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 Procedural Content Transplantation (PCT) for content generation. 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 PCT) based on specific quality aspects of video games. 44 subjects including professional developers and players participated in our experiment. **Results** The results suggest that PCT generate 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. PCT has been underexplored so far because the reuse factor of transplantation 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 PCT 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, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BACKGROUND

Video games are complex creations that involve a seamless integration of art and software during the development process to produce the final product. 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 (often through state machines or decision trees). 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. Game engines are development environments that 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 [4]).

One key artefact of game engines are software models. Unreal proposes its own modeling language (Unreal Blueprints) [3], Unity proposes Unity Visual Scripting [11], and a recent survey in Model-Driven Game Development [16] 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. While code allows for more control over the content, software models raise the abstraction level, thus promoting the use of domain terms and minimizing implementation and technological details.

For instance, Kromaia is a commercial video game that works with his own game engine and software models from a DSL. 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. Kromaia's Bosses can be built either using C++ code or software models.

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, excessive delays in content creation (with notorious examples in Cyberpunk 2077 [15] or GTA VI [8]) or with 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 content generation techniques as a potential solution to (semi-)automate generation of new content within video games [6]. Content generation can be grouped in three main categories according to the survey by Barriga et al. [2]: Traditional techniques (TM) that generate content under a procedure without evaluation; Machine Learning (ML) techniques that train models to generate new content; and Search-Based (SB) techniques that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

Traditional techniques include pseudo-random number generators, cellular automata, or generative grammars. Developers must design an approach for a specific type of content which produces useful elements for that type of content without an evaluation or a learning process. The generation of vegetation is probably the most successful case of video game content generation through Traditional techniques, with the development of tools such as SpeedTree [14].

Machine Learning techniques train models to generate content based on training data, thus reducing the need for human domain-specific knowledge. The use of ML techniques for content generation was recently reviewed in 2018 by Summerville *et al.* [12], who could not identify any successful case study used by the industry. While the use of Deep Learning (a subfield of ML) has provided some advances in content generation as reviewed by Liu *et al.* [7], its application is still limited. DL approaches resist the specification and enforcement of explicit constraints, such as setting up the number of rooms in a dungeon. Constraints are important for game developers to achieve their vision of the content.

In 2010, Togelius *et al.* [13] surveyed SB techniques for content generation. SB techniques also do not suffer from the limitations of ML techniques, since they are easier to constrain and provide more accessible explanations of the generated results. In fact, some of the weaknesses of other techniques become a strength in SB techniques. For instance, in SB techniques, the objective function can use constraints to guide the search. SB approaches for PCG generate content that is evaluated within the algorithm prior to its use. During the evaluation phase, content is appraised to decide whether to use it, discard it, or recombine its building blocks to generate further content.

In our study we have identified the most relevant and close work in content generation literature to the context of the experiment. Gallota *et al.* [5] proposed a hybrid Evolutionary Algorithm for generating NPCs, which combines an L-system with a Feasible Infeasible Two Population Evolutionary Algorithm. We choose Gallota *et al.* as PCG baseline because (1) it is specific for spaceships that can play the role of bosses which is comparable to the content of the context of our experiment (Kromaia), and (3) it achieves the best state-of-the-art results for this type of content.

In our work, we want to study a new angle to tackle video games content generation through reusing existent content, inspired by software transplantation techniques [9, 10]. Software transplantation [1] is a more traditional software technique that introduced the transplantation metaphor into software development. In software transplantation, the developers will select an organ (a fragment of code) from a donor (a program), and a host (another program) that will receive the organ. The organ and the host will serve as inputs for a transplantation algorithm that will generate new software by automatically combining the organ and the host. In the context of content generation, the organ is a fragment of content from a video game content, and the host another video game content. The hypothesis is that a transplantation technique can release latent content that results from combining fragments of existing content. Furthermore, a transplantation approach provides more control to developers in comparison to current content generation techniques that are solely based on random generation, leading to results that are closer to developers' expectations.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Craig Miles, Arun Lakhotia, and Andrew Walenstein. 2012. In situ reuse of logically extracted functional components. *Journal in Computer Virology* 8 (2012), 73–84.

[10] Justyna Petke, Mark Harman, William B Langdon, and Westley Weimer. 2014. Using genetic improvement and code transplants to specialise a C++ program to a problem class. In *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*. Springer, 137–149.

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

[12] 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

[13] 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.

[14] Speed Tree. [n. d.]. Speed Tree. https://store.speedtree.com. Accessed: 01/02/24.

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

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