OSSGameBench: A Large-Scale Dataset of Development Activities in Open-Source Video Games

Faiz Marsad
University of Calgary
Calgary, Canada
faiz.marsad@ucalgary.ca

Nimmi Weeraddana
University of Calgary
Calgary, Canada
nimmi.weeraddana@ucalgary.ca

Abstract

Video games are a distinct type of software that engage users through deeply immersive and interactive experiences. Yet, unlike other types of software where standardized benchmark datasets have accelerated research on downstream tasks such as defect localization and repair, the game development domain lacks comparable resources that reflect its distinctive characteristics. For instance, defects in games often arise from the complex interplay between source code (e.g., C++ files) and non-code assets (e.g., graphic files), whereas in other software, such complications are less prevalent. Therefore, specialized datasets are required to capture the multi-disciplinary demands unique to game development. Unavailability of such specialized datasets limits the analysis, development, and evaluation of data-driven solutions tailored to game development.

To fill this void, we introduce OSSGameBench, a curated benchmark dataset derived from 950 open-source repositories in GitHub, encompassing both playable video games and essential game development tools (e.g., game development frameworks). The dataset contains issues, pull requests, comments, and commits, with explicit mappings among these entities to enable future research aimed at improving the full spectrum of game development.

CCS Concepts

• **Software and its engineering → Software development methods**; **Software libraries and repositories**.

Keywords

open source, video games, issue reports, pull requests, commits

ACM Reference Format:
Faiz Marsad and Nimmi Weeraddana. 2026. OSSGameBench: A Large-Scale Dataset of Development Activities in Open-Source Video Games. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3793302.3793313

1 Introduction

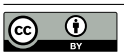
Today's software landscape is evolving at an unprecedented pace. Although most software applications focus on functionality, a subset has evolved into video games, where the focus extends beyond entertainment [2] to education [22], training [13], and well-being [4], offering users a deep sense of presence and engagement in multi-sensory environments. Video games have been shown to provide several benefits [2]. For example, playing video games has been shown to enhance a wide range of cognitive skills, including concentration [3], problem-solving [12], and creativity [21].

Despite the many benefits of video games, developing them has never been easy for development teams [11]. Although teams rigorously enforce quality assurance practices, defects often manifest as crashes [18], performance bottlenecks, or gameplay inconsistencies that directly impact the player experience. For example, the game *Borderlands 4* was launched to negative reviews due to low frame rates and game window crashes, prompting its developers to publicly advise players to seek refunds.[1] Similarly, *Cyberpunk 2077* was released with critical defects despite years of development, with players reporting crashes and missing sound effects.[2]

Compared to other types of software, game development presents unique challenges: (1) games are expected to be highly interactive, providing an immersive experience for users unlike most other software; and (2) games rely on cross-disciplinary expertise, where developers are responsible for the source code and creative professionals (such as artists) manage non-code artistic assets, unlike traditional software development, which depends primarily on source code. Therefore, ensuring quality in games demands coordinated efforts across both technical and creative disciplines, where defects can originate not only from code but also from gameplay logic.

Although there are datasets for mining software repositories and evaluating tools to improve software development in general, game development lacks comparable resources. This gap introduces two main limitations. First, it limits the development and evaluation of customized solutions for game development, such as defect localization, automated defect repair, and reviewer recommendation. Second, it prevents effective training and assessment of GenAI-based developer tools, which require domain-specific datasets.

To address these limitations, we introduce OSSGameBench—a large-scale dataset of issue-tracker data extracted from 950 open-source game repositories. It contains 319,709 GitHub issues, 846,911 issue comments, 374,538 pull requests (PRs), 1,114,748 PR comments, and 2,758,344 commits (i.e., change sets of source code and/or assets). This dataset supports diverse use cases, including: (a) empirical studies of defect reporting and resolution in open-source games; (b) examining cross-disciplinary collaboration between developers and artists; (c) studying the evolution of game assets (e.g., graphics) and corresponding code changes; (d) training and evaluating GenAI tools for defect repairing, and asset validation in games; and (e) building predictive models for review turnaround time.



MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793313

[1] https://www.reddit.com/r/Games/comments/1nej6w7/borderlands_4_launches
[2] https://forums.cdprojektred.com/index.php?threads/problems

Data availability. Our dataset (OSSGameBench) and the replication package are publicly available.[3]

2 Related Work

Politowski et al. [11] compiled a dataset of 1,035 software engineering problems, extracted from 200 video game postmortems (i.e., developers' experiences publicly exposed right after a game is launched). While this dataset contains software engineering challenges, it does not include corresponding change sets, making it difficult to correlate problems with specific defects or fixes.

Li et al. [8] introduced GBGallery, a curated dataset of 76 defects from five video games to support automated game testing research. While this dataset provides a benchmark for game-specific defect detection techniques, it focuses on runtime defect manifestations (e.g., crashes or gameplay glitches) and lacks links to source code fixes and issue-tracker data, such as code review discussions.

Yapağcı et al. [20] presented BugCraft-Bench, a specialized dataset of user-submitted crash reports from the Minecraft game. Note that crashes are a special type of defect where the game window is forced to terminate unexpectedly. This dataset of crash reports was created to evaluate automated bug reproduction frameworks and contains detailed crash reports from the Minecraft bug tracker. While it serves as a test bench for AI agents to reproduce game crashes based on the reports, it is focused on a single AAA game (*Minecraft*) and on one type of defect (i.e., crashes). The dataset also does not include information such as which code commits fixed each crash or any code review comments on those fixes, highlighting a gap in traceability to development artifacts.

Taesiri et al. [16] introduced GlitchBench, a benchmark dataset that provides visual glitch instances and associated textual descriptions and discussions. GlitchBench covers 205 games and 593 glitch instances (with a mix of real and synthetic glitch samples), offering a unique resource for evaluating multiple models on anomaly detection in video game content. However, it falls short of connecting glitches to code or non-code asset fixes, defect tracking systems, or review processes, and therefore cannot be used to study how glitches are fixed or managed in game development settings.

In this paper, we address these limitations by providing a comprehensive dataset that captures multiple aspects of the development lifecycle in game projects. Unlike prior datasets that either focus on qualitative postmortem analyses [11], defect samples lacking traceability to source code [8], crash-specific reports from a single game [20], or purely visual glitch benchmarks without developer artifacts [16], our dataset includes different types of issue reports (e.g., defect reports and feature enhancements), metadata, code changes, PRs, and review comments, enabling end-to-end traceability from issue reporting through PRs to resolution.

3 Dataset Creation Methodology

This section describes our dataset collection process. Figure 1 shows the study design: Project Selection (PS) and Data Retrieval (DR).

(PS) Project Selection

Our study aims to collect data from open-source video game projects to support a variety of downstream research directions (e.g., empirical studies and GenAI-based tools). Therefore, we need to collect

[3]https://zenodo.org/records/18359123

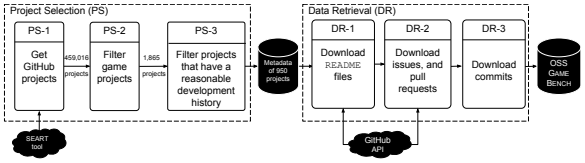


Figure 1: Overview of the dataset creation.

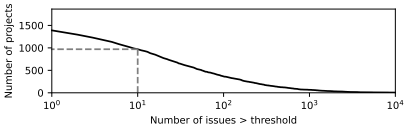


Figure 2: Threshold plot for the number of issues.

a dataset of video game projects that have accrued a rich development history. Below, we describe the steps that we follow to select a representative sample of projects.

(PS-1) Get a list of GitHub projects. We use the Seart tool [1] to query for projects on GitHub that meet the basic inclusion criteria of having more than ten commits [17]. This resulted in 459,016 projects, along with metadata such as the number of issues, programming languages, size (in KB), and topics (if any are declared).[4]

(PS-2) Filter game-related projects. We retain the projects in which the declared topics contain the exact term "game." This filtering step results in 1,865 game-related GitHub projects.

(PS-3) Filter projects that have a reasonable development history. Inspired by prior work [9], we purposely do not restrict our dataset to the most active projects (i.e., those with extensive commit histories), as many less frequently updated projects still play a critical role in the open-source ecosystem. However, we observe that our dataset includes 325 projects with no GitHub issues and 149 projects with only a single issue. To ensure that there is sufficient development activity, we set a minimum threshold for the number of GitHub issues per project. Figure 2 plots candidate threshold values (i.e., number of issues) against the number of surviving projects. We select a threshold of ten issues, which visually corresponds to the "knee" of the curve, where the rate of project reduction begins to stabilize [6, 19]. Although we experimented with the Kneedle algorithm [14], a popular method for detecting inflection points, it identifies 399 projects with more than 87 issues, which is an unnecessarily small subset of projects. Thus, we retain the threshold of ten issues to balance inclusiveness and development activity, yielding 970 projects. Then, we eliminate duplicate projects, resulting in 950 projects for further analysis.

(DR) Data Retrieval

After obtaining a set of projects, we process each project further to obtain data about `README` files, issue reports, PRs, and commits. Below, we describe the steps of this process in detail. Figure 3 shows the schema of the retrieved dataset.

(DR-1) Download `README` file content. After selecting the final 950 projects, we collect the `README.md` file of each project using GitHub REST API.[5] Note that we find 26 projects that do not contain

[4]https://github.com/topics

[5]https://docs.github.com/en/rest

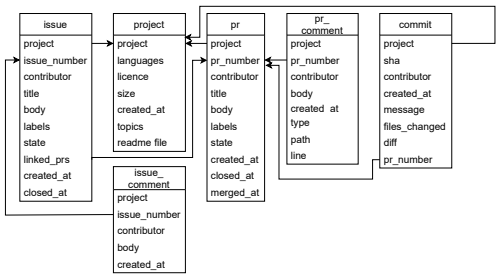


Figure 3: Dataset schema of OSSGameBench.

a README file in their root directory. Although the remaining 924 README files are unstructured, they contain rich textual descriptions of gameplay patterns, game genres, and supported platforms.

(DR-2) Download issue reports and PRs. For each project, we collect issue and PR data using GitHub's GraphQL API.[6] For issues, we retrieve both open and closed reports with structured fields such as title, body (i.e., description), contributor (the creator of the issue), creation and closure timestamps, and labels. Note that labels classify issues into categories such as defects or feature requests. We also collect comments, which provide detailed context for issues. In Figure 3, the `issue` and `issue_comment` entities reflect this data. Although the Seart dataset indicates that each selected project contains more than ten issues, we find six projects (e.g., SceneGate[7]) that do not expose their issue reports via the GitHub API.[6]

We apply the same process to collect PRs, retrieving contributor details, title, body (i.e., description), state (open, closed, or merged), and associated timestamps. We also collect comments linked to each PR, which contain code-review decisions, rationale for changes, debugging arguments, and explanations of patch correctness. This data is reflected as `pr` and `pr_comment` entities in Figure 3.

To establish traceability between issue reports and PRs, we record explicit links between them. We record these links through the `closingIssuesReferences` field of a given issue returned by the GitHub API, which provides references to the PR that formally closes the issue. We further identify more links where contributors informally reference issues in their PR titles. For example, the body text of the PR #78 in project alexbatalov/arcanum-ce,[8] states "Closes #69" and "Closes #70," indicating that the PR closes the issues with issue numbers #69 and #70. To find such cases, we detect patterns such as "fixes," "closes," "resolves," followed by an integer (i.e., an issue number) in both titles and bodies of PRs. We randomly checked a sample of 60 issues and did not find any discrepancies.

Then, we establish explicit traceability between PRs and commits by retrieving the list of commit identifiers (i.e., commit hashes) that are linked to each PR using the GitHub API.[5]

(DR-3) Download commit data. To enable end-to-end traceability from PRs and commit hashes to their corresponding source and asset changes, we retrieve the complete commit history of all 950 projects. For each project, we clone the repository and extract commits using the `git-log` command. The schema is shown as the `commit` entity in Figure 3. For each commit, we record metadata such as the commit hash (`sha`), contributor, timestamp, message, and details such as line-level code changes (i.e., `diff`). This step results in 2,758,344 commits across the 950 GitHub projects. Although the Seart dataset indicates that each selected project contains more than ten commits, we find two projects (e.g., kartoffels[9]) that have less than ten commits that are accessible via the GitHub API.[6]

Together with issues, PRs, and comments, our OSSGameBench dataset provides an end-to-end trace linking issue reports, PRs, and commits in open-source video game-related projects. Table 1 presents the descriptive statistics of OSSGameBench.

Table 1: Descriptive statistics of OSSGameBench.

Metric	Min	1st Quant.	Median	3rd Quant.	Max	Total
Issues	0[10]	25	61	204	12,483	319,709
Issue comments	0	35	96	386	35,065	846,911
PRs	0	15	59	254	14,315	374,538
PR comments	0	10	55	295	99,601	1,114,748
Commits	1[10]	322	791	2,340	75,707	2,758,344

4 Dataset Schema

We now delve into the entities in OSSGameBench (Figure 3): `project`, `issue`, `issue_comment`, `pr`, `pr_comment`, and `commit`.

The `project` entity contains metadata that is collected in Section 3 (PS-3) and (DR-1). For each project, the entity contains metadata such as programming languages used, license type, size, and content from their README files. The `issue` entity contains details of issue reports and is linked to the `project` entity via its `project` attribute. Other attributes include the unique issue identifier, contributor, title, body, labels, and references for any PRs that are linked to an issue (`linked_prs`). The `issue_comment` entity stores the commenter (i.e., `contributor` attribute), body of the comment, timestamp, and linked issue report.

PR data is stored as records in the `pr` entity. Each record includes information such as the unique PR identifier (`number`), contributor, title, body, labels, timestamps, and the associated name of the project. A PR may have discussion comments as well as review comments that provide line-by-line feedback on the change set. The details of these comments are stored in the `pr_comment` entity and are distinguished by a `type` attribute. Lastly, each commit's details—such as the commit hash (`sha`) and change set information—are recorded in the `commit` entity, which links to the `project` and `pr` entities via the `project` and `pr_number` attributes, respectively.

5 Dataset Utility and Coverage

The OSSGameBench dataset can support research on open-source game development. Below, we demonstrate its utility and coverage.

Analysis 1: Programming languages. To identify the dominant programming language used in each project, we parse the language metadata associated with each repository from the `project` entity in OSSGameBench, and obtain the language that accounts for the highest number of source files. Then, we plot the distribution of the top-ten dominant programming languages across all projects, as shown in Figure 4. While this dataset contains projects

[6] https://docs.github.com/en/graphql
[7] https://github.com/SceneGate/SceneGate
[8] https://github.com/alexbatalov/arcanum-ce/pull/78
[9] https://github.com/Patryk27/kartoffels
[10] There are six projects with zero issue reports in our dataset; see DR-2 for more details. Also, there are two projects that contain fewer than ten commits (DR-3).

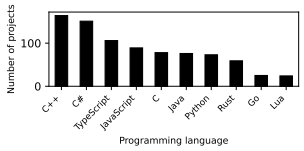


Figure 4: Top ten programming languages across projects.

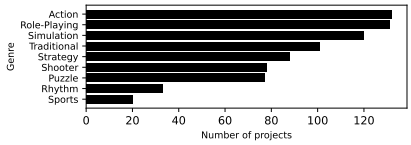


Figure 5: Distribution of game genres.

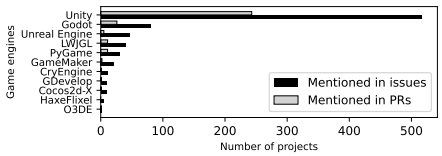


Figure 6: Projects that mention popular game engines in their issue reports and PRs.

implemented in a variety of programming languages, the figure shows that most projects primarily use `C++`, followed by `C#`.

Analysis 2: Game genres. A video game can belong to one or more genres, such as *action*, *role-playing*, *simulation*, *sports*, and *shooter* [15].[11] While the genre of a game is not explicitly available in the metadata of projects (i.e., in the `project` entity), the content of the `README` file of a project may contain such information. Therefore, we perform a *directed content analysis* [5] to classify `README` files in our dataset into game genres based on their textual content. A directed content analysis is a qualitative method in which predefined categories guide the coding of textual artifacts [5]. In particular, two coders—one human (the second author) and one LLM (*OpenAI GPT-4.1*)— classify `README` files of 100 randomly sampled projects in our dataset. Our Online Appendix[3] details the coding procedure and LLM prompts. We then compute the inter-coder agreement using Krippendorff's α [7]; it yields an average α value of 0.68 across all predefined genres, indicating moderate-to-substantial agreement between the coders. We then extend our LLM script to classify all remaining `README` files.

Figure 5 visualizes the distribution of genres in our dataset. Accordingly, the most frequent genres are *action*, *role-playing*, and *simulation*. A close inspection of this data reveals that only 614 projects contain genre information in their `README` files, 66% of which are repositories that serve tooling or infrastructural purposes rather than representing games themselves. For example, FXGL[12] is a *game engine* for `Java` and `Kotlin` projects. A game engine is a software framework primarily designed for video game development. Likewise, XStreaming[13] is a streaming client that enables remote gameplay on mobile devices. This variety highlights our dataset's broad coverage, capturing playable games as well as tools that represent the open-source game development landscape.

[11] https://github.com/uwgamergroup/vocabulary-gameplay-genre/
[12] https://github.com/AlmasB/FXGL
[13] https://github.com/Geocld/XStreaming

Analysis 3: Game engines. A game may be built by leveraging an existing engine (e.g., *Unreal Engine*, *Unity*, or *Godot*), or it may have its own engine. In this analysis, we perform keyword matching to identify the projects that discuss the popular video game engines [10][14,15] in issue reports, comments, PRs, and PR comments. We then count the number of projects that contain issues and PRs mentioning these engines. Figure 6 shows that the *Unity* engine dominates discussions in our dataset, while it also includes a substantial amount of discussions about other game engines.

Analysis 4: Types of issue reports, PRs, and commits. An issue report or a PR may include one or more labels as metadata. By analyzing the labels of issue reports, we find that defect reports are by far the most common issue type, accounting for 58,900 issue reports, followed by enhancement requests (22,517 issues). Our dataset also includes UI-related issues (4,088) that can provide insights into cross-disciplinary collaboration in game development. Similarly, we observe distinct labelling patterns in PRs, with frequent types such as dependency fixes (19,993) and enhancements (9,331). This variety in PR types would support studies on automated program repair by capturing diverse code modification intents.

Lastly, a close inspection of commits reveals that our dataset exhibits a rich representation of non-code assets (e.g., graphical, audio, and engine-specific assets), highlighting the inherently cross-disciplinary nature of open-source game development. For graphical assets, the dataset includes 109,122 commits that modify PNG files, 15,500 that modify SVG files, and 7,265 that modify JPG files. Then, for audio assets, the dataset contains 12,446 commits modifying OGG files, 3,720 modifying WAV files, and 3,506 modifying MP3 files, reflecting teams' engagement with non-code assets.

6 Limitations

While OSSGameBench offers a robust basis for examining challenges in video game development and informing solution design, it is limited to publicly available open-source repositories. Thus, some findings derived from this dataset may not fully generalize to industrial settings. Nevertheless, as shown in Section 5, the included projects span diverse game engines and programming languages, offering insights into real-world game development without legal or confidentiality barriers. Lastly, our data extraction relies on the GitHub API, which does not always preserve traceability between issues, PRs, and commits. For example, if an issue is resolved across multiple PRs without explicit linking, the mapping between the issue and its corresponding PRs may be incomplete.

7 Conclusion

In this paper, we introduced OSSGameBench—a large-scale dataset capturing the development history of 950 open-source video game projects, including 319,709 issue reports, 374,538 PRs, and 2,758,344 commits, offering a statistically representative foundation for empirical research. We hope this dataset will spark new research on video games, empowering development teams and researchers alike to build smarter and more creative tools, enabling improvements in defect management, developer–artist collaboration, and evaluation of GenAI-driven solutions to support game development teams.

[14] https://www.perforce.com/blog/vcs/most-popular-game-engines
[15] https://learn.g2.com/best-game-engine

References

[1] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling projects in github for MSR studies. In *18th International Conference on Mining Software Repositories (MSR)*. doi:10.1109/MSR52588.2021.00074

[2] Isabela Granic, Adam Lobel, and Rutger CME Engels. 2014. The benefits of playing video games. *American psychologist* 69, 1 (2014), 66.

[3] C Shawn Green and Daphne Bavelier. 2012. Learning, attentional control, and action video games. *Current biology* 22, 6 (2012), R197–R206.

[4] Yemaya J Halbrook, Aisling T O'Donnell, and Rachel M Msetfi. 2019. When and how video games can be good: A review of the positive effects of video games on well-being. *Perspectives on Psychological Science* 14, 6 (2019), 1096–1104.

[5] Áine M Humble. 2009. Technique triangulation for validation in directed content analysis. *International journal of qualitative methods* 8, 3 (2009), 34–51.

[6] Ajiromola Kola-Olawuyi, Nimmi Rashinika Weeraddana, and Meiyappan Nagappan. 2024. The impact of code ownership of devops artefacts on the outcome of devops ci builds. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 543–555.

[7] Klaus Krippendorff. 2011. Computing Krippendorff's alpha-reliability. (2011).

[8] Zhuo Li, Yuechen Wu, Lei Ma, Xiaofei Xie, Yingfeng Chen, and Changjie Fan. 2022. GBGallery: A benchmark and framework for game testing. *Empirical Software Engineering* 27, 6 (2022), 140.

[9] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *32nd international conference on automated software engineering*.

[10] Savaş Öztürk. 2025. Analyzing maintainability factors in open-source game engines: implications for game developers. *Multimedia Tools and Applications* (2025), 1–29.

[11] Cristiano Politowski, Fabio Petrillo, Gabriel Cavalheiro Ullmann, Josias de Andrade Werly, and Yann-Gaël Guéhéneuc. 2020. Dataset of video game development problems. In *Proceedings of the 17th international conference on mining software repositories*. 553–557.

[12] Marc R Prensky. 2012. *From digital natives to digital wisdom: Hopeful essays for 21st century learning*. Corwin Press.

[13] James C Rosser, Paul J Lynch, Laurie Cuddihy, Douglas A Gentile, Jonathan Klonsky, and Ronald Merrell. 2007. The impact of video games on training surgeons in the 21st century. *Archives of surgery* 142, 2 (2007), 181–186.

[14] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. 2011. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *Proceedings of the 31st international conference on distributed computing systems workshops*.

[15] Jakub Swacha. 2025. The Relative Popularity of Video Game Genres in the Scientific Literature: A Bibliographic Survey. *Multimodal Technologies and Interaction* 9, 6 (2025), 59.

[16] Mohammad Reza Taesiri, Tianjun Feng, Cor-Paul Bezemer, and Anh Nguyen. 2024. GlitchBench: Can large multimodal models detect video game glitches?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22444–22455.

[17] Nimmi Rashinika Weeraddana, Mahmoud Alfadel, and Shane McIntosh. 2024. Dependency-induced waste in continuous integration: An empirical study of unused dependencies in the npm ecosystem. *Proceedings of the ACM on Software Engineering* 1, FSE (2024).

[18] Nimmi Rashinika Weeraddana, Sarra Habchi, and Shane McIntosh. 2025. Crash Report Prioritization for Large-Scale Scheduled Launches. In *Proc. of the International Conference on Software Engineering (ICSE)*.

[19] Nimmi Rashinika Weeraddana, Xiaoyan Xu, Mahmoud Alfadel, Shane McIntosh, and Meiyappan Nagappan. 2023. An empirical comparison of ethnic and gender diversity of DevOps and non-DevOps contributions to open-source projects. *Empirical Software Engineering* 28, 6 (2023), 150.

[20] Eray Yapağcı, Yavuz Alp Sencer Öztürk, and Eray Tüzün. 2025. BugCraft: End-to-End Crash Bug Reproduction Using LLM Agents in Minecraft. *arXiv preprint arXiv:2503.20036* (2025).

[21] Chloe Shu-Hua Yeh. 2015. Exploring the effects of videogame play on creativity performance and emotional responses. *Computers in Human Behavior* 53 (2015), 396–407.

[22] Richard Zhao, Faisal Aqlan, Lisa Jo Elliott, and Heather C Lum. 2019. Developing a virtual reality game for manufacturing education. In *Proceedings of the 14th international conference on the foundations of digital games*. 1–4.