Video Game Procedural Content Generation Through Software Transplantation

Mar Zamorano*†, Daniel Blasco†, Carlos Cetina‡*,and Federica Sarro*
*University College London, London, UK
†Universidad San Jorge, Zaragoza, Spain
‡Universitat Politècnica de València, Valencia, Spain
maria.lopez.20@ucl.ac.uk, dblasco@usj.es, cetina@upv.es, f.sarro@ucl.ac.uk

***Abstract*—Software transplantation generates new piece of software by reusing existing parts from a given piece of software (i.e., host) to enhance other parts of a same or different software (i.e., donor). In this paper, we argue that software transplantation can be used for automatically producing video game content. We propose the first search-based algorithm for procedural content transplantation and empirically evaluating it in an industrial case study in collaboration with the developers of the commercial video game Kromaia. Specifically, our proposed approach, dubbed IMHOTEP, enables developers to choose *what* video-game content to transplant and *where*, and automatically searches for an appropriate solution to integrate the organ into the host. Such a search is performed by using an evolutionary algorithm guided by a simulation-based fitness function, which is novel w.r.t previous transplantation work generally guided by test-suite compliance.**

We empirically evaluate the effectiveness of IMHOTEP to transplant procedural content, specifically non-playable characters, for the commercial video game Kromaia and benchmarked it against a state-of-the-art approach in search-based procedural content generation, as well as a variant of IMHOTEP itself guided by a test-suite-based fitness function.

Using IMHOTEP, Kromaia developers were able to transplant 129 distinct organs taken from the game's scenarios into five different hosts, thus generating a total of 645 new transplanted non-playable characters for this game. Moreover, we found that the game content generated by using IMHOTEP was 1.5 times superior to the one obtained by using its test-suite-based variant, and 2.5 times superior than the one generated by the state-of-the-art benchmark. Finally, a focus group with game developers indicated their satisfaction with the content generated by IMHOTEP and their willingness to use it for game development.

***Index Terms*—Automated Software Transplantation, Auto-transplantation, Procedural Content Generation, Search-Based Software Engineering, Model-Driven Engineering**

I. INTRODUCTION

The video games industry grows significantly every year [1]. In 2019, it became the largest entertainment industry in terms of revenue after surpassing the combined revenues of the movie and music industries [2]. In 2021, video games generated revenues of $180.3 billion [3], and in 2022, the estimated revenues were of $184.4 billion [4]. Overall, the sum of revenues generated from 2020 to 2022 was almost $43 billion higher than those originally forecasted.

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 formed 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%) [5]. 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 non-playable characters (NPCs) 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 such as Unity [6] and Unreal [7]. These offer 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 to develop the game content (e.g., scenarios, NPCs, items such as weapons) remains.

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 [8]. Current PCG approaches work as follows: Developers provide initial content (usually human-generated) into an algorithm. Afterwards, the algorithm (which could be a traditional, machine learning, or search-based PCG method) generates new content. However, thus far only a few traditional methods have been used to randomly generate vegetation [9] in Unreal and Unity.

In this paper, we propose a new angle to tackle video games content generation inspired by transplantation techniques [10], 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 [11]. 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) [10]. Software transplantation has been successful for different tasks: program repair [12], [13], testing [14], security [15], and functionality improvements [16].

Our PCT proposal introduces for the first time the transplantation metaphor for video-games. 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 a 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 PCG approaches that are solely based on random generation, leading to results that are closer to developers' expectations.

To show the viability of procedural content generation via transplantation we designed and realised the first ever search-based transplantation algorithm for video-game procedural content, dubbed IMHOTEP[1], and evaluated its effectiveness in generating non-playable characters (NPCs) for the commercial video-game Kromaia. We rely on a search-based algorithm as the process of transplantation involves connecting the boundaries of the donor organ with those of the host, and with multiple potential connection points available from the donor, the search space created by the combination of boundaries and host is simply too vast to be thoroughly explored through brute force methods. Moreover, search-based approaches have been successfully applied for traditional software transplantation [10]. Moreover, we propose the use of video game simulations ($S_{Imhotep}$) to guide the search, based on the intuition that it is possible to harness video games' NPCs to run simulations that provide data to asses the transplantation.[2]

To evaluate our proposal we have carried out an industrial case study in collaboration with the developers of the commercial video game Kromaia[3]. Kromaia has been released on PC, PlayStation, and translated to eight different languages. In particular, in the Kromaia case study, we were able to assess the effectiveness of IMHOTEP to transplant 129 different *organs* extracted from the *scenarios* of Kromaia into five of its NPCs *bosses* that act as hosts, generating new video game *bosses*, for a total of 645 successful transplants. This is higher than previous work in the literature, which achieved at most 327 successful transplants [17].

We compare the quality of the 645 bosses generated by using IMHOTEP to the same number of bosses generate by using a search-based PCG approach from the literature [18], which is the most relevant state-of-the-art of a comparable nature, and those generated by a variant of IMHOTEP that uses test-suite as objective function (namely, $T_{Imhotep}$), in line with the traditional software transplantation literature. To perform the comparison, we rely on the concept of game quality and its automated measurement, which is widely accepted in practice [19].

The results show that, out of the three approaches, the content generated through the IMHOTEP obtains the best results: It yields 1.5x better results than $T_{Imhotep}$ and 2.5x better results than baseline. The statistical analysis shows that the differences are statistically significant, and the magnitude of improvement is always 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 current SBPCG in an industrial setting. In summary:

- Our results show that procedural content generation through transplant (*i.e.* PCT) has significantly outperformed classic content generation in the evaluation of this work, opening a new road towards tackling content generation.
- Our transplantation approach has produced the highest number of successful transplants, to date - almost double than those found in previous work. Moreover, the transplants are carried out in an real-world industrial context in contrast to the academic context of other work.
- Our work returns control to the hands of the developers through organ selection. The generated content is more in line with the intent of developers, as discussed in the focus group.
- 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.
- Our analysis of the results reveals interactions between organs that are a promising line of research to advance the field of software transplants.

For replicability, reproducibility and extension of our work, we made IMHOTEP's source code and the data of our study publicly available at https://github.com/SOLAR-group/IMHOTEP.

II. BACKGROUND

A. 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 [20], [21]. 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 [20], [21]. Today, most video games are developed using

[1]Our approach is named after IMHOTEP, who is considered by many to have written the Edwin Smith Papyrus (the oldest known manual of surgery).

[2]In fact, 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.

[3]See the official PlayStation trailer to learn more about Kromaia: https://youtu.be/EhsejJBp8Go

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 [22]). One key artefact of game engines are software models. Unreal proposes its own modeling language (Unreal Blueprints) [23], Unity proposes Unity Visual Scripting [24], and a recent survey [25] 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.

B. Kromaia Case Study

In this subsection we give an overview of Kromaia, the commercial video game used in our industrial case study.

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 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 boss is the NPCs target content that we aim to automatically create via PCT. Bosses can be built either using C++ code or software models. The top part of Figure 1 depicts a boss fight scenario where the player-controlled ship (item A in the figure) is battling the NPC 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 Figure 1 illustrates the two possible development approaches for the Serpent boss (model-driven Vs. code-centric).

Developers can mix both technologies by developing different parts of the boss using one or the other approach indistinctly, but 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 [5], but also counting game designers, artists, UI designers, and QA engineers within their ranks - possibly favours the use of software models over code thanks to the higher abstraction level of the former (combined with their detachment from more technical implementation details) which empowers less tech-focused roles to embrace a more active participation in development tasks. Also, previous work [26] showed that video game developers make fewer mistakes and are more efficient when working with models rather than code.

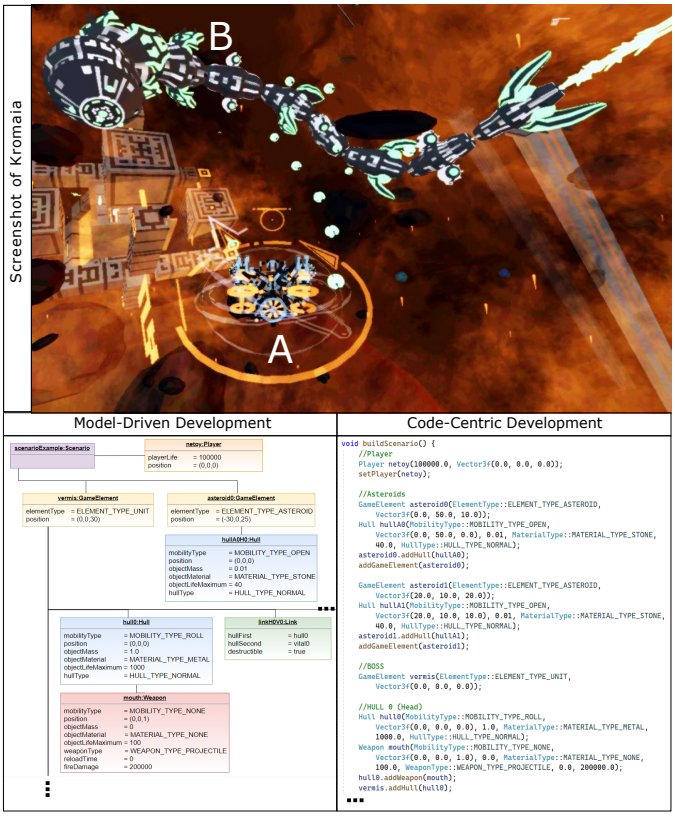

Fig. 1: Kromaia: Model-Driven vs. Code-Centric Development

In 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 defenses; and the movement behaviours associated with 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 all types of video game content, such as bosses, enemies, or environmental elements. The models are created using SDML and interpreted at run-time 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 [27]. More information on the SDML model can be found on-line at https://youtu.be/Vp3Zt4qXkoY. The use of software models makes Kromaia a suitable video game for our study.

III. Our Proposal: Imhotep

This section explains how Imhotep makes use of evolutionary computation [28] and software models to transplant organs within video game content in order to create new content (*i.e.*, Kromaia bosses in our case study). To facilitate the comprehension, we also provide the reader with an example of transplantation for a simplified version of a Kromaia 'boss'

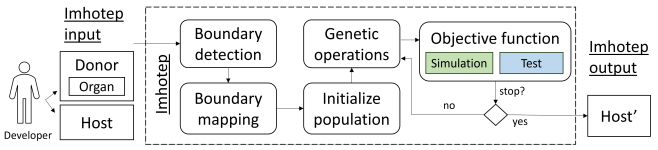

Fig. 2: Overview of IMHOTEP, our proposal for PCT.

inspired by the 'Serpent' boss shown in Figure 1 with letter B. Given the popularity of software models for video-game development (see Section II), we designed IMHOTEP to work with models. Although our running example uses the SDML models of Kromaia, our approach is generic and can be used with other modelling languages because it exploits the idea of boundaries between model elements.

Figure 2 shows an overview of IMHOTEP. On the left there is the input to our approach, namely the organ to be transplanted from the donor and the host where the organ will be transplanted to. 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 to a random point. Genetic operations generate potential solutions for transplantation, while the objective function assesses 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 transplants between organ and host. Next, we describe each step of IMHOTEP.

A. Input selection

IMHOTEP allows 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 meta-model, and the corresponding concrete syntax of the model (see Figure 3 *Metamodel*) from Kromaia. In such model 'Hulls' serve as the structural framework that define the anatomical composition of the models. For example, the boss presented in Figure 1 (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'.

In our example, the source donor model is a simplified version of the original Kromaia 'boss' 'Serpent'. Figure 3 *Input* **Donor** shows the graphical representation of the donor's model . It also shows with dashed lines the elements selected as organ. The host is a model of a regular enemy that could appear in Kromaia. Figure 3 *Input* **Host** shows the graphical representation of the host model.

B. Boundary detection

To transplant an organ into a host we need to find a way to connect them. To this end we exploit the boundaries between the model elements of the organ and the host. The study of boundaries between elements in software models has been ongoing for over ten years, with the aim of managing variability within models [29], [30]. 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 Figure 3 *Metamodel*, the Source and Target meta-relationships are the boundaries between the model elements of the models conforming to that metamodel. In other model 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 our 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. Figure 3 *Boundary detection* **Donor** shows the donor, differentiating each element of the model with a letter from A to S, and the selected organ (namely, H, I, J, K, N, O, P, Q) with its boundaries (which are b11 for the H element; b16 for the K element, and b25 for the Q element). While, the host boundaries are all the points where its model elements connect. Figure 3 *Boundary detection* **Host** shows all the boundaries of the host of our running example: The host has a total of 19 boundaries identified by a tag from ba to bs.

C. Boundary mapping

In the boundary mapping step, IMHOTEP determines a mapping between the organ and the host boundaries. 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.

The table on the Figure 3 *Boundary mapping* 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', and 'Weak Point'. The boundaries b16 and b25 are both 'Hulls' and they can connect with any 'Link'.

D. 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 Figure 3 *Encoding*). In the binary vector, each element from the model is a position in 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

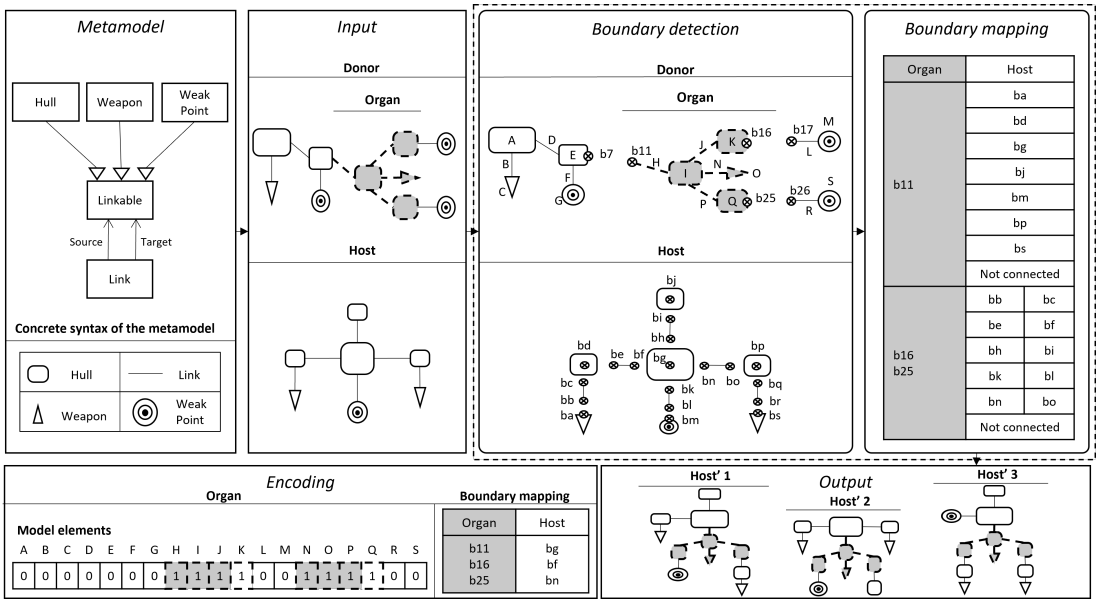

Fig. 3: Overview of IMHOTEP on a running example.

host and the organ placed in a random position (*i.e.* a random mapping between the organ boundaries and the compatible organ boundaries).

E. Genetic operators

IMHOTEP uses traditional genetic operators (namely, selection, crossover, and mutation) to generate new individuals (*i.e.* candidate solutions). Specifically, we use the ranking selection, which ranks the individuals based on the objective function and retains the top ones in the current population. We use a single, random, cut-point crossover, which selects two parent solutions at random, and determines a cut point uniformly at random 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 offspring is mutated by changing any value of the encoding uniformly at random with a certain probability. Figure 3 *Output* shows an 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.

F. Objective function

Our work proposes to harness video games' NPCs to run simulations that provide data to assess the transplants (*i.e.* to compute the value of the objective function assessing the quality of each transplant). Specifically, we propose to use the content generated via transplantation (each individual in the population) into a simulation of the video game. Such a 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 our case study, 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. In fact the simulations leverage mainly the NPCs (but also more video game elements, such as scenarios or items like weapons or powerups),which are usually developed anyway during for most types of video games. In other words, NPCs are integral components of most video game genres such as First-Person Shooter, Real-Time Strategy, or Racing Games. This use of simulations has two advantages: it makes the use of simulations cheaper (*i.e.* it does not involve additional development costs) and it facilitates fidelity to the video game compared to ad hoc development.

In our case study, IMHOTEP compute the objective function value for each individual in the population, through a simulation of a game battle between the boss generated via transplantation (*i.e.* the candidate solution, also referred to as Host') and an NPC spaceship[4]. Since all these elements, as well as the scenarios and items such as weapons or powerups already belong to the game itself, no extra development is needed to run the simulation.

Once a simulation is executed, we need a way to quantify its *quality*. One 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 when creating a game. In fact, different developers may have different interpretations, also depending on the intended users of a given video game. For some, 'fun' is achieved with a difficult game that is very rewarding when

[4]Note that from now we can refer to the simulation-based version of IMHOTEP as $S_{Imhotep}$, to differentiate it from a more traditional objective function based on test-suite-compliance (referred as to $T_{Imhotep}$herein).

progress is made (e.g., Dark Souls [31]), while for others, 'fun' is achieved by effortlessly killing enemies (e.g., Dynasty Warriors [32]). Therefore, we argue that such an intent is key for the evaluation of new generated content. Hence, to evaluate the quality of the candidate solutions generated by IMHOTEP we take 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}), which are commonly used metrics in the literature. Specifically, we compute $F_{Victory}$ and F_{Health} according to Blasco *et al.* [27], 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) ($V_{Optimal}$):

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

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):

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

The $F_{Victory}$ and F_{Health} criteria are combined (*i.e.* averaged) in the objective function $F_{Overall}$ which guides the evolutionary search, as follows:

$$F_{Overall} = min\left(Validity, \sum_{i=1}^{N} F_i / N\right) \quad (3)$$

where *Validity* is a crucial part to take into account the validity of newly generated models by using a run-time interpreter which is already part of the game. In fact, such validation step is needed to discard models with inconsistencies. When a model is stated as non-valid by the interpreter the value of Validity will be 0. $F_{Overall}$ value is the minimum between Validity and the average value of $F_{Victory}$ and F_{Health}, thus it can assume a value in [0, 1].

IV. Experimental Design

In this section we explain the design of the experiments we perform to empirically evaluate IMHOTEP by using the commercial video game Kromaia. We present the research questions that we aim to answer, the evaluation method, and the implementation details.

A. Research Questions

IMHOTEP proposes a new angle for video game procedural content generation, and for this reason we need to assess how it compares to the established practice for PCG . This motivates our first research question:

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

To answer RQ1, we had to identify the most relevant and close work in the PCG literature. We identify the work by Gallota *et al.* [18] as the most representative benchmark for our study. Indeed, Gallota *et al.* 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 of the same nature of IMHOTEP (*i.e.* it uses evolutionary computation), (2) it is specific for spaceships that can play the role of bosses which is comparable to the content of our case study, and (3) it achieves the best state-of-the-art results for this type of content.

Moreover, since we are the first to propose the use of a simulation-based objective function to guide the search for transplantation it is natural to compare it with the established practice in the software transplantation field, which instead relies on the use of a test suite to guide the transplantation. This motivates our second research question:

RQ$_2$: *To what extend using a simulation-based objective function to guide the transplantation is more effective than a test-based one for* IMHOTEP*?*

To answer RQ2 we empirically compare IMHOTEP guided by the simulation-based objective function described in Section III-F (which we refer to as $S_{Imhotep}$) with a test-based variant of IMHOTEP (which we refer to as $T_{Imhotep}$). Specifically, $T_{Imhotep}$ uses an objective function based on the number of test cases that are passed by the transplanted software. The reason for considering this variant is that in traditional software transplantation the best results have been achieved by using the test suite as the objective function. In order, to run $T_{Imhotep}$, the Kromaia's developers provided us with a test suite relevant to the game, consisting of a total of 243 tests selected based on their domain knowledge. Therefore the value of $T_{Imhotep}$'s objective function was computed by running each individual through the 243 tests, recording the number of tests passed and normalizing this value in a scale of [0, 1]. An individual which passes the 243 tests will obtain an objective function score of 1, on the contrary if it does not pass any test it will obtain an objective function score of 0. As in $S_{Imhotep}$, each individual also needs to constitute a valid boss (i.e., solution), receiving a score of 0 if it does not represent a valid one according to the run-time interpreter (see Section III-F).

B. Methodology

Figure 4 provides an overview of the process we followed to empirically assess IMHOTEP and answer RQs 1 and 2 for the Kromaia's case study. The top (white background) part shows the assets of the game itself (content) and the game development (test suite) that are used by the approaches. The middle (grey background) part shows inputs and outputs for each of the approaches compared herein. The bottom (white background) part shows the evaluation criteria used to assess the results.

C. Algorithms' Settings

As described in Section III, developers need to select host and donors as input for IMHOTEP. In our empirical study, Kromaia's developers identified as hosts five different bosses (i.e., Vermis,

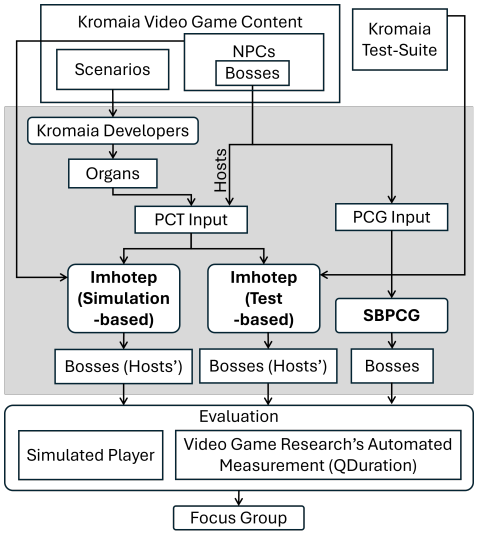

Fig. 4: Overview of the evaluation process.

Teuthus, Argos, Orion, and Maia), which constitute the full set of original bosses from Kromaia. While, as donors, they considered all Kromaia's scenarios and were able to identify 129 organs within them. Each host has more than a thousand model elements, while donor's organs have an average of 255 model elements. Then we run IMHOTEPwith the parameters shown in Table I. We established the stop condition at 2 minutes and 30 seconds, ensuring enough time to obtain suitable solutions.[5] At the end of the evolutionary process, each organ was successfully transplanted to each boss by IMHOTEP, which provided the developers with a total of 645 new bosses (5 hosts * 129 organs) (note we obtain 645 solutions from $S_{Imhotep}$ and 645 from $T_{Imhotep}$).

We executed the SBPCG benchmark by using the parameters presented by the original work and for a total of 129 times for each one of the 5 different hosts, so to obtain the same number of generated individuals (*i.e.* 645).

For all approaches we executed 30 independent runs to account for random variation [33]. Hence, we performed a total of 58,050 independent runs (645*3*30) for our experiment.

The implementation uses the Java(TM) SE Runtime Environment (JDK 1.8) and Java as the programming language. All experiments were run using two PCs with the following specifications: Intel Core i7-8750H, 16GB; and 2x Intel(R) Xeon(R) CPU X5660, 64GB.

1) Evaluation Measures and Statistical Analysis: To compare the solutions provided by the SBPCG benchmark and the two variants of IMHOTEP (*i.e.* $S_{Imhotep}$ and $T_{Imhotep}$), we rely on the concept of game quality and its automated measurement through simulated players. The results by Browne *et al.* demonstrated the validity of this approach, which is now widely accepted in the research community [19]. Therefore,

[5]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 their performance in terms of solution quality on a level playing field.

TABLE I: 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

we need two ingredients to run our experiment: The simulated player and the automated measurement.

The simulated player, developed by the developers of Kromaia, 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 automated measurement is $Q_{Duration}$ which was proven to achieve good results [19]. 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 ($T_{Optimal}$):

$$Q_{Duration} = 1 - \frac{\sum_{d=1}^{Duels} \frac{|T_{Optimal} - T_d|}{T_{Optimal}}}{No.ofDuels} \quad (4)$$

Based on the equation above, the higher the $Q_{Duration}$ of a given approach, the better the solutions it produced.

To measure whether there is any statistical significance difference between the results obtained by the different approaches we perform the Wilcoxon Ranked-Sum test (a.k.a. Mann–Whitney U test) [34] setting the confidence limit, α, at 0.05, and applying the Bonferroni correction (α/K, where K is the number of hypotheses) when multiple hypotheses are tested. We performed a one-sided test since we are interested in knowing if our proposed approach, $S_{Imhotep}$, would be better than the others. In such a case, the one-sided p-value interpretation would be straightforward. Specifically, for RQ1 we test the following null hypothesis: *The distribution of $Q_{Duration}$ values produced by $S_{Imhotep}$ is not better than that produced by the SBPCG benchmark*. If the test rejects the Null Hypothesis, the alternative hypothesis would be accepted: *The*

distribution of $Q_{Duration}$ values produced by $S_{Imhotep}$ is better than that produced by the SBPCG benchmark. Similarly for RQ2 we test the following null hypothesis: *The distribution of $Q_{Duration}$ values produced by $S_{Imhotep}$ is not better than that produced by $T_{Imhotep}$*. If the test rejects the Null Hypothesis, the alternative hypothesis would be accepted: *The distribution of $Q_{Duration}$ values produced by $S_{Imhotep}$ is better than that produced by $T_{Imhotep}$*.

We consider the effect size to assess whether the statistical significance has practical significance [35]. We use the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size measure, as it is recommended to use a standardised measure when not all samples are normally distributed [35], as in our case. $\hat{A}_{12}$ 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: $\hat{A}_{12} = (R_1/m - (m + 1)/2)/n$, 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, values between $[0.0, 0.29]$ and $[0.71, 1.0]$ represent large differences.

TABLE II: RQ1-RQ2. (a) Mean value and standard deviation for $Q_{Duration}$ obtained by each approach per boss and overall. (b) Wilcoxon test and Vargha-Delaney $\hat{A}_{12}$ results obtained by comparing $S_{Imhotep}$ Vs. SBPCG (RQ1) and $S_{Imhotep}$ Vs. $T_{Imhotep}$ (RQ2) per boss and overall. $\hat{A}_{12}$: Large – L.

(a) Mean and standard deviation

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

(b) Wilcoxon / $\hat{A}_{12}$

	RQ1	RQ2
Boss	$p-Value$ / $\hat{A}_{12}$	$p-Value$ / $\hat{A}_{12}$
Argos	$3.25x10^{-23}$ / 0.99 (L)	$1.28x10^{-18}$ / 0.85 (L)
Maia	$3.25x10^{-23}$ / 1.0 (L)	$6.64x10^{-18}$ / 0.85 (L)
Orion	$4.01x10^{-23}$ / 0.98 (L)	$4.95x10^{-22}$ / 0.95 (L)
Teuthus	$3.25x10^{-23}$ / 1.0 (L)	$3.60x10^{-18}$ / 0.87 (L)
Vermis	$3.25x10^{-23}$ / 1.0 (L)	$8.86x10^{-23}$ / 0.95 (L)
Overall	$1.41x10^{-107}$ / 0.98 (L)	$6.58x10^{-93}$ / 0.82 (L)

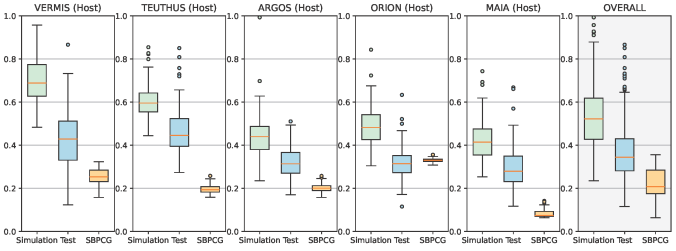

Fig. 5: Results of IMHOTEP ($S_{Imhotep}$ and $T_{Imhotep}$) and the SBPCG benchmark in terms of $Q_{Duration}$.

V. RESULTS

In this section, we present the results obtained by running IMHOTEP and the SBPCG benchmark on Kromaia. Table IIa shows the mean values and standard deviations for $Q_{Duration}$ for each IMHOTEP variant and the SBPCG benchmark, while Figure 5 shows the results in form of boxplots, grouped per host (*i.e.*, the boss of Kromaia used in our experiment, namely Argos, Maia, Orion, Teuthus, and Vermis) and overall. Each boxplot represents the distribution of $Q_{Duration}$ values (obtained as average of 30 independent runs) for each of the 645 solutions obtained from transplantation IMHOTEP ($S_{Imhotep}$ and $T_{Imhotep}$) and SBPCG. We can observe that both variants ($S_{Imhotep}$ and $T_{Imhotep}$) obtained better results than the SBPCG benchmark. Specifically, $S_{Imhotep}$ yielded the best results, followed by $T_{Imhotep}$ and then SBPCG. 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 SBPCG. SBPCG obtained the worst $Q_{Duration}$. Overall, the results reveal that leveraging simulations as objective function pays off in the context of PCT, yielding 1.5x better results than the $T_{Imhotep}$ and 2.5x better results than the SBPCG benchmark.

When analysing whether there is statistical significant differences among the results obtained by $S_{Imhotep}$ and Base. We found that the obtained p-values for $Q_{Duration}$ are always lower than $4.01x10^{-23}$ (see Table IIb). This is below the significance threshold value, so we can comfortably state that $S_{Imhotep}$ provides significant better values for $Q_{Duration}$ with respect to Base. We also observe that all the A12 effect size values are large (see Table IIb), thus confirming the practical magnitude of such a difference. Thus, we conclude that: **Answer to RQ$_1$** $S_{Imhotep}$ performance far surpasses SBPCG with statistically significant difference and large effect size in all cases, exhibiting a remarkable overall enhancement of 250% over SBPCG.

As for the comparison between $S_{Imhotep}$ and $T_{Imhotep}$ (RQ2), we observe that all the p-values achieved when comparing the $Q_{Duration}$ distributions provided by the two IMHOTEP variants are smaller than the significance threshold, thus indicating that the difference in solution quality is statistically significant in favour of $S_{Imhotep}$, and always with a large A12 effect size (see Table IIb). Therefore, we conclude that: **Answer to RQ$_2$** $S_{Imhotep}$ provides significantly better results than $T_{Imhotep}$ in the context of automated content generation through transplantation, with a large effect size in all cases examined. The efficacy of $S_{Imhotep}$ demonstrates a 150% enhancement overall compared to the outcomes of $T_{Imhotep}$.

VI. 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.

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 analysing 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.

So far, the literature on software transplantation does not tackle or even identify interactions between organs. Studying these organ interactions is a line of work to advance the concept of transplantation both in video games and in the general software domain. As part of our evaluation(see Figure 4), we also carried out an informal focus group where we surveyed two developers from Entalto [36] and two developers from Kraken Empire [37]. All of them are seasoned video game developers who devote most of their working hours to realising the software behind different commercial games. We asked them to express anonymously their content preferences, presenting them with the Kromaia's new content produced by either IMHOTEP or by the SBPCG benchmark (note that the source of the generated content was masked to the developers to avoid influencing their answer, *i.e.* they did not how the content was generated). The results showed an unanimous preference for IMHOTEP-generated content. Furthermore, they indicated that they would use it as primary content for the game rather than secondary.[6] Until now, previous PCG work has generated only 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 SBPCG-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. These results have been confirmed in a subsequent larger empirical user-study [38] dedicated to compare content generated via IMHOTEP (more generally referred to as content reuse) and traditional search-based procedural content generation. In fact, this study reveled that developers favour the transplantation approach as they feel that it enhances the underlying content and yields superior outcomes compared to PCG [38]. The developers acknowledged content reuse in form of transplants as a natural progression of the initial original content, while PCG was unfavorably labeled as content that lacked the touch of professional developers.

VII. THREATS TO VALIDITY

To tackle possible threats to the validity of our work, we follow the classification suggested by De Oliveira *et al.* [39].

Conclusion Validity. To minimize *not accounting for random variation*, we run each of the approach (*i.e.* $S_{Imhotep}$, $T_{Imhotep}$and SBPCG) 30 times. Also, we make sure to assess the same number of solutions (*i.e.* 645 new bosses) for each of the approaches, so to make the comparison fair. In order to address the *lack of good descriptive statistics*, we present the standard deviation and a box-plot of the results. We also applied statistical significance tests (Mann-Whitney U) and effect size measurements ($\hat{A}_{12}$) following accepted guidelines [33]. We tackled the *lack of a meaningful comparison baseline* by comparing IMHOTEP to a recent and most relevant Search-Based PCG approach as a benchmark, as detailed in Section IV.

Internal Validity. We provide the source code and the artefacts used in our experiments to allow for reproduction and replication and avoid the *lack of discussion on code instrumentation*. We handled the *lack of real problem instances* by using a commercial video game as the case study for our evaluation and by working closely with its developers in a real-world industrial setting. Likewise, the problem artefacts (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 SBPCG benchmark.

[6]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).

Furthermore, we used a metric for the evaluation that has been widely adopted and *validated* by the research community [19].

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 apply IMHOTEP to another case study, it is necessary an encoding for the transplantation of the content, and leverage the NPCs to obtain the simulation of the objective function. 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.

VIII. Related work

In this section, we discuss work that (1) tackles automated software transplantation and (2) procedural content generation.

Automated Software Transplantation Miles *et al.* [40] and Petke *et al.* [41] proposed the first approaches to transplanting software code across different versions of a same program. This seminal work has inspired follow up research to perform Automated Software Transplantation between different programs [10], or even different programming languages [42] and platforms [43], as summarised below.

Sidiroglou-Douskos *et al.* [44] proposed a technique that divides the donor program by specific functionality, each piece is called a 'shard'. On the other hand, Maras *et al.* [45] 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.* [46] 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. Allamanis *et al.*'s SMARTPASTE [47] takes the organ and replace variable names with holes, the approach using a deep neural network fills the holes. Unlike Allamanis *et al.*, Lu *et al.* [48] introduced program slicing where the host is provided with a draft of the code with holes, or natural language comments. Similarly to Wang *et al.* [46], program slicing looks into a database of programs to identify a relevant code to the current transplant task. Barr *et al.* propose μSCALPEL [10], an automatic code transplant tool that uses genetic programming and testing to transplant code from one program to another. Subsequently, Marginean *et al.* proposes τSCALPEL [42] to achieve the transplantation between different programs and programming languages. 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 a simulation-based objective function yield superior outcomes compared to the test-based objective function that previously attained the most favourable results in traditional software engineering transplantation (μSCALPEL [42]).

Procedural Content Generation PCG refers to the automation or semi-automation of the generation of content in video games [8]. PCG is a large field including many algorithms [49], which can be categorised in three main categories [50]: Traditional methods [51] generating content under a procedure without evaluation; Machine Learning methods (PCGML) [52], [53], [54] that train models to generate new content; and Search-Based methods (SBPCG) [8], [55], [56] that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions. Our work falls in the SBPCG category and it generates content of the NPC type. In the context of NPC generation using SBPCG, Ripamonti *et al.* [57] proposed an approach to generate monsters adapted to players, considering the monster with more death rate the preferred by the player. Pereira *et al.* [58] and, later, Viana *et al.* [59] seek for generating enemies that meet a difficulty criteria for an academic game. Blasco *et al.* [27] focuses 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 [60] and evolutionary algorithm. Gallota *et al.* as well as Blasco *et al.* use a commercial video game in their evaluation. 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 (from scenarios to NPCs).

IX. Conclusion and Future Work

In this study, we introduced the transplant metaphor in PCG for the first time. The results of our case study demonstrate that new content can be successfully generated through transplantation in an industrial setting. 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 to automatically create new content. In addition, we have presented a novel fitness function to guide the search for transplants, which obtained better results than the traditional one. This opens a new opportunity for software transplantation: Co-evolving transplants and simulations. For instance, one could evolve the simulations by adding or removing elements like NPCs, items, and scenarios. A co-evolutionary approach may enable developers to gain a deeper understanding of the contexts where the generated content performs better. Furthermore, co-evolutionary approaches may empower researchers to tackle the challenge of transplanting content between different games. Achieving the former would open the door to a vast and potentially highly original catalog of organs for transplantation that would contribute to achieving what developers seek with their video games: fun.

ACKNOWLEDGMENT

This work has been partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-100), by the Gobierno de Aragón (Spain) (Research Group T61_23R), by the Excellence Network AI4Software (Red2022-134647-T).

REFERENCES

[1] P. Rykała, "The growth of the gaming industry in the context of creative industries," *Biblioteka Regionalisty*, no. 20, pp. 124–136, 2020.

[2] C. Politowski, F. Petrillo, J. E. Montandon, M. T. Valente, and Y.-G. Guéhéneuc, "Are game engines software frameworks? a three-perspective study," *Journal of Systems and Software*, vol. 171, p. 110846, 2021.

[3] T. Wijman, "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, 2021, accessed: 01/12/23.

[4] ——, "The games market in 2022: The year in numbers," https://newzoo.com/resources/blog/the-games-market-in-2022-the-year-in-numbers, 2022, accessed: 01/12/23.

[5] SlashData, "State of the developer nation 23rd edition," [Online; accessed 18-December-2023]. [Online]. Available: https://slashdata-website-cms.s3.amazonaws.com/sample_reports/dsIe6JlZge_KsHWt.pdf

[6] Unity, "Unity," https://unity.com/, accessed: 01/02/24.

[7] Unreal, "Unreal," https://www.unrealengine.com/, accessed: 01/02/24.

[8] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, pp. 1–22, 2013.

[9] S. Tree, "Speed tree," https://store.speedtree.com, accessed: 01/02/24.

[10] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke, "Automated software transplantation," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 257–269.

[11] M. Farshbafnadi, S. Razi, and N. Rezaei, "Chapter 7 - transplantation," in *Clinical Immunology*, N. Rezaei, Ed. Academic Press, 2023, pp. 599–674. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128180068000086

[12] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 364–374.

[13] S. Sidiroglou-Douskos, E. Lahtinen, and M. Rinard, "Automatic error elimination by multi-application code transfer," 2014.

[14] T. Zhang and M. Kim, "Automated transplantation and differential testing for clones," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 665–676.

[15] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 288–302.

[16] S. Sidiroglou-Douskos, E. Lahtinen, A. Eden, F. Long, and M. Rinard, "Codecarboncopy," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 95–105.

[17] B. Reid, C. Treude, and M. Wagner, "Optimising the fit of stack overflow code snippets into existing code," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1945–1953.

[18] R. Gallotta, K. Arulkumaran, and L. Soros, "Evolving spaceships with a hybrid l-system constrained optimisation evolutionary algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 711–714.

[19] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[20] L. Pascarella, F. Palomba, M. Di Penta, and A. Bacchelli, "How is video game development different from software development in open source?" in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 392–402.

[21] J. Chueca, J. Verón, J. Font, F. Pérez, and C. Cetina, "The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games," *Information and Software Technology*, p. 107330, 2023.

[22] CryEngine, "Cryengine," https://www.cryengine.com, accessed: 01/02/24.

[23] U. Blueprint, "Unreal blueprint," https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/, accessed: 01/02/24.

[24] U. Scripting, "Unity scripting," https://unity.com/features/unity-visual-scripting, accessed: 01/02/24.

[25] M. Zhu and A. I. Wang, "Model-driven game development: A literature review," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–32, 2019.

[26] Á. Domingo, J. Echeverría, O. Pastor, and C. Cetina, "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, 2020, pp. 353–367.

[27] D. Blasco, J. Font, M. Zamorano, and C. Cetina, "An evolutionary approach for generating software models: The case of kromaia in game software engineering," *Journal of Systems and Software*, vol. 171, p. 110804, 2021.

[28] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu, *Evolutionary computation*. CRC press, 2000.

[29] Ø. Haugen, A. Wasowski, and K. Czarnecki, "Cvl: common variability language," in *Proceedings of the 16th International Software Product Line Conference-Volume 2*, 2012, pp. 266–267.

[30] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen, and A. Svendsen, "Adding standardized variability to domain specific languages," in *2008 12th International Software Product Line Conference*. IEEE, 2008, pp. 139–148.

[31] K. MacDonald, "Tough love: On dark souls' difficulty," https://www.eurogamer.net/tough-love-on-dark-souls-difficulty, 2019, accessed: 01/02/24.

[32] Z. Oliver, "Dynasty warriors = dumb fun," https://theologygaming.com/dynasty-warriors-dumb-fun/, 2013, accessed: 01/02/24.

[33] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, pp. 594–623, 2013.

[34] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[35] A. Arcuri and L. C. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verification Reliab.*, vol. 24, no. 3, pp. 219–250, 2014. [Online]. Available: https://doi.org/10.1002/stvr.1486

[36] E. Studios, https://www.entaltostudios.com/, accessed: 01/02/24.

[37] K. Empire, https://www.krakenempire.com/, accessed: 01/02/24.

[38] M. Zamorano, A. Domingo, C. Cetina, and F. Sarro, "Game software engineering: A controlled experiment comparing automated content generation techniques," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2024.

[39] M. Barros and A. Neto, "Threats to validity in search-based software engineering empirical studies," *RelaTe-DIA*, vol. 5, 01 2011.

[40] C. Miles, A. Lakhotia, and A. Walenstein, "In situ reuse of logically extracted functional components," *Journal in Computer Virology*, vol. 8, pp. 73–84, 2012.

[41] J. Petke, M. Harman, W. B. Langdon, and W. Weimer, "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, 2014, pp. 137–149.

[42] A. Marginean, "Automated software transplantation," Ph.D. dissertation, UCL (University College London), 2021.

[43] Y. Kwon, W. Wang, Y. Zheng, X. Zhang, and D. Xu, "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*, 2017, pp. 158–169.

[44] S. Sidiroglou-Douskos, E. Davis, and M. Rinard, "Horizontal code transfer via program fracture and recombination," 2015.

[45] J. Maras, M. Štula, and I. Crnković, "Towards specifying pragmatic software reuse," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, 2015, pp. 1–4.

[46] Y. Wang, Y. Feng, R. Martins, A. Kaushik, I. Dillig, and S. P. Reiss, "Hunter: next-generation code reuse for java," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 1028–1032.

[47] M. Allamanis and M. Brockschmidt, "Smartpaste: Learning to adapt source code," *arXiv preprint arXiv:1705.07867*, 2017.

[48] Y. Lu, S. Chaudhuri, C. Jermaine, and D. Melski, "Program splicing," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 338–349.

[49] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer, 2018, vol. 2.

[50] N. A. Barriga, "A Short Introduction to Procedural Content Generation Algorithms for Videogames," *International Journal on Artificial Intelligence Tools*, vol. 28, no. 2, pp. 1–11, 2019.

[51] J. Freiknecht and W. Effelsberg, "A survey on the procedural generation of virtual worlds," *Multimodal Technologies and Interaction*, vol. 1, no. 4, p. 27, 2017.

[52] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgard, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural Content Generation via Machine Learning (PCGML)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[53] J. Liu, S. Snodgrass, A. Khalifa, S. Risi, G. N. Yannakakis, and J. Togelius, "Deep learning for procedural content generation," *Neural Computing and Applications*, vol. 33, no. 1, pp. 19–37, 2021.

[54] K. Souchleris, G. K. Sidiropoulos, and G. A. Papakostas, "Reinforcement learning in game industry—review, prospects and challenges," *Applied Sciences*, vol. 13, no. 4, p. 2443, 2023.

[55] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[56] M. Zamorano, C. Cetina, and F. Sarro, "The quest for content: A survey of search-based procedural content generation for video games," *arXiv preprint arXiv:2311.04710*, 2023.

[57] L. A. Ripamonti, F. Distefano, M. Trubian, D. Maggiorini, and D. Gadia, "Dragon: diversity regulated adaptive generator online," *Multimedia Tools and Applications*, vol. 80, no. 26, pp. 34 933–34 969, 2021.

[58] L. T. Pereira, B. M. Viana, and C. F. Toledo, "Procedural enemy generation through parallel evolutionary algorithm," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2021, pp. 126–135.

[59] B. M. Viana, L. T. Pereira, and C. F. Toledo, "Illuminating the space of enemies through map-elites," in *2022 IEEE Conference on Games (CoG)*. IEEE, 2022, pp. 17–24.

[60] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *Journal of theoretical biology*, vol. 18, no. 3, pp. 280–299, 1968.