An Experiment on Content Generation of Game Software Engineering

Anonymous Author(s)

ABSTRACT

Background Video games are complex projects that involve a seamless integration of art and software during the development process to compound the final product. In the creation of a video game, software is fundamental as it governs the behavior and attributes that shape the player's experience within the game. When assessing the quality of a video game, one needs to consider specific quality aspects, namely 'design', 'difficulty', 'fun', and 'immersibility', which are not considered for traditional software. On the other hand, there are not well- established best practice for the empirical assessment of video game as instead there are for the empirical evaluation of more traditional software. **Aims** Our goal is to carry out a rigorous empirical evaluation of the latest proposals to automatically generate content for videogames following best practise established for traditional software. Specifically, we compare Procedural Content Generation (PCG) and Reuse-based Content Generation (RCG). Our study also considers the perception of players and professional developers on the content generation. **Method** We conducted a controlled experiment where human-subjects had to play with and evaluate content automatically generated for a commercial video-game by the two techniques (PCG and RCG) based on specific quality aspects of video games. 44 subjects including professional developers and players participated in our experiment. **Results** The results suggest that RCG generates content of higher quality than PCG which is more aligned with the pre-existent content. **Conclusions** The results can turn the tides for content generation. RCG has been underexplored so far because the reuse factor of RCG is perceived as repetition by the developers, who ultimately want to avoid repetition in their video games as much as possible. However, our study revealed that using RCG unlocks latent content that is actually favoured by players and developers.

KEYWORDS

Empirical Study, Automated Software Transplantation, Procedural Content Generation

ACM Reference Format:

Anonymous Author(s). 2024. An Experiment on Content Generation of Game Software Engineering. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BACKGROUND

In this section we explore the importance of software in video game development, the generation of content for video games, and the real-world context that we make use of on our experiment to perform the corresponding tasks.

1.1 Software in video games

The development process of video games requires a harmonious combination of artistic elements and software integration, resulting in intricate and multifaceted creations. Software plays a crucial role in every aspect of a video game's creation, as it dictates the behavior and features that can be seen or experienced within the game. For instance, software is responsible for controlling the logic behind the behaviors of non-playable characters (NPCs) in a game. As video games evolve and become more sophisticated, the software powering them also becomes increasingly intricate.

Nowadays, most video games are developed by means of game engines. One can argue that game engines are software frameworks [13]. Game engines integrate a graphics engine and a physics engine as well as tools for both to accelerate development. The most popular ones are Unity and Unreal Engine, but it is also possible for a studio to make its own specific engine (e.g., CryEngine [5]).

One key artefact of game engines are software models. These are software models such as those proposed by the Model Driven Development paradigm [15] which should not be confused with either 3D Meshes or AI Models. Unreal proposes Unreal Blueprints [3], Unity proposes Unity Visual Scripting [14], and a recent survey in Model-Driven Game Development [19] reveals that UML and Domain Specific Language (DSL) models are also being adopted by development teams. Developers can use the software models to create video game content instead of using the traditional coding approach (C++ on Unreal or C# on Unity). While code allows for more control over the content, software models raise the abstraction level, thus promoting the use of domain concepts and minimizing implementation and technological details.

1.2 Content Generation for Video Games

The process of content generation for video games is typically slow, tedious, expensive, and susceptible to errors. Thus, leading to problems that the industry have such us: (1) excessive delays in content creation (with notorious examples in Cyberpunk 2077 [18] or GTA VI [10]) or (2) the ever-increasing demand for game content derived from post-launch updates, Downloadable Content (DLCs), games as a service, or platform-exclusive content.

To address these challenges, researchers have been exploring procedural content generation techniques as a potential solution to (semi-)automate the generation of new content within video games [8]. Procedural content generation can be grouped in three main categories according to the survey by Barriga *et al.* [2]: Traditional

techniques that generate content under a procedure without evaluation; Machine Learning techniques [9, 16] that train models to generate new content; and Search-Based techniques [17] that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

Content can also be created through reuse. In fact, since the term software engineering was coined at the NATO Conference held in Garmisch in 1968 [11], its evolution has been tied to the concept of reuse. Either applying an opportunistic approach such as clone-and-own [6], or applying systematic approaches as software product lines (assembling predefined features) [12] or as software transplantation (a feature is transplanted from a donor to a host) [1]. A recent SLR on game software engineering [4] identifies the relevance of both Reuse-based Content Generation (RCG) and Procedural Content Generation (PCG).

1.3 Kromaia Video Game for the Experiment

Kromaia is a commercial video game released on Playstation and Steam, translated into eight languages. On Kromaia, each level consists of a three-dimensional space where a player-controlled spaceship has to fly from a starting point to a target destination, reaching the goal before being destroyed. The gameplay experience involves exploring floating structures, avoiding asteroids, and finding items along the route, while basic enemies try to damage the spaceship by firing projectiles. If the player manages to reach the destination, the ultimate antagonist corresponding to that level (which is referred to as *boss*) appears and must be defeated in order to complete the level.

In the context of Kromaia, developers generate content through PCG by means of the work of Gallota *et al.* (which combines an L-system with an evolutionary Algorithm) [7] because it is specific for spaceships that can play the role of bosses, and it achieves the best state-of-the-art results for this type of content. Developers also generate content through RCG by means of reusing features between Kromaia´s content. Specifically, the developers select a feature (a fragment of content) from a donor, and a host (another content) that will receive the feature. Despite the research efforts in both PCG and RCG and the importance of content generation for video game development, there is no study that directly compares them.

2 DISCUSSION

Reuse of content on video games is not a popular practice. This is due to the prejudice of the developers that fears reusing to be repetitive. On the other hand, the randomness of PCG is perceived positively as an extension in the range of the creativity space for new content. Our experiment shows that this negative view of reuse is not aligned with the results. On the contrary, it reinforces the RCG pathway which boosts the latent content and leads to better results than PCG. During the focus group, subjects agree on that RCG was as a natural evolution of the original content. In contrast, PCG was negatively classified as content that did not appear to have been developed by humans.

Previous studies considered only players as the subjects. In our study we go one step ahead and analyse the differences between players and developers. For researchers it can be difficult to find developers to run experiments. However, that could not be the case for development studios. For instance, a large studio can enroll developers from different projects from the studio. This is relevant for studios because they put a lot of effort into enrolling players (not developers) for their games. It may seem paradoxical that it is hard to find players, but the experience of testing parts of a game in development is not the same as testing a full game as the developers in the focus group pointed out. Our experiment reveals not relevant differences in terms of statistical values between players and developers, suggesting that studios can leverage their developers. Furthermore, there is a relevant difference in terms of feedback in favor of developers: developers provided more beneficial feedback.

This experiment combines the specific quality aspects of video games ('design', 'difficulty', 'fun', and 'immersibility') and the rigorousness of more traditional software work. This includes the replication package that we have not found in previous work. One may think that the complexity of video games makes it difficult to design packages for replication. Nevertheless, we expect that our work along with the replication package available will provide a basis and inspiration for future researchers.

Availability Replication package is at:

Acknowledgements Omitted for blind review.

REFERENCES

[1] Earl T Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. 2015. Automated software transplantation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 257–269.

[2] Nicolas A. Barriga. 2019. A Short Introduction to Procedural Content Generation Algorithms for Videogames. *International Journal on Artificial Intelligence Tools* 28, 2 (2019), 1–11. https://doi.org/10.1142/S0218213019300011

[3] Unreal Blueprint. [n. d.]. Unreal Blueprint. https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/. Accessed: 01/02/24.

[4] Jorge Chueca, Javier Verón, Jaime Font, Francisca Pérez, and Carlos Cetina. 2023. The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games. *Information and Software Technology* (2023), 107330.

[5] CryEngine. [n. d.]. CryEngine. https://www.cryengine.com. Accessed: 01/02/24.

[6] Stefan Fischer, Lukas Linsbauer, Roberto E Lopez-Herrejon, and Alexander Egyed. 2015. The ECCO tool: Extraction and composition for clone-and-own. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 665–668.

[7] Roberto Gallotta, Kai Arulkumaran, and LB Soros. 2022. Evolving spaceships with a hybrid L-system constrained optimisation evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 711–714.

[8] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1 (2013), 1–22.

[9] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. 2021. Deep learning for procedural content generation. *Neural Computing and Applications* 33, 1 (2021), 19–37.

[10] Tuhin Das Mahapatra. 2023. Why is Rockstar delaying GTA 6? Here are some possible breakdowns. https://www.hindustantimes.com/technology/why-is-rockstar-delaying-gta-6-here-are-some-possible-breakdowns-101681440818791.html. Accessed: 01/02/24.

[11] Peter Naur and Brian Randell. 1969. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*.

[12] Klaus Pohl and Andreas Metzger. 2018. Software product lines. *The Essence of Software Engineering* (2018), 185–201.

[13] Cristiano Politowski, Fabio Petrillo, João Eduardo Montandon, Marco Tulio Valente, and Yann-Gaël Guéhéneuc. 2021. Are game engines software frameworks? A three-perspective study. *Journal of Systems and Software* 171 (2021), 110846.

[14] Unity Scripting. [n. d.]. Unity Scripting. https://unity.com/features/unity-visual-scripting. Accessed: 01/02/24.

[15] Bran Selic. 2003. The pragmatics of model-driven development. *IEEE software* 20, 5 (2003), 19–25.

[16] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgard, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270. https://doi.org/10.1109/tg.2018.2846639 arXiv:1702.00539

[17] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Trans. on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[18] Steve Watts. 2020. All The Cyberpunk 2077 Delays. https://www.gamespot.com/gallery/all-the-cyberpunk-2077-delays/2900-3618/. Accessed: 01/02/24.

[19] Meng Zhu and Alf Inge Wang. 2019. Model-driven game development: A literature review. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–32.