Automated Transplantation for Procedural Content Generation in Video Games

Mar Zamorano, Carlos Cetina, Federica Sarro

***Abstract*—**

***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 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 [5]. In a video game, software often 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 the game levels to the NPCs or game items such as weapons and power ups.

Content generation is a generally slow, tedious, costly, and error-prone manual process. In order to cope with the growing demand for content for video games, researchers are 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 [6]. Usually, current PCG approaches work as follows: developers provide initial content (usually human-generated content) into an algorithm to work with. Afterwards, the algorithm (Traditional, Machine Learning, or Search-Based methods) will generate new content. Only a few traditional methods have succeeded in providing tools used by the industry to randomly generate vegetation (e.g., SpeedTree in Unreal and Unity).

In this paper, we propose a new angle to tackle PCG for video games inspired by transplantation techniques [7], that 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 [8]. 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) [7]. Software transplantation has achieved success on different tasks: program repair [9], [10], testing [11], security [12], or functionality improvements [13].

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 (other 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[4], relies on Search-based Software Engineering (SBSE) because SBSE has demonstrated success on software transplantation [7]. In the literature, software transplantation approaches guide the search by using test-suites. The transplantation assessment is determined by the amount of test that a candidate solution is able to pass. Our work not only explores the use of test-suite (Test-based Imhotep variant) but also we explore the use of video game simulations (Simulation-based Imhotep variant), 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

[1]Non-playable characters which are not controlled by a player.
[2]https://unity.com/
[3]https://www.unrealengine.com/

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

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), NPCs explore the game levels 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[5]. The game has been released on PC, PlayStation, and translated to eight different languages. To evaluate Imhotep, 129 different organs extracted from the scenarios of Kromaia are transplanted into 5 of the video game bosses that act as hosts, generating new video game bosses in the process. In total, our approach analysed 645 transplants. To the best of our knowledge, our work has more transplants than previous work in the literature, with a maximum of 327 successful transplants [14].

The results of the two Imhotep variants (test-based and simulation-based Imhotep) and a PCG baseline from the literature [14] are compared against an oracle (provided by developers). The results show that, out of the three approaches (the two Imhotep variants and the baseline), the content generated through the simulation-based Imhotep variant obtains the closest results to the oracle for all the generation scenarios (32% better than the PCG baseline). The test-based Imhotep variant obtains the second place (25% better than the PCG baseline), with the baseline obtaining worse results than the other two in all scenarios. The generated bosses are a promising starting point: Developers can either include them directly in the game, modify them to better suit their needs, or inspect them to find novelties from which they can create more original designs.

Our contributions can be summarized as follows:

1 Novel application of Software Transplantation to Procedural Content Generation (PCT approach),
2 Software Transplantation of software models in the field of video games development, and
3 Comparison of two objective functions based on the trends in Software Transplantation and on the trends in PCG.

The rest of the paper is structured as follows: Section II provides some background to MAR ask Fe word in comments our work. Section III describes our approach, depicting its usage for PCG. Section IV details the evaluation of our approach. Section V highlights the results of our research. Section VI discusses the outcomes of the paper and future lines of work. Section VII outlines the threats to the validity of our work. Section VIII reviews the works related to this one. Finally, Section IX concludes the paper by summarizing the main contributions and results.

II. Background

A. 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 for the video game [15], [16].

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 [6] 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 [17] 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, 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 Figure 1).

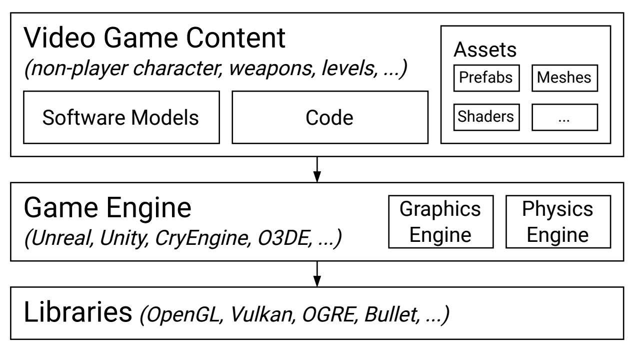

Fig. 1: Overview of video game artifacts.

B. 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

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

[6]https://unity.com/

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

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.

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 defeated in order to complete Level 1. The bottom part of the figure illustrates the two possible development strategies for the boss.

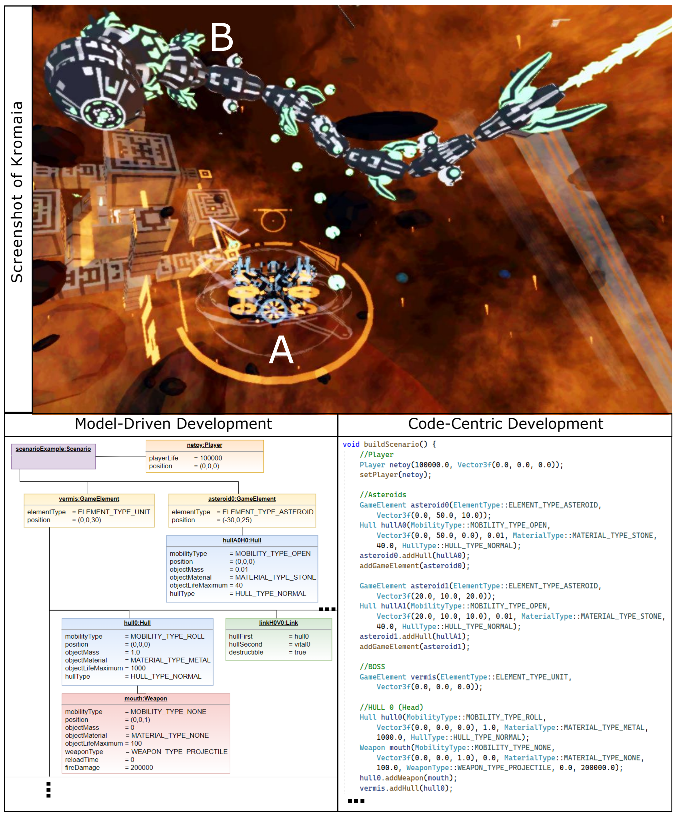

Fig. 2: Model-Driven Development vs. Code-Centric Development in the context of Kromaia

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 implement the two simultaneously. 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 [5], but also counting game designers, artists, UI designers, and QA engineers within their ranks - possibly favours the use of software models over code, since 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 [18] confirmed that video game developers make fewer mistakes and are more efficient when working with the models than with the 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 defenses; 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 [19]. More information on the SDML model can be found in the following video presentation: https://youtu.be/Vp3Zt4qXkoY.

III. Our Imhotep Approach

This section will explain how our approach makes use of evolutionary computation to transplant elements 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 auto-transplantation of content within a simplified version of 'bosses' of the video game Kromaia.

Figure 3 shows an overview of our approach. At the top left of the figure we show the input to our approach, which are the organs to be transplanted from the donor and the hosts where the organ will be transplanted. Afterwards, Imhotep detects the points of the organ that allows the transplantation and the points where the organ can be inserted. To initialize the population of the evolutionary algorithm, the organ is transplanted in a random point and cloned. When the evolutionary algorithm finish the execution we obtain a ranked list by the objective function of the best transplantation between organ and host. We describe each step on detail in the following subsections.

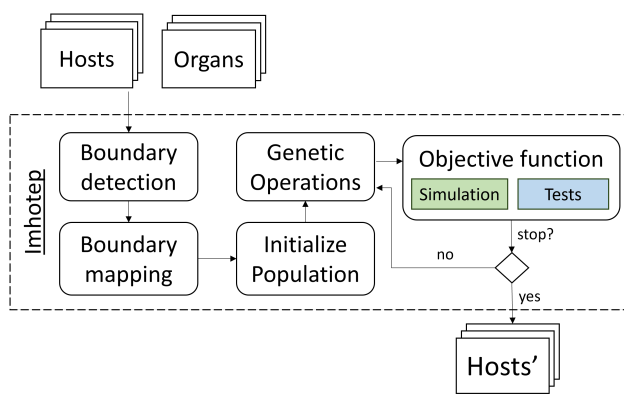

Fig. 3: Overview of Imhotep

A. Input selection

The very first step of our approach is the definition of the input. Imhotep requires the developers to define a source model content (donor) with the organ that will be transplanted, and a target model content (host). The models used in Imhotep are models based on SDML as explained in the background section (II). The donor and host from the example are a simplified version of the donor and host used in the evaluation, which we think they will help to understand the approach.

In our example we present a simplified version of the metamodel, and the corresponding concrete syntax of the model (Fig. 4) from the video game Kromaia. We use a graphical representation to help the comprehension of the reader, however the original metamodel does not work with a graphical model representation as it is not a requirement on every metamodel. The type of model will depend on the metamodel and models that developers decide on.

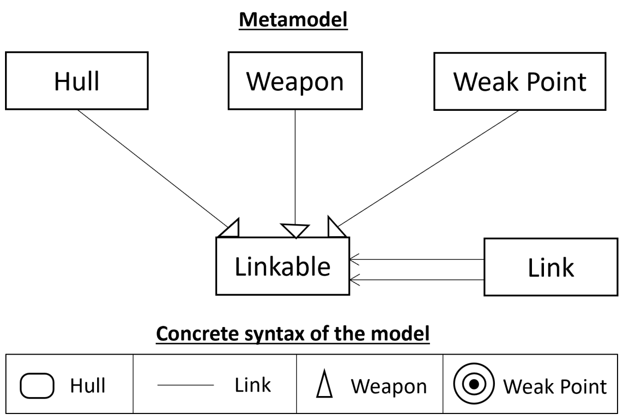

Fig. 4: Simplified metamodel with the corresponding concrete syntax of the model.

Based on the metamodel of our example, we define the inputs as stated. First, we define the source donor, that is a simplified version of an original ‘boss’ from Kromaia, called ‘Serpent’ (Fig. 5). The original model is a SDML model written on a XML file with approximately 1700 lines of code. Figure 5 shows the graphical representation of the donor, differentiating each element of the model with a letter from A to S. It also shows in green the elements selected as organ (the elements H, I, J, K, N, O, P, Q). Secondly, we define the host of our example. To that extent, we have created a regular enemy that could appear in Kromaia following the metamodel (Fig. 6).

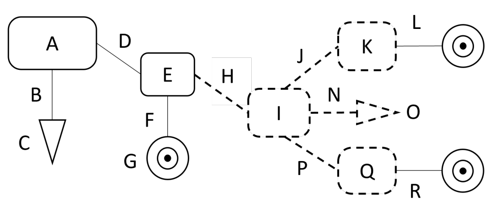

Fig. 5: Donor with organ selection in dotted lines. The letters represent each element of the model.

B. 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 of the organ

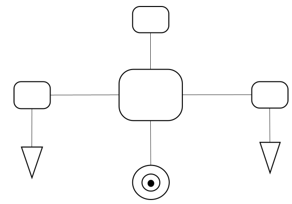

Fig. 6: Host

and the host. A boundary is a connection point on an element capable of connecting two distinct elements within a model. The connection is restricted by the rules of the metamodel. Imhotep automatically identifies the boundaries of the selected organ, and all the boundaries of the host before initializing the evolutionary algorithm.

Following our example the approach will detect the boundaries of the organ, and the host. The boundaries of the organ will be the connection points between donor and host. The elements that connect with the rest of the donor are H, K, and Q. We can then state the boundaries on each element. Figure 7 shows the donor, the organ, and the boundaries (boundaries are represented by a circle crossed). The boundaries of the organ will be; b11 for the H element; b16 for the K element, and b25 for the Q element.

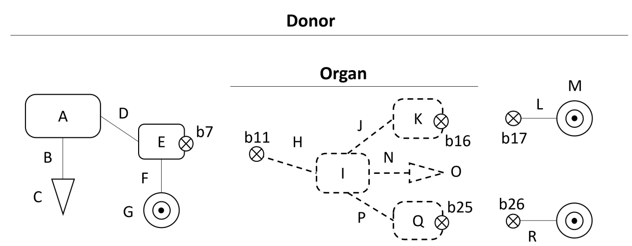

Fig. 7: Definition of donor boundaries. The boundary is represented by a circle crossed.

On the other hand, the boundaries of the host will be all the points where its elements connect. Figure 8 shows all the boundaries of the host of the example. The host has a total of 19 boundaries identified by a tag from ba to bs.

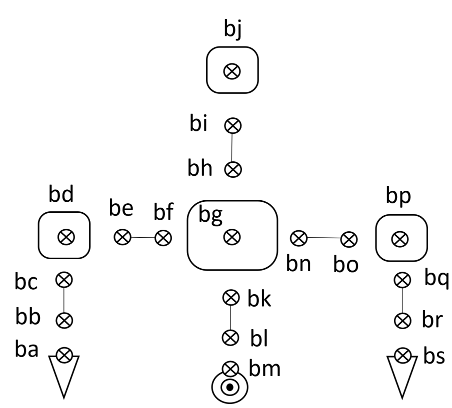

Fig. 8: Definition of host boundaries. The boundary is represented by a circle crossed.

C. Boundaries mapping

In the boundary mapping step, Imhotep determine the relation between the boundaries of the organ and the host. From each boundary in the organ we map all possible connections that could have with boundaries of the host, including the possibility of not connecting the boundary to the host boundaries (Table. I).

Table I map the compatible boundary connections between organ and host. 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’ or remain unconnected. The boundaries b16 and b25 are both ‘Hulls’ and they can connect with any ‘Link’ or remain unconnected.

Organ boundary from donor	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 I: Mapping of compatible boundaries between organ and host.

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, each individual represents a software model from the game. We use a similar encoding version of Blasco *et al.* [19] that has been adapted to work with transplantations. The size of the encoding in the previous work was 64 and in this work its size is of 150. The initial population of Imhotep contains 100 clones of a same individual, which is composed by a host and an organ placed in a random position.

E. Genetic Operators

Imhotep has three genetic operations (selection, crossover, and mutation) to generate new individuals. To select the individuals we use the ranking selection, which ranks the population by the objective function and takes the top 10% of the individuals in the current population. The size of the population is limited to 100, *i.e.*, the selection will take 10 individuals to apply the operators.

We use a single, random, cut-point crossover. It 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 child solutions by putting the first part of the first parent with the second part of the second parent for the first child and putting the first part of the second parent with the second part of the first parent for the second child.

The new individual has a probability of 1/150 to mutate any value of the encoding. The probability is based on the size of the encoding of an individual which is 150 in our approach.

After all the operations have been applied, the new set of individuals are evaluated by the objective function and added to the population. The last individuals by ranking in the population are discarded to maintain the size of the population on 100.

Figure. 9 shows handmade new individuals that could results from our example.

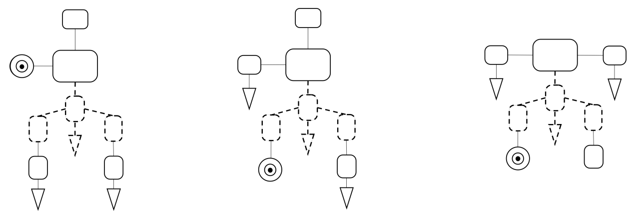

Fig. 9: Example of individuals to be evaluated by the objective function

F. Objective function

The objective function in Imhotep assesses the quality of each individual as a model. First, as done in previous work that use Kromaia [19], the models pass through a validation process followed by a quantitative measurement. In our work we assess quantitatively the objective function by two means: Test-based and Simulation-based objective functions. We use two different objective functions due to the differences in the the state-of-the-art of software transplantation and PCG. The state-of-the-art in software transplantation mainly work with Test-based objective function, while the state-of-the-art in NPCs PCG work with Simulation-based objective function.

The validation step before the Test-based or Simulation objective function is a requirement that Kromaia integrates in the game to avoid models with inconsistent data. 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 the objective function will be 0.0.

The models that pass the validation process will then be assess by the Test-based and Simulation-based objective functions. For the Test-based objective function we ask the developers to provide the set of tests that they consider relevant to our work. 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.

On the other hand, the Simulation-based objective function as in Blasco *et al.* [19] simulates an in game battle between the boss and a player. 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.* [19]:

$F_{Victory}$ 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) ($V_{Optimal}$):

$$F_{Victory} = 1 - \frac{|\, V_{Optimal} - V_P \,|}{V_{Optimal}} \tag{1}$$

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

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

$F_{Overall}$ is an average fitness value for a boss model that includes the fitness criteria described above. $F_{Overall}$ 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.

$$F_{Overall} = min(Validity, \frac{\sum_{i=1}^{N} F_i}{N}) \tag{3}$$

IV. Experimental Design

In this section we explain how we evaluate the feasibility of automated transplantation in video games through Imhotep. To do so, we run an experiment evaluating Imhotep, with a measure from the literature, and we have conducted a preliminary evaluation with human developers. Through this section, we will present the research questions that we aim to answer, the evaluation process (including the measure quality for the solutions and baseline), and the implementation details.

A. Research Questions

We aim to answer the following research questions:

RQ1: *What is the quality of the models generated by Imhotep in contrast to the models from the oracle?*

RQ2: *What is the quality of the models generated whith each variant of Imhotep (Simulation-based and Test-based)?*

RQ3: *Is there a significant difference between a traditional PCG approach and a transplantation approach?*

B. Planning and execution

Figure 10 shows an overview of the evaluation process. The upper part of the figure shows the software models selected from the original video game content provided by developers from Kromaia, which are the inputs for the test cases.

In the figure, the output of the test cases are a baseline and the two variants of our approach. We used the work by Gallota *et al.* [20] as PCG baseline. Gallota *et al.* presented a hybrid Evolutionary Algorithm for generating NPCs, more precisely spaceships. Their approach combine an L-system with Feasible Infeasible Two Population Algorithm. Gallota *et al.* were able to evolve spaceships that match some statistics of human-designed spaceships. The two variants of our Imhotep approach work as described in Section III to form the transplanted models that are considered to be the most relevant transplantations.

The evaluate the output of the test cases we compare them with an oracle. The oracle is extracted from the original software models from the video game Kromaia. The oracle and the output pass through a simulation provided by the game developers of Kromaia, which simulates an in game battle between the boss and a player. From the simulation we extract the duration, which is a metric commonly used by the literature [21].

Duration: The duration of duels between players and boss units is expected to be around a certain optimal value. For the video game 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 ($T_{Optimal}$):

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

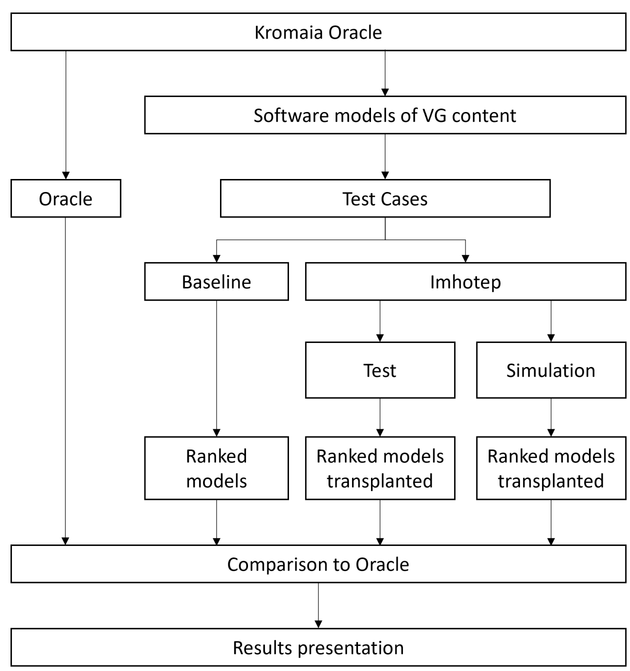

Fig. 10: Overview of our evaluation process.

C. Implementation details

For the evaluation we used 5 different host (Vermis, Teuthus, Argos, Orion, and Maia), which are original bosses from the video game Kromaia. As a donor we used the scenarios of the video game, and we collected 129 organs, that are elements from the scenario. Each organ was transplanted individually to each boss. Hence, we obtain a total of 645 new individuals (5*129). Each variant of Imhotep provided a total of 645 new individuals as output, 645 new individuals from Simulation-based and 645 individuals from Test-based. In the case of the baseline, to make it fair, the approach was executed 129 times with each one of the 5 different hosts to obtain 645 new individuals.

In order to compare the baseline to the variants of Imhotep, we chose the parameters shown in Table II to calibrate the evolutionary algorithm and the objective function. We established the stop condition at 2 minutes and 30 seconds, ensuring that the approaches run long enough to obtain the best solutions. Even though the population size is 100 individuals, we only present the best candidate in each run of the variants and the same with the baseline. We assume that the best candidate solutions are those with the highest objective function value.

The evaluation of Imhotep and the baseline was done using two Pcs with the following specifications; Intel Core i7-8750H, 16GB, 2.2GHz; and 2x Intel(R) Xeon(R) CPU X5660, 64GB, 2.80GHz. The implementation uses the Java(TM) SE Runtime Environment (JDK 1.8), together with Java as the programming language. For purposes of replicability, the implementation source code and the data (software models and oracles) are publicly available at the following URL: TODO *package of replicability*

Parameter description	**Value**
Stop Time	2m 30s
Population Size	100
Number of parents	2
Number of offspring from parents	2
Crossover probability	1
Mutation probability	1/150

TABLE II: Parameter settings

V. Results

Figure 11 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 PCG Baseline. The executions are grouped by each host (boss of Kromaia) that has been used in our experiment (Vermis, Teuthus, Argos, Orion, and Maia). The last column, with shaded background, shows the average of all the hosts for each objective function and the baseline. In addition, the oracle indicates the value obtained by the human-generated final boss models that were obtained from Kromaia.

Each boxplot is generated from the results of each host obtained from the transplantation of each of the 5 hosts with each of the 129 organs. Therefore, each boxplot represents 645 values of a specific host-organ transplantation in a final boss model. Figure 11 shows in each column how the quality values obtained for each of the three strategies studied in our evaluation differ from the values for the models generated by the developers, which are represented by the horizontal red dashed lines that cross each host column. The boxplots that are closer to the horizontal lines are more similar in quality to the models produced by the developers. Additionally, the use of boxplots allows for the representation of the different results for the strategies used.

TODO *Analysis of the results. Simulation has the best results, test also better than baseline...*

VI. Discussion

VII. Threats to Validity

To acknowledge the threats to the validity of our work, we use the classification suggested by De Oliveira *et al.* [22].

1. Conclusion Validity Threats. To minimize *not accounting for random variation*, we have a total of 645 transplants for each host on each variation of Imhotep and the baseline. Also, each transplant has run for 2 minutes and 30 seconds. 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 tackled the *lack of a meaningful comparison baseline* by comparing our two variants of Imhotep with a recent traditional PCG approach as baseline.

2. Internal Validity Threats. To mitigate *poor parameter settings* we have presented the parameters used in our experiment, and for the PCG baseline we have used the parameters presented by the original work. We provide the source code and the artifacts used in our experiments to allow 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.

3. Construct Validity Threats. To prevent the *lack of assessing the validity of cost measures* threat, we made a fair comparison between the two variants of our approach and the baseline. Furthermore, we used duration as our metric for the evaluation, which is a metric adopted and *validated* from the literature [21]. To mitigate the *lack of discussing the underlying model subjected to optimization*, we use the original SDML of the video game provided by the developers of Kromaia.

4. External Validity Threats. To mitigate the *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 our approach in other video games, three main ingredients are required as in other SBSE approaches: encoding, operations, and fitness function. The crossover and mutation operations are extensively utilized. The encoding and the fitness function depend on the content to generate. Our approach should be replicated with other DSL and video games before assuring its generalization. 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.

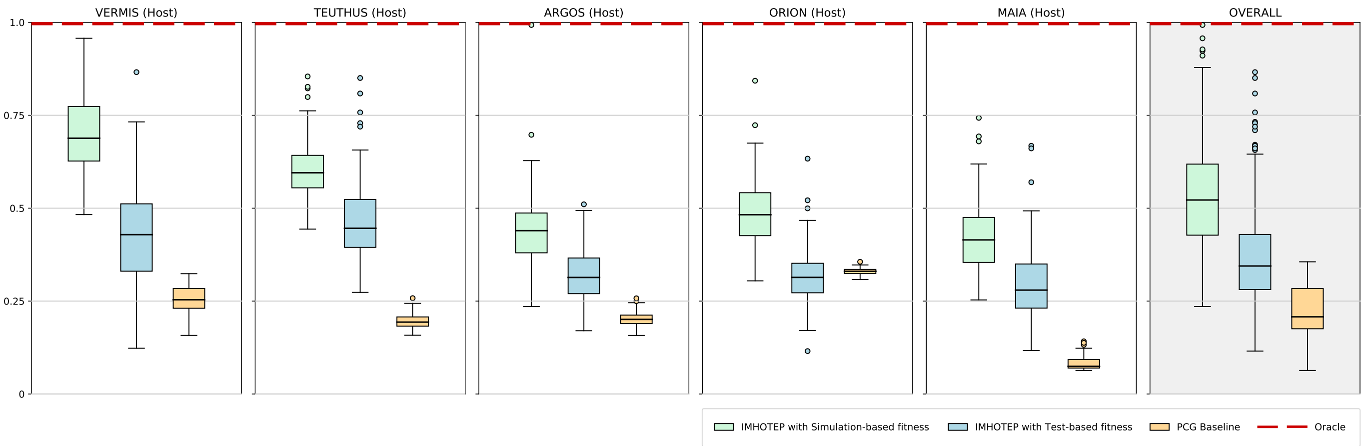

Fig. 11: Results

TABLE III: Mean Values and Standard Deviations

	Vermis	Teuthus	Argos	Orion	Maia	Overall
Simulation	0.699 ± 0.105	0.607 ± 0.074	0.439 ± 0.093	0.488 ± 0.087	0.430 ± 0.121	0.533 ± 0.142
Test	0.424 ± 0.130	0.463 ± 0.105	0.321 ± 0.069	0.314 ± 0.068	0.295 ± 0.093	0.363 ± 0.117
Baseline	0.254 ± 0.033	0.195 ± 0.018	0.201 ± 0.018	0.329 ± 0.008	0.084 ± 0.018	0.213 ± 0.083

TABLE IV: Max and Min values.

	Vermis		Teuthus		Argos		Orion		Maia		Overall	
	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min
Simulation	1.042	0.482	0.854	0.443	0.992	0.235	0.842	0.304	1.285	0.253	1.285	0.235
Test	0.866	0.123	0.850	0.273	0.510	0.170	0.633	0.115	0.667	0.117	0.866	0.115
Baseline	0.323	0.157	0.257	0.158	0.257	0.157	0.355	0.307	0.141	0.063	0.355	0.063

VIII. Related work

This work generates original 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.

A. Automated Software Transplantation

On functionality transplantation, Miles *et al.* [23] and Petke *et al.* [24] 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.* [25] 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.* [26] 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.* [27] 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 [28] 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.* [28] use Gated Graph Neural Networks [29] to predict the correct variable name in an expression.

In 2018, Lu *et al.* [30] 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 [7] 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 [31] 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 [32] 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 [33] 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 (μSCALPEL).

B. Procedural Content Generation

Procedural Content Generation (PCG) refers to the automation or semi-automation of the generation of content in video games [6]. The types of content generated by PCG are diverse, such as vegetation [34], sound [35], terrain [36], Non-Playable Characters (NPCs) [37], dungeons [38], puzzles [39], and even the rules of a game [40].PCG is a large field spanning many algorithms [41], which can be grouped in three main categories according to the survey of PCG techniques by Barriga et al. [42]: Traditional methods [43] that generate content under a procedure without evaluation; Machine Learning methods (PCGML) [44], [45], [46] that train models to generate new content; and Search-Based methods (SBPCG) [6], [47] that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

An interesting aspect of SBPCG is the objective function (or fitness function) that guides the search towards an optimal solution. SBPCG differentiates between three different types [47]: direct, simulation, and interactive. Direct objective functions are those that are based on the available knowledge of developers (that is, the developers themselves participate in the assessment of the objective function). Direct objective functions can be either theory-driven (meaning that the opinion of the developers is directly leveraged) or data-driven (meaning that information about relevant parameters is extracted from artefacts like questionnaires or player models). Simulation objective functions replicate real situations to estimate the behaviour of real players. Work in this area focuses mainly on developing more human-like agents, bots, and AIs to be used as objective functions. Simulation objective functions can be static, where the simulator agent does not change during the simulation, or dynamic, where agents that learn during simulation are used. Finally, interactive objective functions are those that involve players in the composition of the objective function. In SBPCG, interactive objective functions can be either explicit, when players are outright asked for their opinions, or implicit, when the data is indirectly extracted or inferred from the observation of the actions of the players and the results of those actions.

Our work is positioned within SBPCG in the NPCs category, our approach transplant scenario elements into a NPC to obtain a novel version of the NPC. From the best of our knowledge we are the first work applying transplantation in PCG. Guarneri *et al.* [48] or Norton *et al.* [49] generate NPC monsters through an evolutionary algorithm with the aim of obtaining a diversity set of new monsters. With the same goal, Ripamonti *et al.* [50] developed a novel approach to generate monsters adapted to players, considering the monster with more death rate the preferred by the player. Pereira *et al.* [51] and later extended by Viana *et al.* [37] instead of diversity seek for generating enemies that meet a difficulty criteria.

Our work uses the same case study as Blasco *et al.* [19] who generate spaceship enemies which quality is comparable to manually content created by developers. Their approach also works software models as we do, instead of code. On other hand, to generate also spaceships, Gallota *et al.* [20] used a combination of Lindenmayer systems [52] and evolutionary algorithm.

Our research introduces a fresh perspective on content generation through the use of transplantation, which sets it apart from traditional procedural content generation (PCG) methods. Transplantation enables the seamless integration of various content types, facilitating in our work the transplant of elements from scenarios to NPCs.

IX. Conclusion

Acknowledgments

This work was supported in part by the Spanish National R+D+i Plan under the Project VARIATIVA (PID2021-128695OB-100), the José Castillejo program (CAS18/00272), and the ERC grant no. 741278 (EPIC: Evolutionary Program Improvement Collaborators).

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," [Online; accessed 1-December-2023]. [Online]. Available: https://newzoo.com/resources/blog/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming

[4] ——, "The games market in 2022: The year in numbers," [Online; accessed 1-December-2023]. [Online]. Available: https://newzoo.com/resources/blog/the-games-market-in-2022-the-year-in-numbers

[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] 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.

[7] 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.

[8] 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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

[29] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[30] 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.

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

[32] 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.

[33] Y. Kwon, X. Zhang, and D. Xu, "Pietrace: Platform independent executable trace," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 48–58.

[34] C. Mora, S. Jardim, and J. Valente, "Flora generation and evolution algorithm for virtual environments," in *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2021, pp. 1–6.

[35] D. Plans and D. Morelli, "Experience-driven procedural music generation for games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 3, pp. 192–198, 2012.

[36] M. Frade, F. Fernandéz de Vega, C. Cotta *et al.*, "Breeding terrains with genetic terrain programming: the evolution of terrain generators," *International Journal of Computer Games Technology*, vol. 2009, 2009.

[37] 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.

[38] B. M. Viana and S. R. dos Santos, "A survey of procedural dungeon generation," in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2019, pp. 29–38.

[39] B. De Kegel and M. Haahr, "Procedural puzzle generation: A survey," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 21–40, 2019.

[40] C. B. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland University of Technology, 2008.

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

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] A. Guarneri, D. Maggiorini, L. Ripamonti, and M. Trubian, "Golem: generator of life embedded into mmos," in *ECAL 2013: The Twelfth European Conference on Artificial Life*. MIT press, 2013, pp. 585–592.

[49] D. Norton, L. A. Ripamonti, M. Ornaghi, D. Gadia, and D. Maggiorini, "Monsters of darwin: A strategic game based on artificial intelligence and genetic algorithms," in *GHITALY*, vol. 1956. CEUR-WS, 2017.

[50] 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.

[51] 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.

[52] 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.