Automated Software Transplantation on Procedural Content Generation

Mar Zamorano
maria.lopez.20@ucl.ac.uk
mzamorano@usj.es
University College London
London, UK
Universidad San Jorge
Zaragoza, Spain

Carlos Cetina
ccetina@usj.es
University College London
London, UK
Universidad San Jorge
Zaragoza, Spain

Federica Sarro
f.sarro@ucl.ac.uk
University College London
London, UK

ABSTRACT

Procedural Content Generation (PCG) aims for the (semi) automatic generation of new content within video games. However, developers usually have limited control over the generation process. We propose the first transplantation algorithm in the field of PCG that allows game developers to choose an organ from a donor and a host that will receive the organ. Through our approach, we aim to search for an appropriate solution to integrate the organ into the host. We also study two distinct objective functions: The one accepted in the literature of software transplantation (test-based variant), and a novel objective function (simulation-based variant). In our evaluation, we have transplanted a total of 129 distinct organs from the scenarios of Kromaia (a commercial video game) into 5 video game bosses as hosts. Our simulation-based variant produces results that are 1.5 times superior to those of the test-based variant and 2.5 times superior to the PCG baseline. The statistical analysis confirms the significance of these differences and highlights the substantial magnitude of improvement. Furthermore, a focus group with game developers indicated their satisfaction with the generated content. Our analysis of the results also reveals organ interactions that have not been identified previously in the literature.

KEYWORDS

Automated Software Transplantation, Auto-transplantation, Procedural Content Generation, Search-Based Software Engineering, Model-Driven Engineering

ACM Reference Format:
Mar Zamorano, Carlos Cetina, and Federica Sarro. 2024. Automated Software Transplantation on Procedural Content Generation. In *Companion Proceedings of the 32nd ACM Symposium on the Foundations of Software Engineering (FSE '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

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

1 INTRODUCTION

The video games industry grows significantly every year [44]. In 2019, it became the largest entertainment industry in terms of revenue after surpassing the combined revenues of the movie and music industries [40]. In 2021, video games generated revenues of $180.3 billion [58], and in 2022, the estimated revenues were of $184.4 billion [59]. Overall, the sum of revenues generated from 2020 to 2022 was almost $43 billion higher than those originally forecasted for the period.

Video games are complex creations where art and software go hand in hand during the development process to conform the final product. Hence, development teams are conformed by different profiles, where the majority are software developers (24%), but also include game designers (23%), artists (15%), UI designers (8%), and QA engineers (5%), based on a recent survey with professional game developers [48]. In a video game, software permeates every aspect of the development, since it governs all the elements and actions that can appear or happen within the game. For instance, software controls the logic behind the actions of NPCs[1] within a game (often through state machines or decision trees). As video games become more and more advanced, their software also becomes more complex.

To alleviate the complexity of video game development, most video games are developed using game engines. The most popular video game engines are Unity [2] and Unreal [3]. Game engines are development environments that integrate a graphics engine and a physics engine, as well as tools to accelerate development. For example, they provide a ready-to-use implementation of gravity or collisions between elements. Game engines significantly speed up the development of video games. However, for game developers, the main challenge is to develop the game content. Game content includes from game scenarios to NPCs or game items such as weapons.

Content generation is generally a slow, tedious, costly, and error-prone manual process. To cope with the growing demand for content for video games, researchers have been working towards Procedural Content Generation (PCG). PCG refers to the field of knowledge that aims at the (semi) automatic generation of new content within video games [20]. Current PCG approaches work as follows: Developers provide initial content (usually human-generated content) into an algorithm. Afterwards, the algorithm (Traditional, Machine Learning, or Search-Based methods) will generate new content. This far only

[1] Non-playable characters.
[2] https://unity.com/
[3] https://www.unrealengine.com/

a few traditional methods have succeeded in providing tools used by the industry to randomly generate vegetation (e.g., SpeedTree [4] in Unreal and Unity).

In this paper, we propose a new angle to tackle PCG for video games inspired by transplantation techniques [4], which we named Procedural Content Transplantation (PCT). In medicine, *transplantation* is a procedure in which cells, tissues, or organs of an individual are replaced by those of another individual, or the same person [15]. In software, researchers understand transplantation as a procedure in which a fragment (organ) of a software element (donor) is transferred into another software element (host) [4]. Software transplantation has been successful for different tasks: program repair [47, 57], testing [62], security [60], and functionality improvements [46].

Our PCT proposal introduces for the first time the transplantation metaphor into PCG. In our approach, the developers of a game will select an organ (a fragment of video game content) from a donor (video game content) and a host (another video game content) that will receive the organ. The organ and the host will serve as inputs for our transplantation algorithm that will generate new content for the game by automatically combining the organ and the host. Our hypothesis is that our transplantation approach can release latent content that results from combining fragments of existing content. Furthermore, our transplantation approach provides more control to developers in comparison to current industrial approaches that are based on random generation, leading to results that are closer to developers' expectations.

Our approach, called IMHOTEP[5], relies on Search-based Software Engineering (SBSE) because SBSE has demonstrated success in software transplantation [4]. In the literature, software transplantation approaches guide the search by using test-suites, *i.e.*, the transplantation assessment is determined by the amount of tests that a candidate solution is able to pass. Our work not only explores the use of test-suite ($T_{Imhotep}$) but also the use of video game simulations ($S_{Imhotep}$) to guide the search. Our hypothesis is that it is possible to harness video games' NPCs to run simulations that provide data to asses the transplantations. Within video games, it is typical to find NPCs that serve as companions to the player, adversaries to defeat, or inhabitants of the virtual world. These NPCs have pre-programmed behaviours that could be used in game simulations. For instance, in a first-person shooter game (like the renowned Doom video game), NPCs explore the game scenarios in search of weapons and power-ups to engage in combat with other NPCs or the player.

We have evaluated our proposal over the Kromaia case study. Kromaia is a commercial video game about flying and shooting with a spaceship in a three-dimensional space[6]. The game has been released on PC, PlayStation, and translated to eight different languages. To evaluate IMHOTEP, we transplant 129 different organs extracted from the scenarios of Kromaia into 5 of the video game bosses that act as hosts, generating new video game bosses. In total, our approach analysed 645 transplants. To the best of our knowledge, our work has more transplants than previous work in the literature, which achieved a maximum of 327 successful transplants [41].

[4] https://store.speedtree.com
[5] Our approach is named after IMHOTEP, who is considered by many to have written the Edwin Smith Papyrus (the oldest known manual of surgery).
[6] See the official PlayStation trailer to learn more about Kromaia: https://youtu.be/EhsejJBp8Go

We compare the results of the two IMHOTEP variants ($T_{Imhotep}$ and $S_{Imhotep}$) to the PCG baseline from the literature [18]. To make the comparison, we rely on the concept of game quality and its automated measurement, which is widely accepted in practice [9].

The results show that, out of the three approaches (the two IMHOTEP variants and the baseline), the content generated through the $S_{Imhotep}$ variant obtains the best results. This approach yields 1.5x better results than $T_{Imhotep}$ and 2.5x better results than baseline. The statistical analysis shows that the differences are significant, and the magnitude of improvement is large.

To the best of our knowledge, this is the first work that leverages transplantation to generate video game content, obtaining more favourable solutions than the baseline in an industrial setting. Specifically, we claim that:

- The results show that PCG through transplants is not only feasible, but desirable. Our approach has significantly outperformed classic content generation in the evaluation of this work, opening a new road towards tackling the pressing problem that the industry has with excessive delays in content creation (with notorious examples in Cyberpunk 2077 [56] or GTA VI [29]) and 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 this date, this is the transplantation work with the most successful transplants - almost double than its pursuer. Moreover, the transplants are carried out in an real-world industrial context in contras to the academic context of the pursuer.
- Our work returns control to the hands of the developers through organ selection. In comparison, the most successful industrial approaches (such as SpeedTree) lie on chance. The generated content is hence more in line with the intent of developers, as discussed in the focus group. This is key towards the real-world industrial usage of the generated automatically content.
- Our work reveals that harnessing simulations rather than test suites leads to significantly better results. This may empower software transplantation researchers to reconsider the usage of test suites in their work. Additionally, the upgrowth of digital twins encourages studying simulation-guided transplants in other domains beyond video games.
- Our analysis of the results reveals interactions between organs that have not been identified previously in the literature. These interactions are a promising line of research to advance the field of software transplants.
- Finally, this work is also relevant towards raising awareness for the need of research in the software part of video games. Despite the importance of software for video games, and the increase in the relevance of video games themselves in our society, video games remain as a relatively unexplored topic that has not received much attention from the software engineering research community.

The rest of the paper is structured as follows: Section 2 provides some background to better understand our work. Section 3 describes our approach, depicting its usage for PCG. Section 4 details the evaluation of our approach. Section 5 highlights the results of our research. Section 6 discusses the outcomes of the paper and future lines of work. Section 7 outlines the threats to the validity of our work. Section 8 reviews the works related to this one. Finally, Section 9 concludes the paper by summarizing the main contributions and results.

2 BACKGROUND

2.1 Model-driven video game development

Video games are pieces of software that, like any other software, need to be designed, developed, and maintained over time. However, there are some particularities of video games that make them differ from traditional software, such as the artistic component of the videogame, the complexity of the rendering pipelines, the heterogeneous nature of video game development teams, and the abstract nature of the final purpose of a video game: fun.

Hence, video games present characteristics that differentiate their development and maintenance from the development and maintenance of classic software. Examples of these differences can be found in how video game developers must contribute to the implementation of different kinds of artifacts (e.g., shaders, meshes, or prefabs) or in the challenges they face when locating bugs or reusing code [11, 36].

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 [7]).

One key artifact of game engines are software models. Unreal proposes its own modeling language (Unreal Blueprints), and a recent survey in Model-Driven Game Development [63] 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. Through software models, developers are freed from a significant part of the implementation details of physics and graphics, and can focus on the content of the game itself (see Fig. 1).

2.2 Kromaia

The research presented in this paper is framed within the context of a commercial video game case study, Kromaia. In particular, our evaluation uses the bosses of the video game to evaluate the approach. Each level of Kromaia 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 final boss corresponding to that level appears and must be defeated in order to complete the level.

[7] https://www.cryengine.com

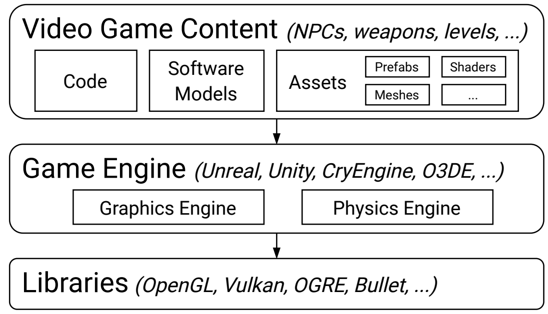

Figure 1: Overview of video game artifacts.

Bosses can be built either using C++ code or software models. The top part of Figure 2 depicts a boss fight scenario where the player-controlled ship (item A in the figure) is battling The Serpent (item B in the figure), which is the final boss that must be defeated in order to complete Level 1. The bottom part of the figure illustrates the two possible development approaches for the Serpent boss.

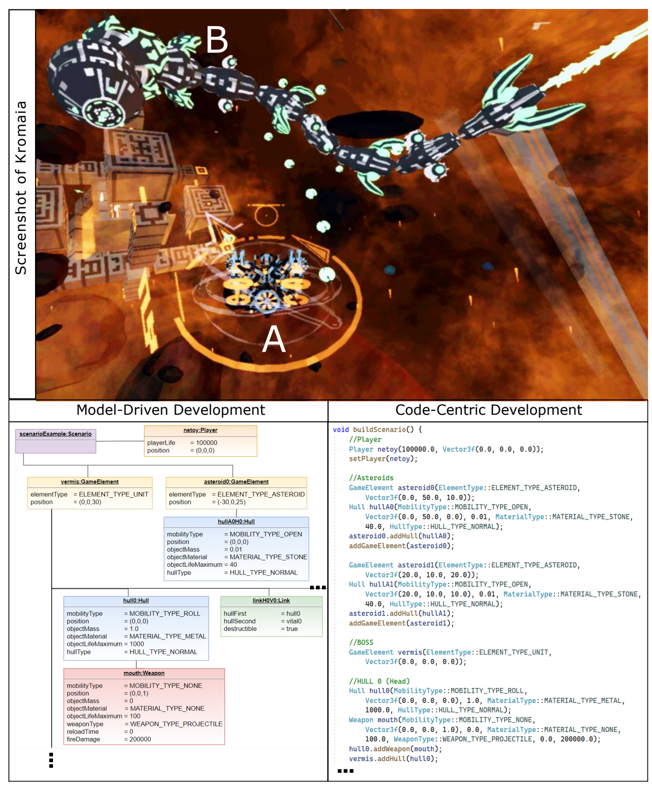

Figure 2: Model-Driven Development vs. Code-Centric Development in the context of Kromaia

Even though Figure 2 shows excerpts of the implementation of the Serpent both in the form of software models and code, it is not necessary to realize both in order to implement this content. Although developers can mix both technologies, developing different parts of the boss using one or the other indistinctly, they are also free to implement the content using software models exclusively or to do so purely via code. However, the heterogeneous nature of video game development teams - comprised majorly of programmers [48], but also counting game designers, artists, UI designers, and QA engineers within their ranks - possibly favours the use of software models over code fact the higher abstraction level of the former (combined with their detachment from more technical implementation details) empowers less tech-focused roles to embrace a more active participation in development tasks. Furthermore, an experiment [14] confirmed that video game developers make fewer mistakes and are more efficient when working with models rather than code.

Within the context of Kromaia, the elements of the game are created through software models, and more specifically, through the Shooter Definition Model Language (SDML). SDML is a DSL model for the video game domain that defines aspects that are included in video game entities: the anatomical structure (including their components, physical properties, and connections); the amount and distribution of vulnerable parts, weapons, and defences; and the movement behaviours associated to the whole body or its parts. SDML has concepts such as hulls, links, weak points, weapons, and AI components, and allows for the development of any game element, such as bosses, enemies, or environmental elements. The models are created using SDML and interpreted at runtime to generate the corresponding game entities. In other words, software models created using SDML are translated into C++ objects at runtime using an interpreter integrated into the game engine [7]. More information on the SDML model can be found in the following video presentation: https://youtu.be/Vp3Zt4qXkoY.

3 OUR IMHOTEP APPROACH

This section explains how our IMHOTEP approach makes use of evolutionary computation to transplant organs within video games content. We first present an overview of our approach and subsequently provide the details of the approach. To help the reader, we provide along with the approach explanation an example of transplantation of content within a simplified version of 'bosses' of the video game Kromaia.

Fig. 3 shows an overview of our approach. At the top left of the figure we show the input to our approach, namely the organ to be transplanted from the donor and the host where the organ will be transplanted into. Afterwards, IMHOTEP detects the points of the organ that allows the transplantation and the points where the organ can be inserted into the host. To initialize the population of the evolutionary algorithm, the organ is cloned and transplanted in a random point. Genetic operations generate potential solutions for transplantation, while the objective function asseses the quality of these solutions. This process of generating and assessing is repeated until a specific stop condition is met. When the evolutionary algorithm finishes the execution we obtain a ranked list based on the given objective function of the best transplantations between organ and host.

In video games, software models are popular (compared to classic software) possibly because they facilitate the participation of non-programmers (e.g., artists) in the development process. Therefore, our IMHOTEP approach is designed to work with models. Although we illustrate the running example with the SDML models of the case study, our approach is generic and can be used with other modelling languages because it exploits the idea of boundaries between model elements. Next, we describe each step of IMHOTEP in the following subsections.

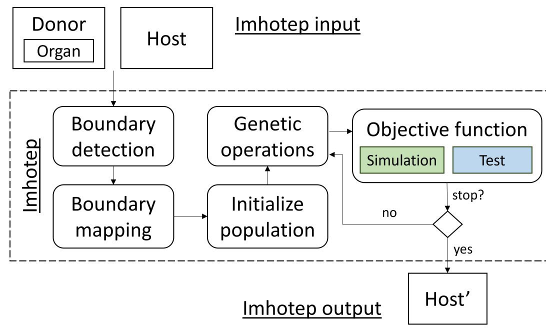

Figure 3: Overview of our IMHOTEP approach.

3.1 Input selection

IMHOTEP requires the developers to identify a source model content (donor) with the organ that will be transplanted, and a target model content (host). In our running example we present a simplified version of the metamodel, and the corresponding concrete syntax of the model (see Fig. 4) from the video game Kromaia. 'Hulls' serve as the structural framework that define the anatomical composition of the models. For example, the boss presented in Fig. 2 (identified as 'B') has its body built by hulls. 'Weak points' are conceptual elements that possess the vulnerability to be harmed. 'Weapons' are tangible items capable of causing harm through direct contact, such as discharging projectiles like bullets. Hulls, weak points, and weapons are attached between them through 'Links'.

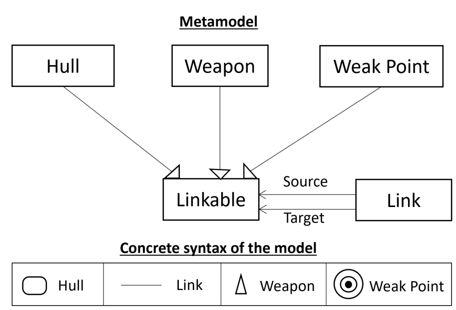

Figure 4: Simplified metamodel with the corresponding concrete syntax of the model.

In the running example, the source donor model is a simplified version of an original 'boss' from Kromaia, called 'Serpent'. Fig. 5 (a) shows the graphical representation of the donor's model, differentiating each element of the model with a letter from A to S. It also shows with dashed lines the elements selected as organ (the elements H, I, J, K, N, O, P, Q). This simplified example is inspired by the boss shown in Fig. 2 with letter B. In the running example, the host is a model of a regular enemy that could appear in Kromaia. Fig. 5 (b) shows the graphical representation of the host model.

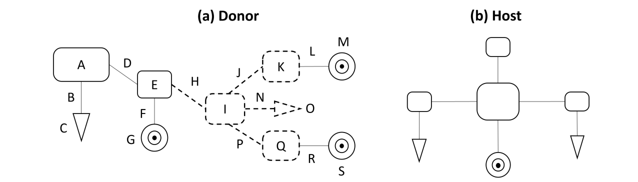

Figure 5: (a) Donor model with organ selection in dashed lines. (b) Host model.

3.2 Boundary detection

To transplant an organ into a host we need to find a way to connect them. To do that we use the boundaries between the model elements of the organ and the host. A boundary is a connection point capable of connecting two distinct model elements within a model. The connection is restricted by the rules of the metamodel. In the simplified example in Fig. 4, the Source and Target meta-relationships are the boundaries between the model elements of the models conforming to that metamodel. In other metamodel languages, there will be other meta-relationships with other names that will be the boundaries.

IMHOTEP automatically identifies the boundaries of the selected organ, and all the boundaries of the host. In the running example, the boundaries of the organ are the connection points between donor and host. The elements that connect with the rest of the donor are H, K, and Q. Fig. 6 (a) shows the donor, the organ, and the boundaries (boundaries are represented by a circle crossed). The boundaries of the organ are as follows: b11 for the H element; b16 for the K element, and b25 for the Q element.

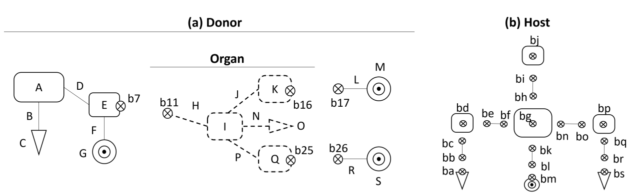

Figure 6: (a) Donor model boundaries. (b) Host model boundaries. The boundary is represented by a circle crossed.

On the other hand, the boundaries of the host are all the points where its model elements connect. Figure 6 (b) shows all the boundaries of the host of the running example. The host has a total of 19 boundaries identified by a tag from ba to bs.

3.3 Boundary mapping

In the boundary mapping step, IMHOTEP determines the mapping between the boundaries of the organ and the host. For each boundary in the organ, IMHOTEP considers all compatible boundaries of the host, including the possibility of not connecting the boundary to the host boundaries. The boundary compatibility is determined by the metamodel.

Table 1 shows a boundary mapping between the organ and the host of the running example. The boundary b11 is a boundary from a 'Link' from the model and according to the metamodel it can connect to any 'Hull', 'Weapon', 'Weak Point'. The boundaries b16 and b25 are both 'Hulls' and they can connect with any 'Link'.

Organ boundaries	Host boundaries	
b11	ba	bm
	bd	bp
	bg	bs
	bj	Not connected
b16 b25	bb	bc
	be	bf
	bh	bi
	bk	bl
	bn	bo
	Not connected	

Table 1: Mapping of compatible boundaries between organ and host.

3.4 Initialize population

In evolutionary algorithms a population is a collection of possible solutions for a problem. The encoding is the problem representation that an algorithm is capable to understand.

In our work, the encoding requires a binary vector that represents the organ in the donor, and the boundary mapping (see Fig. 7). In the binary vector, each element from the model is a position from the vector. If a position in the vector has a '1', it means that the element from the model is part of the organ. On the other hand, each boundary from the organ gets assigned a compatible boundary from the host. The initial population of IMHOTEP contains individuals composed by the host and the organ placed in a random position (a random mapping between the organ boundaries and the compatible organ boundaries).

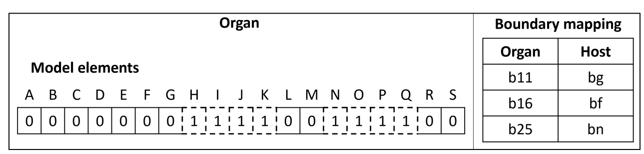

Figure 7: Example of encoding.

3.5 Genetic Operators

IMHOTEP uses genetic operations (selection, crossover, and mutation) to generate new individuals (*i.e.* candidate solutions).

To select the individuals, we use the ranking selection, which ranks the population by the objective function and takes the top individuals in the current population.

We use a single, random, cut-point crossover, which starts by selecting and splitting two parent solutions at random. When two parent individuals are selected, a random cut point is determined to split them into two sub-vectors. Then, the crossover creates two children solutions by combining the first part of the first parent with the second part of the second parent for the first child, and the first part of the second parent with the second part of the first parent for the second child. Finally, the new individual has a probability to mutate any value of the encoding.

Fig. 8 shows example of new individuals that could results from our running example. For simplicity, these individuals have unaltered organs, but illustrate different boundary mappings between organ and host.

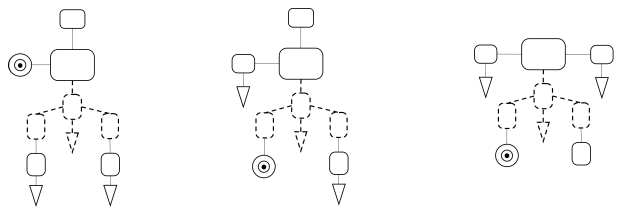

Figure 8: Example of individuals.

3.6 Objective function

Our work proposes to harness video games' NPCs to run simulations that provide data to assess the transplantations. The first thing that differentiates video games from traditional software is that the basic requirement of video games is 'fun'. 'Fun' is an abstract concept and the developers are in charge of interpreting it. In fact, different developers may have different interpretations. For some game developers, 'fun' is achieved with a difficult game that is very rewarding when progress is made (e.g., Dark Souls [28]). While for other developers, 'fun' is achieved by effortlessly killing enemies (e.g., Dynasty Warriors [35]). Therefore, we argue that the developer intent is key for content generation.

Specifically, we propose to introduce the generated content (each individual in the population) into a simulation of the video game. The simulation produces a data trace of the events that have occurred. Using the data from the trace, we can check how well aligned are the events with the intention of the developers. In the running example, the simulation is a duel between a spaceship and a boss. The simulation generates data about the duel, such as the damage inflicted. The intention of the developers may be that the duel ends with the victory of the spaceship with a remaining life of less than 10%.

Our proposal does not require ad hoc development of simulations. We propose that the simulations leverage mainly the NPCs (but also more video game elements, such as scenarios or items like weapons or powerups). NPCs are naturally developed during the development process of a video game. In other words, NPCs are integral components of most video game genres such as First-Person Shooter (FPS), Real-Time Strategy (RTS), our racing games. We aim two goals with the aforementioned. On the one hand, it makes the use of simulations cheaper, i.e. it does not involve additional development costs, and secondly, it facilitates fidelity to the video game compared to ad hoc development. In the running example, during the simulation, the generated content is the boss, who can be accompanied by more NPCs acting as secondary enemies. Additionally, the spaceship that confronts the boss is an NPC representing an allied ship. Finally, the scenario, and items such as weapons or powerups also belong to the game itself.

In this work, the Simulation-based IMHOTEP ($S_{Imhotep}$) assesses the transplants through a simulation of a game battle between the boss (Host') and a NPC spaceship. The information retrieved from the simulation is the data that the developers regard as relevant, using their domain knowledge. Hence, our approach takes into account the percentage of simulated player victories ($F_{Victory}$) and the percentage of simulated player health left once the player wins a duel (F_{Health}). The calculation of $F_{Victory}$ and F_{Health} is performed in the same way as Blasco *et al.* [7], as described below:

$F_{Victory}$ is calculated as the difference between the number of human player victories (V_P) and the optimal number of victories (33%, according to the developers of Kromaia and their criteria) ($V_{Optimal}$):

$$F_{Victory} = 1 - \frac{\mid V_{Optimal} - V_P \mid}{V_{Optimal}} \quad (1)$$

F_{Health}, which refers to completed duels that end in spaceship victories, is the average difference between the spaceship's health percentage once the duel is over (Θ_P) and the optimal health level that the spaceship should have at that point ($\Theta_{Optimal}$, 20%, according to the developers):

$$F_{Health} = 1 - \frac{\sum_{d=1}^{V_P} \frac{|\Theta_{Optimal} - \Theta_P|}{\Theta_{Optimal}}}{V_P} \quad (2)$$

$F_{Overall}$ is an average fitness value for a boss model that includes the fitness criteria described above. $F_{Overall}$ also includes a validation part. The validation part is to avoid models with inconsistencies. The validity of the models is performed by a run-time interpreter that is part of the game. When the model is stated as non-valid the value of Validity will be 0. $F_{Overall}$ can assume a value in [0, 1] which is used to assess a boss model when our IMHOTEP approach is applied to the Kromaia case study.

$$F_{Overall} = minValidity, \frac{\sum_{i=1}^{N} F_i}{N} \quad (3)$$

4 EXPERIMENTAL DESIGN

In this section we explain how we empirically evaluate our proposal for automated transplantation in video games through IMHOTEP. Through this section, we present the research questions that we aim to answer, the evaluation process, and the implementation details.

4.1 Research Questions

$S_{Imhotep}$ propose a new angle for PCG, and for that reason we want to compare it to the established practice in the video game field in our first research question:

RQ$_1$: *How does $S_{Imhotep}$ perform with respect to the PCG baseline?*

$S_{Imhotep}$is also a new angle to guide the transplantation. We propose to leverage the possibilities of the vide game domain by means of simulations, so we want to compare it to the established practice (test suite to guide the transplantation) in the software transplantation field. This motivates our second research question:

RQ$_2$: *What is the performance in terms of solution quality of $S_{Imhotep}$ and $T_{Imhotep}$?*

4.2 Planning and execution

Fig. 9 shows an overview of the evaluation. The white background part at the top shows the assets of the game itself (content) and the game development (test suite) that are used by the approaches. The grey background part in the middle shows the inputs and outputs of the three approaches (the two IMHOTEP variants and the baseline). Finally, the white background part at the bottom shows the evaluation of the results of the approaches.

We used the work by Gallota *et al.* [18] as PCG baseline. Gallota *et al.* proposed a hybrid Evolutionary Algorithm for generating NPCs. Specifically, their approach combines an L-system with a Feasible Infeasible Two Population Evolutionary Algorithm. We choose Gallota *et al.* as PCG baseline because (1) this work is specific for spaceships that can play the role of bosses which is comparable to the content of the case study, and (2) it achieves the best state-of-the-art results for that content.

In the evaluation we also include a test-based variant of IMHOTEP ($T_{Imhotep}$). In this variant, the assessment is carried out by passing tests. The reason for including this variant is that in classic software transplantation the best results have been achieved by using the test suite for the assessment. For the $T_{Imhotep}$ we ask the developers to provide the set of tests that are relevant for the evaluation. The developers from Kromaia provided us with a total of 243 tests selected based on their domain knowledge. The objective value will be retrieved when each individual pass through the 243 tests, normalized in a scale of [0, 1]. An individual which passes the 243 tests will obtain 1.0, on the contrary if it does not pass any test it will obtain 0.0. As in the $S_{Imhotep}$, the each individual needs to pass through a validation, giving 0.0 to those that are not valid.

For the evaluation we used five different hosts (Vermis, Teuthus, Argos, Orion, and Maia), which are the full set of original bosses from the video game Kromaia. As donors, Kromaia developers considered all Kromaia's scenarios to identify 129 organs. Each host has more than a thousand model elements. Organs have an average of 255 model elements.

Each organ was transplanted individually to each boss. Each variant of IMHOTEP provided a total of 645 new individuals (5 hosts * 129 organs) as output, 645 new individuals from $S_{Imhotep}$ and 645 individuals from $T_{Imhotep}$. In the case of the baseline (which relies on the L-system to generate the content instead of transplanting organs), to make it a fair comparison, the baseline was executed 129 times with each one of the 5 different hosts to obtain 645 new individuals. In addition, we executed 30 independent runs (to account for random variation as suggested by Arcuri and Fraser [3]). Hence, we obtain a total of 58050 independent runs (645*3*30).

To compare the solutions of the baseline and the two variants of IMHOTEP ($S_{Imhotep}$ and $T_{Imhotep}$), we rely on the concept of game quality and its automated measurement through simulated players. The results of Browne *et al.* demonstrated the validity of the approach which is widely accepted in the research community [9]. Therefore, we need two ingredients to run our experiment: The simulated player and the automated measurement.

The simulated player, developed by the developers of the Kromaia video game, possesses the ability to mimic human player behaviour. Our approach incorporates their algorithm, utilizing it to simulate battles between the generated bosses and the simulated player. Within these simulations, the simulated player confronts the boss, strategically targeting and destroying its weak points. Meanwhile, the boss operates in accordance with its anatomical structure, behavioural patterns, and attack/defensive dynamics, aiming to overcome the simulated player. Both entities within the simulation actively strive to emerge victorious, eschewing draws or ties, and ensuring a definitive win. The algorithm grants the simulated player the capability to employ various strategies when engaging with a boss, as it can be parameterized to define its fighting approach. The simulation parameters were provided by the developers, who analysed battles between human players and bosses to inform their decision-making.

The automated measurement is $Q_{Duration}$ which was proven to achieve good results [9]. The duration of duels between simulated players and bosses units is expected to be around a certain optimal value. For the Kromaia case study, through tests and questionnaires with players, the developers determined that concentration and engagement for an average boss reach their peak at approximately 10 minutes ($T_{Optimal}$), whereas the maximum accepted time was estimated to be 20 minutes ($2 * T_{Optimal}$). Significant deviations from that reference value are good design-flaw indicators: short games are probably too easy; and duels that go on a lot longer than expected tend to make players lose interest. The criterion $Q_{Duration}$ is a measure of the average difference between the duration of each duel (T_d) and the desired, optimal duration ($T_{Optimal}$):

$$Q_{Duration} = 1 - \frac{\sum_{d=1}^{Duels} \frac{|T_{Optimal} - T_d|}{T_{Optimal}}}{Duels} \quad (4)$$

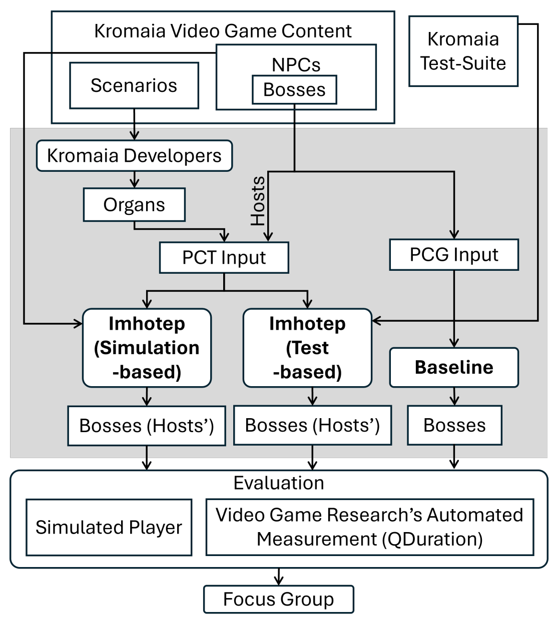

Figure 9: Overview of the evaluation process.

4.3 Implementation details

We chose the parameters shown in Table 2 to calibrate our IMHOTEP approach. We established the stop condition at 2 minutes and 30 seconds, ensuring that the approaches run long enough to obtain suitable solutions. The focus of this paper is not to tune the values to improve the performance of the approaches when applied to a specific problem, but rather to compare the performance of the approaches in terms of solution quality on a level playing field.

The evaluation of IMHOTEP and the baseline was done using two PCs with the following specifications; Intel Core i7-8750H, 16GB; and 2x Intel(R) Xeon(R) CPU X5660, 64GB. The implementation uses the Java(TM) SE Runtime Environment (JDK 1.8), together with Java as the programming language. For purposes of replicability and extension of our work, the implementation source code and the data are publicly available at the following URL: https://anonymous.4open.science/r/Imhotep/

Table 2: IMHOTEP parameter settings

Parameter description	Value
Stop Criterion	2m 30s
Population size	100
Number of parents	2
Number of offspring	2
Crossover probability	1
Mutation probability	1/150

5 RESULTS

In this section, we present whether there is any statistical significance between the results obtained by the two variants of IMHOTEP ($S_{Imhotep}$ and $T_{Imhotep}$) and the PCG baseline. To do that, we perform the Mann-Whitney U [30] setting the confidence limit, α, at 0.05, and applying the Bonferroni correction (αK, where K is the number of hypotheses) when multiple hypotheses are tested.

Unlike parametric tests, the Mann-Whitney U raises the bar for significance, by making no assumptions about underlying data distributions. Moreover, we used effect size to assess whether the statistical significance has practical significance effect size [2]. To this end we use the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size measure, as it is recommended to use a standardised measure rather than a pooled one like the Cohen's d when not all samples are normally distributed [2], as in our case. The $\hat{A}_{12}$ statistic measures the probability that an algorithm A yields greater values for a given performance measure M than another algorithm B, based on the following equation:

$$\hat{A}_{12} = R_1 m - m\ 12n \quad (5)$$

where R_1 is the rank sum of the first data group we are comparing, and m and n are the number of observations in the first and second data sample, respectively. Values between $0.44, 0.56$ represent negligible differences, values between $0.56, 0.64$ and $0.36, 0.44$ represent small differences, values between $0.64, 0.71$ and $0.29, 0:44$ represent medium differences, and values between $0.0, 0.29$ and $0.71, 1.0$ represent large differences.

The mean values and standard deviations for $Q_{Duration}$for each IMHOTEP variants and the baseline are presented in Table 3. Both variants ($S_{Imhotep}$ and $T_{Imhotep}$) obtained better results than the baseline (Base). Specifically, $S_{Imhotep}$ yielded the best results, followed by $T_{Imhotep}$ and then baseline.

Figure 10 shows the results of the evaluation execution of our approach when using the two objective functions (simulation-Based and test-Based) from IMHOTEP and the Baseline. The executions are grouped by each host (boss of Kromaia) that has been used in our experiment (Argos, Maia, Orion, Teuthus, and Vermis). The last column, with shaded background, shows the average of all the hosts for each objective function and the baseline.

Each boxplot is generated from the results of each host obtained from transplantation IMHOTEP or generation (baseline). Each boxplot represents 645 values of a specific host-organ transplantation (IMHOTEP) or 645 generations from a specific host (baseline). Each value in the boxplot is the mean value (between the 30 independent runs) of the quality indicator ($Q_{Duration}$) for one of the transplants (IMHOTEP) or generations (baseline).

5.1 Research Question 1: $S_{Imhotep}$ vs Baseline

The variants obtained an average value of 44.85% in $Q_{Duration}$, with $S_{Imhotep}$ being the variant that obtained the best results overall (53.31% in $Q_{Duration}$). $T_{Imhotep}$obtained 36.39% in the overall $Q_{Duration}$, which also outperformed the baseline. The baseline obtained the worst $Q_{Duration}$. All in all, the results reveal that leveraging simulations as objective function pays off in the context of PCT, yielding 1.5x better results than the test-based variant and 2.5x better results than baseline.

5.2 Research Question 2: $S_{Imhotep}$ vs $T_{Imhotep}$

The obtained $p-values$ for $Q_{Duration}$ are less than $2.2x10^{-16}$. Since the $p-Values$ are smaller than 0.05, we can state that there are differences among the algorithms for the quality indicator of $Q_{Duration}$.

Table **??** shows the $p-Values$ of the Holm's post hoc analysis for each pair-wise comparison and quality indicator. All of the $p-Values$ obtained in $Q_{Duration}$ were smaller than their corresponding significance threshold value (0.05), indicating that the differences in solution quality between the two variants and the baseline are significant.

RQ$_2$ answer. Since the Holm's post hoc $p-values$ for $Q_{Duration}$ (shown in Table **??**) are smaller than 0.05, we can state that there are significant differences between the variants and the baseline.

When comparing algorithms with a large enough number of runs, statistically significant differences can be obtained even if they are so small as to be of no practical value [2]. Thus, it is important to assess if an algorithm is statistically better than another and to assess the magnitude of the improvement. Effect size measures are needed to analyze this.

For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [19, 52]. $\hat{A}_{12}$ measures the probability that running one algorithm yields higher values than running another algorithm. If the two algorithms are equivalent, then $\hat{A}_{12}$ will be 0.5.

Table **??** shows the values of the effect size statistics between pair-wise comparisons of algorithms in Kromaia. Specifically, the upper part of the table shows the $\hat{A}_{12}$ values, whereas the lower part of the table shows Cliff's Delta [12] values for $Q_{Duration}$. From the results, we can determine that the performance results obtained by IMHOTEP with the simulation-based variant, IMHOTEP with the test-based variant, and the PCG baseline are significant in $Q_{Duration}$. The magnitude of improvement using IMHOTEP instead of the baseline

Table 3: Mean values and standard deviations for $Q_{Duration}$ for each approach.

	Argos	Maia	Orion	Teuthus	Vermis	Overall
$S_{Imhotep}$	43.92 ± 9.30	43.08 ± 12.09	48.86 ± 8.69	60.78 ± 7.38	69.90 ± 10.52	53.31 ± 14.26
$T_{Imhotep}$	32.17 ± 6.94	29.52 ± 9.34	31.41 ± 6.83	46.33 ± 10.54	42.50 ± 12.96	36.39 ± 11.72
Baseline	20.15 ± 1.86	8.43 ± 1.81	32.97 ± 0.85	19.53 ± 1.88	25.48 ± 3.31	21.31 ± 8.32

Table 4: Mann-Withney U pair-wise test results / Vargha-Delaney $\hat{A}_{12}$ effect sizes obtained comparing $S_{Imhotep}$ and the PCG Baseline. $\hat{A}_{12}$: Large – L.

Boss	$p-Value$ / $\hat{A}_{12}$
Argos	$3.25x10^{-23}$ / 0.99 (L)
Maia	$3.25x10^{-23}$ / 1.0 (L)
Orion	$4.01x10^{-23}$ / 0.98 (L)
Teuthus	$3.25x10^{-23}$ / 1.0 (L)
Vermis	$3.25x10^{-23}$ / 1.0 (L)
Overall	$1.41x10^{-107}$ / 0.98 (L)

Table 5: Mann-Withney U pair-wise test results / Vargha-Delaney $\hat{A}_{12}$ effect sizes obtained comparing $S_{Imhotep}$ and $T_{Imhotep}$. $\hat{A}_{12}$: Large – L.

Boss	$p-Value$ / $\hat{A}_{12}$
Argos	$1.28x10^{-18}$ / 0.85 (L)
Maia	$6.64x10^{-18}$ / 0.85 (L)
Orion	$4.95x10^{-22}$ / 0.95 (L)
Teuthus	$3.60x10^{-18}$ / 0.87 (L)
Vermis	$8.86x10^{-23}$ / 0.95 (L)
Overall	$6.58x10^{-93}$ / 0.82 (L)

can be interpreted as being large based on the magnitude scales [43] of the Cliff's Delta values. Hence, IMHOTEP has an actual impact on performance. The highest differences between IMHOTEP and the baseline are obtained when the simulation-based variant is used, obtaining better results in $Q_{Duration}$ for 97% of the runs.

We can draw conclusions about how much the quality of the solution is influenced by each variant of IMHOTEP compared to baseline from the results of Table **??**. The results reveal that the magnitude of improvement in $Q_{Duration}$ using any variant is large compared to the baseline according to the magnitude scales [43] of the Cliff's Delta values.

6 DISCUSSION

To begin with, our work revolves around the transplantation of organs between two very different types of content in video games: scenarios and bosses. One may wonder why not transplanting organs between contents of the same type, such as between bosses. Technically, it should also be a smaller challenge to transplant organs among the same type of content due to the similarities and shared structures. However, video games put the focus on fun, which is many times achieved by avoiding repetition. Since the number of bosses is usually very limited in video games, transplanting between bosses could lead to repetition, hurting fun and creating negative play experiences for the players. In contrast, scenarios provide an abundant and promising source of organs that can withstand repetition, since it is frequent for a relevant portion of a scenario to not be explored by a player during a game: while players spend most of the time playing within scenarios, the focus of scenarios on completing goals combined with their sheer extension renders them difficult to explore in full. Hence, reusing between bosses and scenarios is more original and relevant for fun. As future work, we will also explore conducting transplants between contents of different games.

Since transplanting an organ to a host contributes to generating new desirable content, one might consider performing more than one transplant on the same host to continue creating novel content. In its current state, our approach allows for only one organ to be transplanted at a time, but it should be possible to repeatedly transplant the same organ onto the same host, or to consider chains of transplants where desirable combinations of organs can be identified and transplanted in bulk into a host. However, upon analyzing the results, we have detected various interactions between organs that may help guide an approach that considered multiple transplants:

- **Organ dependencies** occur when an organ requires for another organ to be present in the host to work properly. For instance, a spike weapon must be mounted on a hull belonging to the body of a boss and cannot appear by itself. In other words, a spike weapon organ depends on the existence of a hull organ to be able to be included in the boss.
- **Organ incompatibilities** happen when an organ should not appear in the host under any circumstances. For instance, consider attaching a black hole organ to a hull belonging to the boss. The black hole organ destroys everything it touches, so it would instantly end the boss without triggering the end condition for the game, since the battle is considered as completed only when the player is the one responsible for ending the boss. This would actively block player progress, which is undesirable for the game.
- **Organ synergies** are found when the functionality of an organ benefits from the existence of another organ in the host. For instance, adding one or more weapons to a hull where a weak spot is located protects the boss from the player, building a more interesting challenge.
- **Organ discordances** take place when the functionality of an organ is hindered by the existence of another organ in the host. For instance, annexing a hull with a mobile arm to another hull with a laser may cause the laser beam to be intermittently blocked, decreasing its attack capabilities.

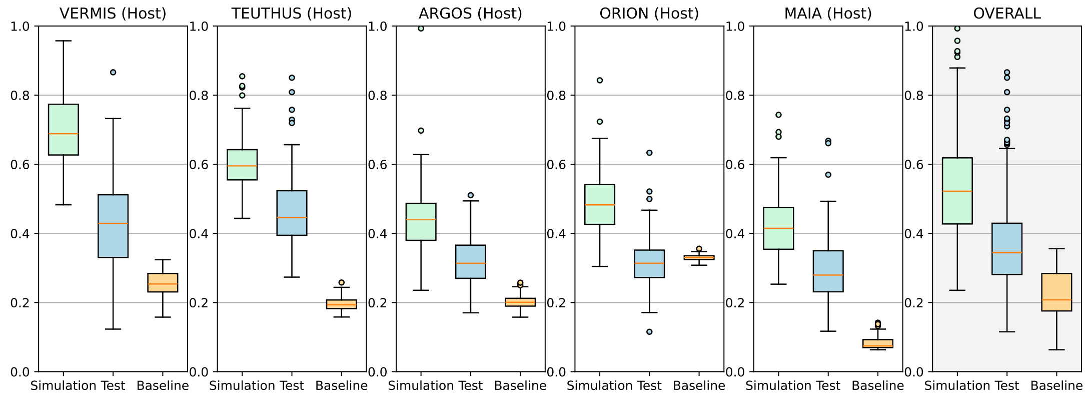

Figure 10: Results of our IMHOTEP approaches (simulation-based and test-based) and the baseline for the quality measurement ($Q_{Duration}$).

So far, the literature on software transplantation does not tackle or even identify interactions between organs. Studying these organ interactions is a promising line of work to advance the concept of transplantation both in video games and in the general software domain.

Concerning the focus group (see the bottom part of Fig. 9), we conducted a survey with two developers from Entalto[8] and two from Kraken Empire[9]. All of them are seasoned video game developers who devote most of their working hours to the software behind the games. We openly asked about their content preferences, presenting them with generated content whose origin (that is to say, generated by either IMHOTEP or by the baseline) was masked, and there was unanimous preference for IMHOTEP-generated content.

Furthermore, they indicated that they would use it as primary content for the game rather than secondary. Primary content is that which conforms an essential part of the experience of the players, while secondary content is that which does not directly affect the main experience but contributes to creating the atmosphere of the game (for instance, distant decoration). Until now, PCG works generated results used as secondary content. In that sense, the possibility of using generated content as primary content represents an advancement in PCG. Developers justify this choice by arguing that the content generated by IMHOTEP aligns better with the vision of the game, whereas the baseline-generated content feels more random in purpose even when reusing content that was created within the context and vision of the game by the developers.

7 THREATS TO VALIDITY

To tackle possible threats to the validity of our work, we follow the classification suggested by De Oliveira *et al.* [6].

Conclusion Validity. To minimize *not accounting for random variation*, we run a total of 645 transplants for each host on each variation of IMHOTEP and the baseline. In order to address the *lack of good descriptive statistics*, we present the standard deviation, min-max range and a box-plot from the results of the experiments realized. We also applied statistical significance tests (the Quade test and Holm's post-hoc analysis) and effect size measurements ($\hat{A}_{12}$ and Cliff's Delta) following accepted guidelines [3]. We tackled the *lack of a meaningful comparison baseline* by comparing IMHOTEP to a recent PCG approach as a baseline.

Internal Validity. We provide the source code and the artifacts used in our experiments to allow for its reproduction as suggested to avoid the *lack of discussion on code instrumentation*. We handled the *lack of real problem instances* by selecting a commercial video game as the case study for the evaluation. Likewise, the problem artifacts (donor, organs and hosts) were directly obtained from the video game developers and the documentation itself.

Construct Validity. To prevent the *lack of assessing the validity of cost measures*, we made a fair comparison between the two variants of our approach and the baseline. Furthermore, we used a metric for the evaluation that has been widely adopted and *validated* by the research community [9].

External Validity. To mitigate the lack of *generalization* threat, we designed our approach to be generic and applicable not only to our industrial case study but also for generating content in other different video games. To avoid the *lack of a clear object selection strategy* in our experiment, we have selected the instances from a commercial video game, which represents real-world instances. In fact, IMHOTEP can be applied where NPCs are available. NPCs are usually available in popular game genres such as car games (rival drivers), FPS games (bots), or RTS games (rival generals). For those cases were there is no NPC, the developers should ponder the trade-off of the cost of developing the NPCs and the benefits of generating content with our approach. Our approach should be replicated with other video games before assuring its generalization.

8 RELATED WORK

This work generates content in video games leveraging software transplantation. In this section, we discuss: (1) work that tackles automated software transplantation; and (2) work that tackles video game content generation.

[8] https://www.entaltostudios.com/
[9] https://www.krakenempire.com/

8.1 Automated Software Transplantation

Miles *et al.* [33] and Petke *et al.* [38] proposed the first approaches that transplant software code in a same program (assuming that different versions of the programs are considered a same program). When transplanting within the same program, there is no need for adapters: alterations in organ or host to adapt the organ in the host. Sidiroglou-Douskos *et al.* [45] proposed a technique that divides the donor program by specific functionality, each piece is called a 'shard'. The approach insert the shard into the host without modifications, that is, the work from Sidiroglou-Douskos does not use adapters either.

On the other hand, Maras *et al.* [31] proposed a three step general approach, without implementing it, which applies feature localization to identify the organ; then code analysis and adaptation, and finally feature integration. Wang *et al.* [55] instead of using feature localization, takes as inputs the desired type signature of the organ and a natural language description of its functionality. With that, the approach called Hunter uses any existing code search engine to search for a method to transplant in a database of software repositories. Further, Hunter generates adapter functions to transform the types from the desired type signature into the type signatures of the candidate functions.

Allamanis *et al.*'s SMARTPASTE [1] presents a different strategy to adapt the organ into the host. SMARTPASTE takes the organ and replace variable names with holes, the approach using a deep neural network fills the holes. Allamanis *et al.* [1] use Gated Graph Neural Networks [24] to predict the correct variable name in an expression.

In 2018, Lu *et al.* [27] introduced program splicing, a framework to automate the process of copy, paste, and organ modification. In their approach, unlike Allamanis *et al.*, who puts holes into the organ, the host is provided with a draft of the code with holes, or natural language comments. Similarily to, Wang *et al.*, program splicing looks into a database of programs to identify a relevant code to the current transplant task. Finally, the approach selects the more suitable result found to fill the holes in the draft.

μSCALPEL [4] is an automatic code transplant tool that uses genetic programming and testing to transplant code from one program to another. μSCALPEL uses test cases to define and maintain functionalities, small changes are made to the transplanted code, and code that does not aid in passing tests can be discarded, reducing the code to its minimal functioning form. τSCALPEL [32] achieves the transplantation between different programs and programming languages.

We have seen so far that Automated Software Transplantation transplants within the same platform. However, Kwon *et al.* propose CPR [21] that transplants an entire program between different platforms. CPR realizes software transplantation by synthesizing a platform independent program from a platform dependent program. To synthesis the platform independent program, CPR uses PIEtrace [22] to construct a set of trace programs, which captures the control flow path and the data dependencies observed during a concrete execution, and replaces all the platform dependencies with the concrete values that it observed during the concrete execution. Finally, CPR merges all these trace programs together to handle any input, by replacing the concrete values observed during the executions, with input variables.

To the best of our knowledge our is the first proposal addressing automated software transplantation in the field of content generation for video games. Our proposal allows the transplantation between different types of content. We have demonstrated that in this context the simulations yield superior outcomes compared to the test-based objective function that previously attained the most favourable results in traditional software engineering transplantation (μSCALPEL).

8.2 Procedural Content Generation

Procedural Content Generation (PCG) refers to the automation or semi-automation of the generation of content in video games [20]. The types of content generated by PCG are diverse, such as vegetation [34], sound [39], terrain [16], Non-Playable Characters [54], dungeons [53], puzzles [13], and even the rules of a game [10]. PCG is a large field spanning many algorithms [61], which can be grouped in three main categories according to the survey of PCG techniques by Barriga et al. [5]: Traditional methods [17] that generate content under a procedure without evaluation; Machine Learning methods (PCGML) [26, 49, 50] that train models to generate new content; and Search-Based methods (SBPCG) [20, 51] that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

Our work generates content of the NPC type. In the context of NPC generation using SBPCG, Ripamonti *et al.* [42] developed a novel approach to generate monsters adapted to players, considering the monster with more death rate the preferred by the player. To evaluate the monsters, they recreated an environment with the main aspects from a MMORPG [10] game. Pereira *et al.* [37] and later extended by Viana *et al.* [54] seek for generating enemies that meet a difficulty criteria. Pereira *et al.* and Viana *et al.* use the same research academic game in their experimental designs. Blasco *et al.* [7] focusses on generating spaceship enemies that are comparable to the ones manually created by developers. To generate spaceships, Gallota *et al.* [18] used a combination of Lindenmayer systems [25] and evolutionary algorithm. Gallota *et al.* as well as Blasco *et al.* use a commercial video game in their evaluation.

In the context of ML, to the best of our knowledge there is a gap in the generation of NPC. ML research focus on other aspects of video games, such AI [8] or graphical aesthetics [23].

The motivation of our work comes from the limitations that we detected in previous work. Previous work focused on speeding up development time. However, the influence of the developers on the generated content was limited. The generated content depended on randomness resulting on generated content not aligned with the intention of the developers. As a result, the generated content was either not used or used as secondary content.

Our work is the first approach that tackles automated software transplantation if the field of video games. Furthermore, our proposal allows the transplantation between different types of content. More precisely, in this work, we transplant organs from a scenarios to an NPCs.

9 CONCLUSION

Procedural Content Generation (PCG) aims for the (semi) automatic generation of new content within video games. Typically, current

[10] Massive Multiplayer Online Role-Playing Games

PCG methods are operated by developers providing initial content to an algorithm, which then generates additional content. However, the developers have limited control over the generation process, which results in the generated content being used as secondary content, or not been used at all.

In this study, we empower developers by introducing the transplantation metaphor into PCG for the first time. Our approach allows game developers to choose an organ from a donor and a host that will receive the organ. Through our approach, we aim to search for a suitable solution to integrate the organ into the host. To guide our search, we propose two distinct objective functions: one based on test case following conventional software transplantation method, and another novel objective function based on simulations, proposed here.

Our proposal has been empirically assessed by using the commercial video game Kromaia. To evaluate our approach, we have transplanted a total of 129 distinct organs from the scenarios of Kromaia into 5 video game bosses, which serve as hosts. This transplantation process has resulted in the creation of 1290 new video game bosses. We then compare the outcomes of our approach (the two variants) with a PCG baseline.

Our $S_{Imhotep}$ produces results that are 1.5 times superior to those of the $T_{Imhotep}$ and 2.5 times superior to the baseline. The statistical analysis confirms the significance of these differences and highlights the substantial magnitude of improvement. Furthermore, a focus group with game developers indicated that first, they would use the generated content by our approach, and secondly, that they would use it as primary content for the game rather than secondary.

Our results demonstrate that we have successfully generated new content through transplantation. Not only that alone, we have accomplish 645 transplantation in total for a commercial video game. Furthermore, our work achieves transplantation between different types of content which results in expanding the library of organs available. This can inspire researchers and developers to explore the use of different types of content for creating new content automatically. In addition, we have presented a novel objective function to guide search software transplantation, which has obtained better results than traditional one. This novel search guidance opens a door in the field of software transplantation.

ACKNOWLEDGMENTS

Work supported by National R+D+i Plan PID2021-128695OB-100, José Castillejo CAS18/00272, ERC grant 741278.

REFERENCES

[1] Miltiadis Allamanis and Marc Brockschmidt. 2017. Smartpaste: Learning to adapt source code. *arXiv preprint arXiv:1705.07867* (2017).

[2] Andrea Arcuri and Lionel C. Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Softw. Test. Verification Reliab.* 24, 3 (2014), 219–250. https://doi.org/10.1002/STVR.1486

[3] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18 (2013), 594–623.

[4] 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.

[5] 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

[6] Márcio Barros and Arilo Neto. 2011. Threats to Validity in Search-based Software Engineering Empirical Studies. *RelaTe-DIA* 5 (01 2011).

[7] Daniel Blasco, Jaime Font, Mar Zamorano, and Carlos Cetina. 2021. An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering. *Journal of Systems and Software* 171 (2021), 110804.

[8] Michele Brocchini, Marco Mameli, Emanuele Balloni, Laura Della Sciucca, Luca Rossi, Marina Paolanti, Emanuele Frontoni, and Primo Zingaretti. 2022. MONStEr: A Deep Learning-Based System for the Automatic Generation of Gaming Assets. In *International Conference on Image Analysis and Processing*. Springer, 280–290.

[9] Cameron Browne and Frederic Maire. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010), 1–16.

[10] Cameron Bolitho Browne. 2008. *Automatic generation and evaluation of recombination games*. Ph. D. Dissertation. Queensland University of Technology.

[11] 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.

[12] Norman Cliff. 1996. *Ordinal methods for behavioral data analysis.* Lawrence Erlbaum Associates, Inc. 212 pages. https://doi.org/10.4324/9781315806730

[13] Barbara De Kegel and Mads Haahr. 2019. Procedural puzzle generation: A survey. *IEEE Transactions on Games* 12, 1 (2019), 21–40.

[14] África Domingo, Jorge Echeverría, Oscar Pastor, and Carlos Cetina. 2020. Evaluating the Benefits of Model-Driven Development: Empirical Evaluation Paper. In *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32*. Springer, 353–367.

[15] Melina Farshbafnadi, Sepideh Razi, and Nima Rezaei. 2023. Chapter 7 - Transplantation. In *Clinical Immunology*, Nima Rezaei (Ed.). Academic Press, 599–674. https://doi.org/10.1016/B978-0-12-818006-8.00008-6

[16] Miguel Frade, Francisco Fernandéz de Vega, Carlos Cotta, et al. 2009. Breeding terrains with genetic terrain programming: the evolution of terrain generators. *International Journal of Computer Games Technology* 2009 (2009).

[17] Jonas Freiknecht and Wolfgang Effelsberg. 2017. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction* 1, 4 (2017), 27.

[18] 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.

[19] Robert J. Grissom and John J. Kim. 2005. *Effect sizes for research: A broad practical approach*. Lawrence Erlbaum Associates Publishers, New Jersey, United States.

[20] 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.

[21] Yonghwi Kwon, Weihang Wang, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu. 2017. Cpr: cross platform binary code reuse via platform independent trace program. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 158–169.

[22] Yonghwi Kwon, Xiangyu Zhang, and Dongyan Xu. 2013. Pietrace: Platform independent executable trace. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 48–58.

[23] Ruizhe Li, Masanori Nakayama, and Issei Fujishiro. 2020. Automatic Generation of 3D Natural Anime-like Non-Player Characters with Machine Learning. In *2020 International Conference on Cyberworlds (CW)*. IEEE, 110–116.

[24] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[25] Aristid Lindenmayer. 1968. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology* 18, 3 (1968), 280–299.

[26] 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.

[27] Yanxin Lu, Swarat Chaudhuri, Chris Jermaine, and David Melski. 2018. Program splicing. In *Proceedings of the 40th International Conference on Software Engineering*. 338–349.

[28] Keza MacDonald. 2019. Tough Love: On Dark Souls' Difficulty. https://www.eurogamer.net/tough-love-on-dark-souls-difficulty. Accessed: 01/02/24.

[29] 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.

[30] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[31] Josip Maras, Maja Štula, and Ivica Crnković. 2015. Towards specifying pragmatic software reuse. In *Proceedings of the 2015 European Conference on Software*

Architecture Workshops. 1–4.

[32] Alexandru Marginean. 2021. *Automated Software Transplantation*. Ph. D. Dissertation. UCL (University College London).

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

[34] Carlos Mora, Sandra Jardim, and Jorge Valente. 2021. Flora Generation and Evolution Algorithm for Virtual Environments. In *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 1–6.

[35] Zachery Oliver. 2013. DYNASTY WARRIORS = DUMB FUN. https://theologygaming.com/dynasty-warriors-dumb-fun/. Accessed: 01/02/24.

[36] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. 2018. How is video game development different from software development in open source?. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 392–402.

[37] Leonardo T Pereira, Breno MF Viana, and Claudio FM Toledo. 2021. Procedural enemy generation through parallel evolutionary algorithm. In *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 126–135.

[38] 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.

[39] David Plans and Davide Morelli. 2012. Experience-driven procedural music generation for games. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 3 (2012), 192–198.

[40] 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.

[41] Brittany Reid, Christoph Treude, and Markus Wagner. 2020. Optimising the fit of stack overflow code snippets into existing code. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1945–1953.

[42] Laura Anna Ripamonti, Federico Distefano, Marco Trubian, Dario Maggiorini, and Davide Gadia. 2021. DRAGON: diversity regulated adaptive generator online. *Multimedia Tools and Applications* 80, 26 (2021), 34933–34969.

[43] Jeanine Romano, Jeffrey D. Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?. In *annual meeting of the Florida Association of Institutional Research*. 1–3.

[44] Piotr Rykała. 2020. The growth of the gaming industry in the context of creative industries. *Biblioteka Regionalisty* 20 (2020), 124–136.

[45] Stelios Sidiroglou-Douskos, Eli Davis, and Martin Rinard. 2015. Horizontal code transfer via program fracture and recombination. (2015).

[46] Stelios Sidiroglou-Douskos, Eric Lahtinen, Anthony Eden, Fan Long, and Martin Rinard. 2017. CodeCarbonCopy. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 95–105.

[47] Stelios Sidiroglou-Douskos, Eric Lahtinen, and Martin Rinard. 2014. Automatic error elimination by multi-application code transfer. (2014).

[48] SlashData. [n. d.]. State of the Developer Nation 23rd Edition. ([n. d.]). https://slashdata-website-cms.s3.amazonaws.com/sample_reports/dsIe6JlZge_KsHWt.pdf [Online; accessed 18-December-2023].

[49] Konstantinos Souchleris, George K Sidiropoulos, and George A Papakostas. 2023. Reinforcement Learning in Game Industry—Review, Prospects and Challenges. *Applied Sciences* 13, 4 (2023), 2443.

[50] 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

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

[52] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101 arXiv:http://jeb.sagepub.com/content/25/2/101.full.pdf+html

[53] Breno MF Viana and Selan R dos Santos. 2019. A survey of procedural dungeon generation. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 29–38.

[54] Breno MF Viana, Leonardo T Pereira, and Claudio FM Toledo. 2022. Illuminating the space of enemies through map-elites. In *2022 IEEE Conference on Games (CoG)*. IEEE, 17–24.

[55] Yuepeng Wang, Yu Feng, Ruben Martins, Arati Kaushik, Isil Dillig, and Steven P Reiss. 2016. Hunter: next-generation code reuse for java. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 1028–1032.

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

[57] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 364–374.

[58] Tom Wijman. 2021. The Games Market and Beyond in 2021: The Year in Numbers. https://newzoo.com/resources/blog/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming. Accessed: 01/12/23.

[59] Tom Wijman. 2022. The Games Market in 2022: The Year in Numbers. https://newzoo.com/resources/blog/the-games-market-in-2022-the-year-in-numbers. Accessed: 01/12/23.

[60] Wei Yang, Deguang Kong, Tao Xie, and Carl A Gunter. 2017. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. 288–302.

[61] Georgios N Yannakakis and Julian Togelius. 2018. *Artificial intelligence and games*. Vol. 2. Springer.

[62] Tianyi Zhang and Miryung Kim. 2017. Automated transplantation and differential testing for clones. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 665–676.

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