugs form them and after identification it will classify the bug into their related bug categories using CNN (convolutional neural network). We are taking GAME STEAM ENGINE as a case study to scrap gaming reviews and train model on that reviews for bugs classification.

1. Introduction

The game industry has become one of the most profitable markets in the entertainment industry. Knowing what customers and users think and how they feel about a game is a central piece to drive the decision-making process, of any game developer or game studio, towards the user satisfaction. The steadily increasing popularity of computer games has led to the rise of a multi-billion-dollar industry.

Due to the scale of the computer game industry, developing a successful game is challenging. In addition, prior studies show that gamers are extremely hard to please, making the quality of games an important issue. Most online game stores allow users to review a game that they bought. Such reviews can make or break a game, as other potential buyers often base their purchasing decisions on the reviews of a game.

Studying gamer reviews help developers better understand the concerns and further improve the user-perceived quality. Bugs that persist into releases of video games can have negative impacts on both developers and users, but particular aspects of testing in game development can lead to difficulties in effectively catching these missed bugs. It has become common practice for developers to apply updates to games in order to fix missed bugs.

We are giving a tools which takes a user review dataset and then identify bugs form them and after identification it will classify the bug into their related bug categories. We are taking GAME STEAM ENGINE as a case study to scrap gaming reviews and train model on that reviews for bugs classification.

2. Literature Review

Taxonomies and case studies are presented in the past to get information from application reviews which include user response toward the app, bugs user are facing, the new improvements users want in game.
A recent study by [1] is presented name as PUMA, an auto-mated, phrase-based approach to extract user opinions in app reviews. This approach includes a technique to extract phrases in reviews using part-of-speech (POS) templates; a technique to cluster phrases having similar meanings (each cluster is considered as a major user opinion); and a technique to monitor phrase clusters with negative sentiments for their outbreaks over time. They used PUMA to study two popular apps and found that it can reveal severe problems of those apps reported in their user reviews.

Another study have done for the classification of bugs by [2]. They have examined more than 100 thousand bug reports of 380 projects, they found that bugs can be classified into four types based on the location of their fixes. Type 1 bugs are the ones that fixed by modifying a single location in the code, while Type 2 refers to bugs that are fixed in more than one location. Type 3 refers to multiple bugs that are fixed in the exact same location. Type 4 is an extension of Type 3, where multiple bugs are resolved by modifying the same set of locations. Another probabilistic techniques to classify app reviews was introduced by [3]. They introduce several probabilistic techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and text ratings. For this, they use review metadata such as the star rating and the tense, as well as, text classification, natural language processing, and sentiment analysis techniques. They conducted a series of experiments to compare the accuracy of the techniques and compared them with simple string matching. To detect bad type of smells in unity project a unity linter tool was presented by [4]. Unity Linter, a static analysis tool that supports Unity video game developers to detect seven types of bad smells they have identified as relevant in video game development. Such smell types pertain to performance, maintainability and incorrect behavior problems. Unity Linter is, in general, accurate enough in detecting smells (86%-100% precision and 50%-100% recall), and their study shows that the studied smell types occur in 39%-97% of the analyzed projects.

For detecting bugs in games in real time an deep enforcement learning technique was presented by [5] .Deep enforcement learning (DRL) can be used to increase test coverage, find exploits, test map difficulty, and to detect common problems that arise in the testing of first-person shooter (FPS) games. RL is better suited for modular integration where it can complement rather than replace existing techniques. Not all problems are better solved with RL and training is substantially easier when focusing on single, well isolated tasks. RL can complement scripted tests in edge cases where human-like navigation, exploration, exploit detection and difficulty evaluation is hard to achieve. To check how mobile app reviews are different from gaming reviews a case study was presented by [6]. This study performs an empirical study of the reviews of 6224 games on the Steam platform, one of the most popular digital game delivery platforms, to better understand if game reviews share similar characteristics with mobile app reviews, and thereby understand whether the conclusions and tools from mobile app review studies can be leveraged by game developers.

An automated system level game testing methodology was presented by [7].They provides a detailed modeling methodology to support automated system-level game testing. As part of the methodology, they provide guidelines for modeling the platform games for testing using our proposed game test modeling profile. They have used domain modeling for representing the game structure and UML state machines for behavioral modeling. They presented the details related to automated test case generation, execution, and oracle generation. Demonstrate their model-based testing approach by applying it on two cases studies, a widely referenced and open source implementation of Mario brothers game and an industrial case study of an endless runner game.

To detect how long user spend time to use the app or play the game a case study was presented in [8]. They reported on an exploratory study, which analyzes over one million reviews from the Apple App Store. They investigated how and when users provide feedback, inspected the feedback content, and analyzed its impact on the user community. Found that most of the feedback is provided shortly after new releases, with a quickly decreasing frequency over time. Reviews typically contain multiple topics, such as user experience, bug reports, and feature requests. The quality and constructiveness vary widely, from helpful advices and innovative ideas to insulting offenses. Feedback content has an impact on download numbers: positive messages usually lead to better ratings and vice versa. Negative feedback such as shortcomings is typically destructive and miss’s context details and user experience.

To extract bugs from games videos while playing a research paper was published by [9]/ They have studied the number of gameplay videos on steam platform and YouTube. They proposed an approach which uses a random forest classifier to rank gameplay videos based on their likelihood of being a bug video. Their proposed approach achieves a precision that is 43% higher than that of the naive keyword searching approach on a manually labelled dataset of 96 videos.

To classify the bugs into specific categories based on their location a taxonomy was presented by [10]. They presented a taxonomy of possible failures, divided into temporal and non-temporal failures. The taxonomy can guide the thinking of designers and testers alike, helping them expose bugs in the game. This will lead to games being better tested and designed, with fewer failures when released. This paper has discussed a separation between game design specification and implementation. They have considered other taxonomies too which include Bezier taxonomy. Bainbridge also have previously classified the video games glitches. This study has divided the implementation failure to temporal and non-temporal failures. Which categorized the bugs. This taxonomy improves the effectiveness of pre-release testing, but it creates the vital groundwork to allow validation of exciting new research in video game failure reduction.

To address shortcomings of this prior taxonomy and to attain a deeper understanding of the types of bugs in games, they expanded the taxonomy of bug types.[11] They have analyzed 12,122 bug fixes taken from 723 updates for 30 popular games on the Steam platform. They have further categorized these bug fixes using our taxonomy of bug types. Then analyzed the frequency at which the different bug types appear in the update notes and investigated which types of bugs recur more often over multiple updates. Additionally, they investigated which types of bugs most frequently appear in urgent updates or hotfixes, as the bugs that appear in these updates are more likely to have a severe negative impact on users.

Table 1: Summary of the related work.

Ref. No., Year	Methodology/Approach	Strengths	Weaknesses
[1]	• Data mining based techniques (Mine the review based on phrases) • PUMA, an auto-mated, phrase-based approach to extract user opinions in app reviews.	• User opinions in reviews • Phrases from reviews	• Approach cannot extract opinions corresponding to textual expressions • No classification and categorization of bugs
[2]	• Statistical Method (Pearson's chi-squared test)	• Simplifying bug reports • Classification of bugs into four categories	• Focused on simplifying bug reports • Fewer bug reports mentioned
[5]	• Deep Learning algorithms (Deep enforcement learning DRL's)	• Navigation of FPS type games, • Game exploits and bugs, distribution of visited states, and difficulty evaluation	• Limited to FPS games • Not all problems can be solved by RL(Reinforcement learning) technique including bugs classification and categorization
[3]	• Deep Learning algorithms (sentiment analysis techniques)	• Classification of app reviews into four types Bug reports, feature requests, user experiences, and text ratings	• Limited to bug reports, feature only no method for bug classification
[4]	• Static method unity linter tool	• Unity games and projects	• Static analysis is not particularly used to finding bugs in games • No classification of bugs
[6]	• Statistical Method (Wilcoxon signed rank test)	• Similarity check between game reviews characteristics with mobile app reviews characteristics • Checking the information which can be extracted from the reviews	• Only collected one month of reviews that have an accurate number of playing hours
[7]	• Unity test tool	• Testing of mobile game using unity	• Only tested for application which is developed on unity engine

		tool	• No any report and identification of bugs
[12]	• Game loop video	• XML written template which generates a message based on game's current state	• GUI need to be reprogrammed for every game • Detection of bugs in real time but no explanation for its classification
[8]	• Supervised Machine Learning algorithms (Model based testing)	• Model based testing to model the conceptual and behavioral details to be tested	• Focused only on functional testing of the game • Developers have to develop separate test ready models
[13]	• Statistical Method (Descriptive statistical method)	• User reviews to understand user requirements from store • Aggregation and use of feedback	• No identification of bugs from user reviews
[9]	• Supervised Machine Learning algorithms (random forest classifier)	• Game play videos from steam platform and YouTube, used videos meta data and compare it with keywords to identify bug videos	• Limited to videos data • Not able to detect bugs from textual data
[14]	• Stratify app model	• Grouping of reviews based on user concerns • User general sentiment on each entity • User review classification	• No classification of bugs
[10]	• Divide-and-conquer	• Possible failures in games and division in temporal and non-temporal failures, categorization of bugs	• No tool to automate the process • Taxonomy to just classify the bugs into different categories
[11]	• Statistical Method Tool (cosine similarity measure)	• Bug fixes taken from app updates, • Frequency of bugs in games	• Manually evaluated these fixes to find the true recurring bugs

2.1 Research Gap

Reviews contain technically useful information that can be further utilized to understand the user's requirements and needs. Recent analysis has revealed that reviews are helpful for developers to resolve issues and identify requirements to meet the end user's expectations. The manual analysis is impractical for a large number of reviews, received every day. Therefore, research on app-review-mining has followed the Text Mining approaches to extract valuable information from the reviews. This approaches able to detect feature requests, user responses and bug reports. So studies have also able to find the non-functional requirements from the reviews. But there is a need of tool to which can take user reviews as an input then after training and modeling the data it can generate the classification bug report.

2.2 Problem Statement

Several research studies have been conducted to mine and categorize user-reviews into actionable software maintenance requests, including feature requests and bug reports. Existing literature has focused on user response toward games and detection of bugs in reviews. But there is need of a tool which can mine the reviews and extract bugs from it and then classify them in to bugs categories.

2.3 Research Objectives/Research Questions

- **Research Objective:** To provide a model which can classify and then categorize the bugs in to their types from games reviews

RQ1: What is the traditional bug classification and identification process?

RQ4: What kind of techniques has been proposed in past for classification and identification of bugs from gaming reviews?

RQ2: What evaluation/classification/identification metrics have been used?

RQ3: What are gamers talking about in gaming reviews?

RQ5: What is the overall research productivity in this domain?

3. Proposed Research Methodology:

We are using steam game platform as a case study to collect reviews of games using a custom crawler. As reviews are in natural language so we are applying NLP techniques (Tokenization, Tokenization, stop words removal, stemming, lemmatization) to extract valuable information (Negative feedback, positive feedback) from reviews. Then we are training a standard deep learning model for text classification and sentiment analysis uses a word embedding layer and one-dimensional convolutional neural network. The model can be expanded by using multiple parallel convolutional neural networks that read the source document of different sizes. This, in effect, creates a multichannel convolutional neural network for text that reads text with different n-gram sizes (groups of words). Then with custom bug categories dictionary we are training the model to classify the bug in the related categories.

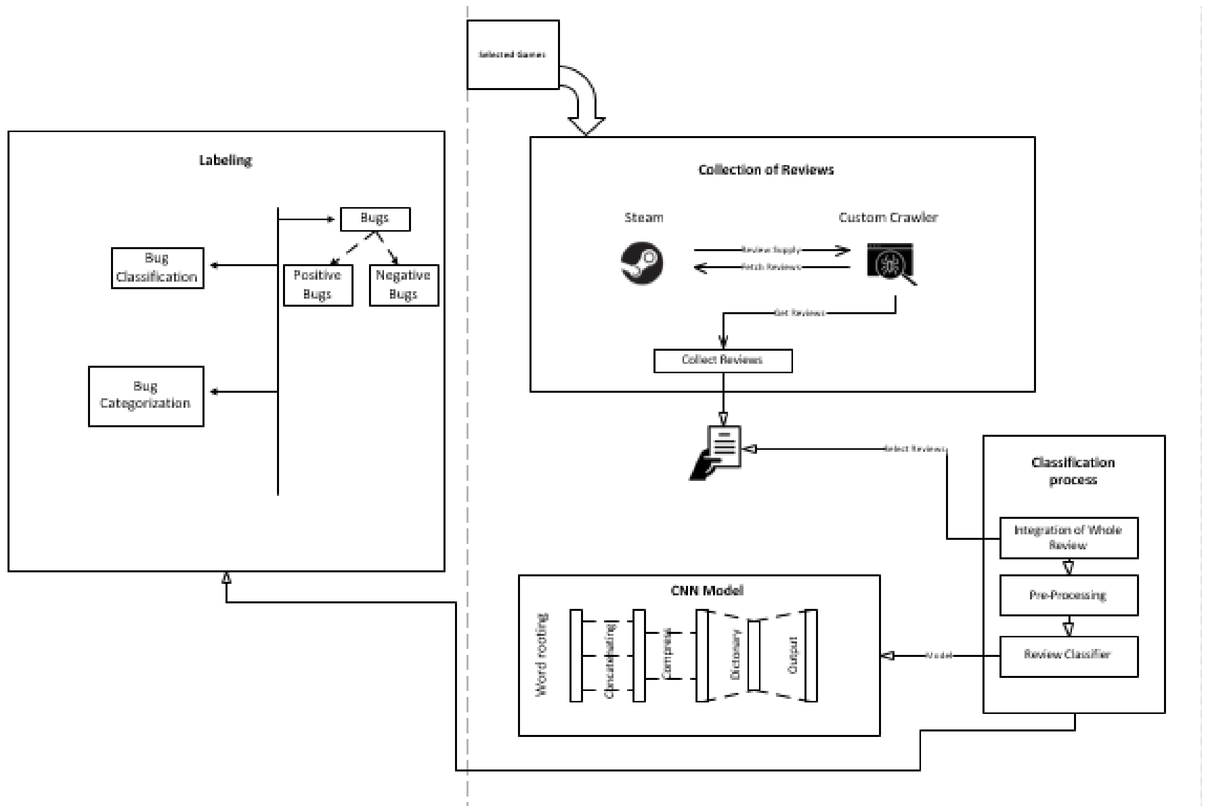


Figure 1: System architecture high level diagram

3.1 Algorithm and Implementation Details

We are considering STEAM ENGINE game reviews as a dataset for our CNN model to classify the bug in the respected categories.

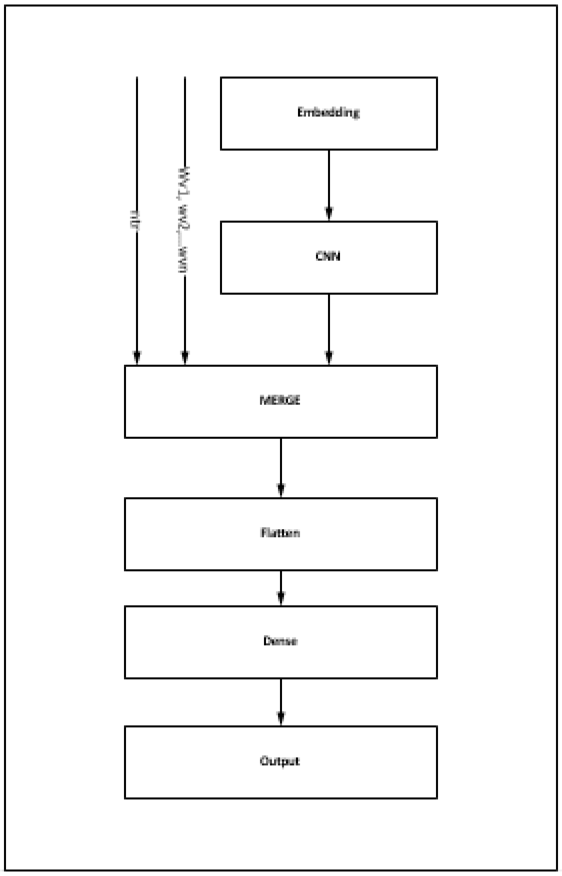
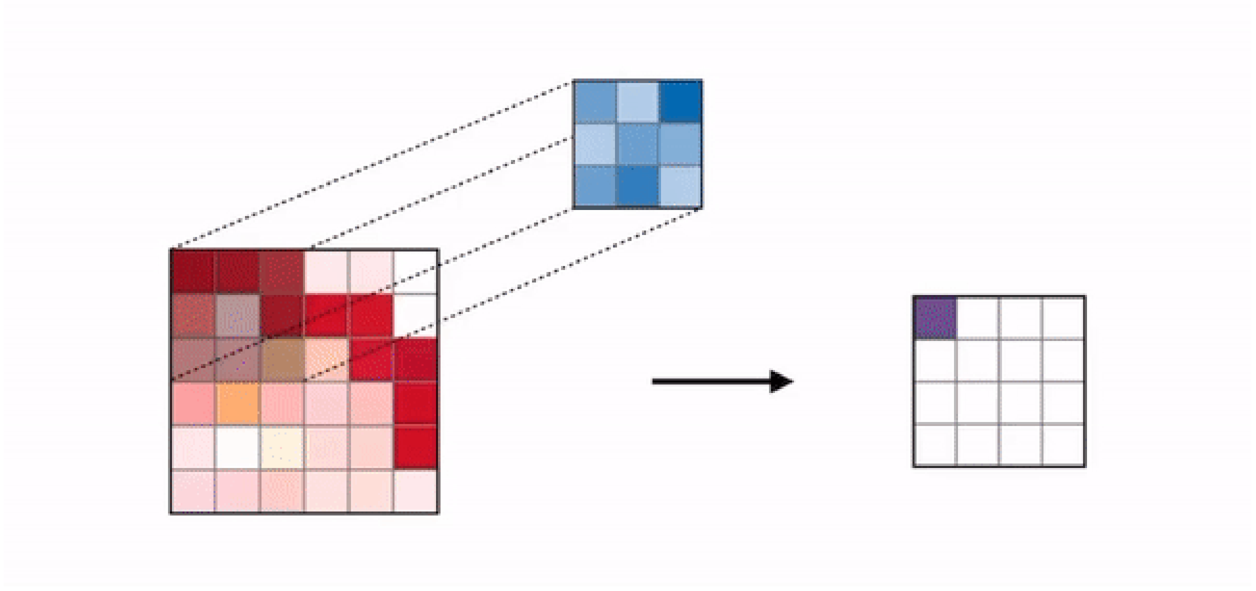


Figure 1.2 CNN Model Working

3.2 CNN (Convolution neural network)

- **Convolutional layers**
  The convolution layer uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyper parameters include filter size F and padding P. We will be using **Rectified linear unit (ReLU) for activation function**. It aims at introducing non-linearity's to the network (g(z) = max(0,z)).



Filter hyper parameters:

The convolution layer contains filters; in which we will apply the following parameters.

1. **Dimensions of a filter:** We will be applying filter of size 3 * 3 to an input, which will produce the feature map.
2. **Padding:** We will be applying full padding, in which filter see the input end to end. It can be achieved by adding 0 to each side of the boundaries of the input.

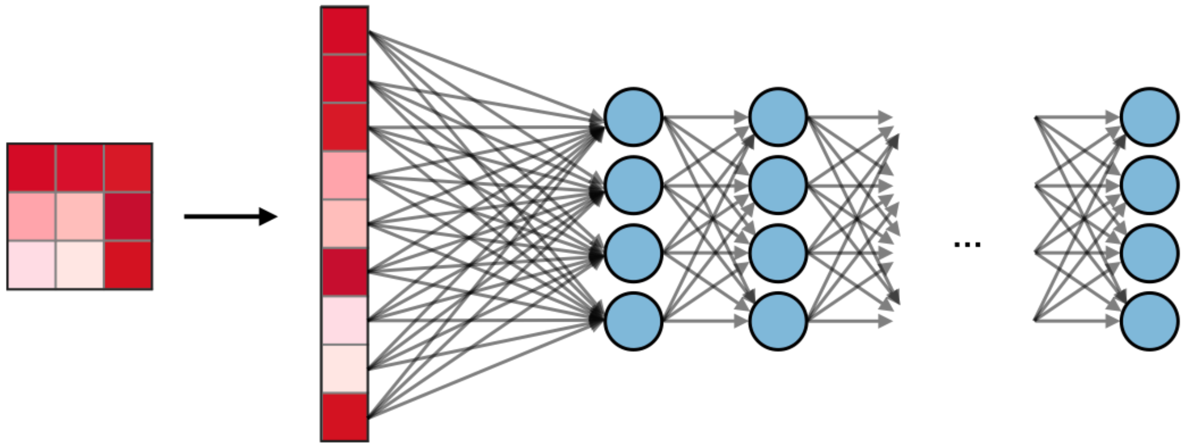
- **Pooling Layers**

The pooling layer is a down sampling operation, typically applied after a convolution layer, which does some spatial invariance.

Type	Purpose
Max Pooling	Each pooling operation selects the maximum value of the current view.

- **Fully Connected Layer**

  The fully connected layer operates on a flattened input where each input is connected to all neurons.



4. References

[1] P. M. Vu, H. V Pham, T. T. Nguyen, and T. T. Nguyen, "Phrase-based extraction of user opinions in mobile app reviews," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 726–731.

[2] M. Nayrolles and A. Hamou-Lhadj, "Towards a classification of bugs to facilitate software maintainability tasks," *Proc. - Int. Conf. Softw. Eng.*, pp. 25–32, 2018, doi: 10.1145/3194095.3194101.

[3] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requir. Eng.*, vol. 21, no. 3, pp. 311–331, 2016, doi: 10.1007/s00766-016-0251-9.

[4] A. Borrelli, V. Nardone, G. A. Di Lucca, G. Canfora, and M. Di Penta, "Detecting Video Game-Specific Bad Smells in Unity Projects," *Proc. - 2020 IEEE/ACM 17th Int. Conf. Min. Softw. Repos. MSR 2020*, pp. 198–208, 2020, doi: 10.1145/3379597.3387454.

[5] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting Automated Game Testing with Deep Reinforcement Learning," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 600–603, doi: 10.1109/CoG47356.2020.9231552.

[6] D. Lin, C. P. Bezemer, Y. Zou, and A. E. Hassan, *An empirical study of game reviews on the Steam platform*, vol. 24, no. 1. Empirical Software Engineering, 2019.

[7] A. C. Barus, R. Deddy Hasiholan Tobing, D. N. Pratiwi, S. A. Damanik, and J. Pasaribu, "Mobile game testing: Case study of a puzzle game genre," *Proc. 2015 Int. Conf. Autom. Cogn. Sci. Opt. Micro Electro-Mechanical Syst. Inf. Technol. ICACOMIT 2015*, pp. 145–149, 2016, doi: 10.1109/ICACOMIT.2015.7440194.

[8] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 426–435, doi: 10.1109/MODELS.2015.7338274.

[9] D. Lin, C. P. Bezemer, and A. E. Hassan, "Identifying gameplay videos that exhibit bugs in computer games," *Empir. Softw. Eng.*, vol. 24, no. 6, pp. 4006–4033, 2019, doi: 10.1007/s10664-019-09733-6.

[10] C. Lewis, J. Whitehead, and N. Wardrip-Fruin, "What went wrong: A taxonomy of video game bugs," *FDG 2010 - Proc. 5th Int. Conf. Found. Digit. Games*, pp. 108–115, 2010, doi: 10.1145/1822348.1822363.

[11] A. Truelove, E. S. de Almeida, and I. Ahmed, "We'll Fix It in Post: What Do Bug Fixes in Video Game Update Notes Tell Us?," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 736–747, doi: 10.1109/ICSE43902.2021.00073.

[12] S. Varvaressos, K. Lavoie, A. B. Massé, S. Gaboury, and S. Hallé, "Automated Bug Finding in Video Games: A Case Study for Runtime Monitoring," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 2014, pp. 143–152, doi: 10.1109/ICST.2014.27.

[13] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 125–134, doi: 10.1109/RE.2013.6636712.

[14] Y. Liu, Y. Li, Y. Guo, and M. Zhang, "Stratify Mobile App Reviews: E-LDA Model Based on Hot 'Entity' Discovery," in *2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, 2016, pp. 581–588, doi: 10.1109/SITIS.2016.97.