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 tests 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.
[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 preprogrammed 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 [15] 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 [16], [17].

Nowadays, most video games are developed by means of game engines. Game engines are development environments that integrate a graphics engine and a physics engine as well as tools for both to accelerate development. The most popular ones are Unity and Unreal Engine, but it is also possible for a studio to make its own specific engine (e.g., CryEngine [6]).

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 [18] 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 Fig. 1).

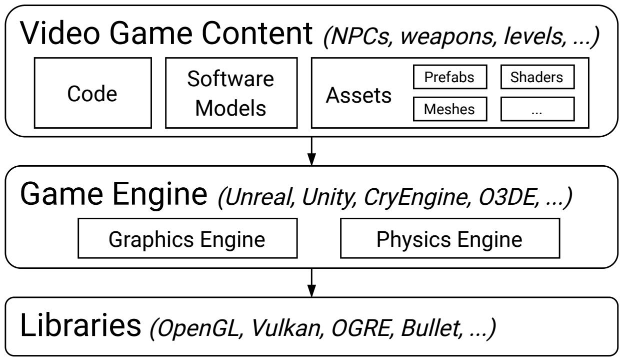

Video Game Content (NPCs, weapons, levels, ...)
Code
Software Models
Assets
Prefabs
Shaders
Meshes
...
Game Engine (Unreal, Unity, CryEngine, O3DE, ...)
Graphics Engine
Physics Engine
Libraries (OpenGL, Vulkan, OGRE, Bullet, ...)

Fig. 1: Overview of video game artifacts.

B. Kromaia

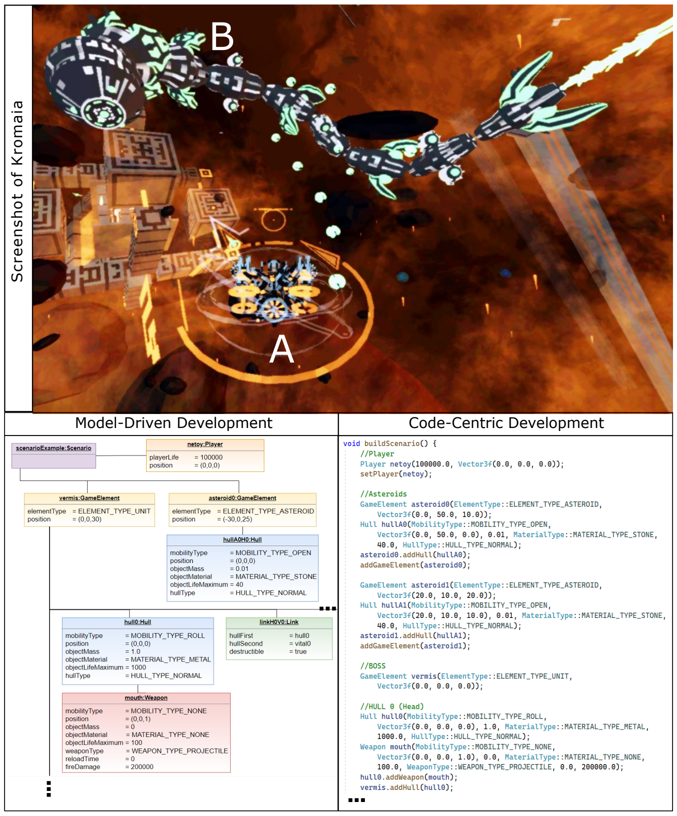
The research presented in this paper is framed within the context of a commercial video game case study, Kromaia. In particular, our evaluation uses the bosses of the video game to evaluate the approach. Each level of Kromaia consists of a three-dimensional space where a player-controlled spaceship has 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://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 approaches for the boss.



```
void buildScenario() {
    //Player
    Player netoy(100000.0, Vector3f(0.0, 0.0, 0.0));
    setPlayer(netoy);

    //Asteroids
    GameElement asteroid0(ElementType::ELEMENT_TYPE_ASTEROID,
        Vector3f(0.0, 50.0, 10.0));
    Hull hullA0(MobilityType::MOBILITY_TYPE_OPEN,
        Vector3f(0.0, 50.0, 0.0), 0.01, MaterialType::MATERIAL_TYPE_STONE,
        40.0, HullType::HULL_TYPE_NORMAL);
    asteroid0.addHull(hullA0);
    addGameElement(asteroid0);

    GameElement asteroid1(ElementType::ELEMENT_TYPE_ASTEROID,
        Vector3f(20.0, 10.0, 20.0));
    Hull hullA1(MobilityType::MOBILITY_TYPE_OPEN,
        Vector3f(20.0, 10.0, 10.0), 0.01, MaterialType::MATERIAL_TYPE_STONE,
        40.0, HullType::HULL_TYPE_NORMAL);
    asteroid1.addHull(hullA1);
    addGameElement(asteroid1);

    //BOSS
    GameElement vermis(ElementType::ELEMENT_TYPE_UNIT,
        Vector3f(0.0, 0.0, 0.0));

    //HULL 0 (Head)
    Hull hull0(MobilityType::MOBILITY_TYPE_ROLL,
        Vector3f(0.0, 0.0, 0.0), 1.0, MaterialType::MATERIAL_TYPE_METAL,
        1000.0, HullType::HULL_TYPE_NORMAL);
    Weapon mouth(MobilityType::MOBILITY_TYPE_NONE,
        Vector3f(0.0, 0.0, 1.0), 0.0, MaterialType::MATERIAL_TYPE_NONE,
        100.0, WeaponType::WEAPON_TYPE_PROJECTILE, 0.0, 200000.0);
    hull0.addWeapon(mouth);
    vermis.addHull(hull0);
...
```

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 [19] 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 [20]. More information on the SDML model can be found in the following video presentation: https://youtu.be/Vp3Zt4qXkoY.

III. Our Imhotep Approach

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

Fig. 3 shows an overview of our approach. At the top left of the figure we show the input to our approach, which are the organ to be transplanted from the donor and the host 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 into the host. To initialize the population of the evolutionary algorithm, the organ is cloned and transplanted in a random point. Genetic operations generate potential solutions for transplantation, while the objective function asses the quality of these solutions. This process of generating and assessing is repeated until a specific stop condition is met. When the evolutionary algorithm finish the execution we obtain a ranked list by the objective function of the best transplantation between organ and host.

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

A. Input selection

Imhotep requires the developers to identify a source model content (donor) with the organ that will be transplanted, and a target model content (host). In our running example we present a simplified version of the metamodel, and the corresponding

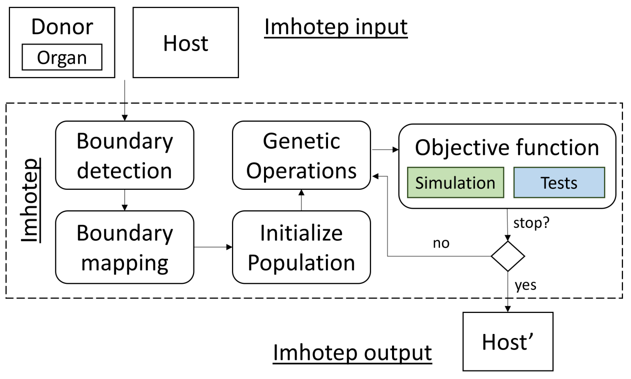


Fig. 3: Overview of our Imhotep approach.

concrete syntax of the model (see Fig. 4) from the video game Kromaia. 'Hulls' serve as the structural framework that define the anatomical composition of the models. For example, the boss presented on Fig. 2 (identified as 'B') has its body built by hulls. 'Weak points' 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'.

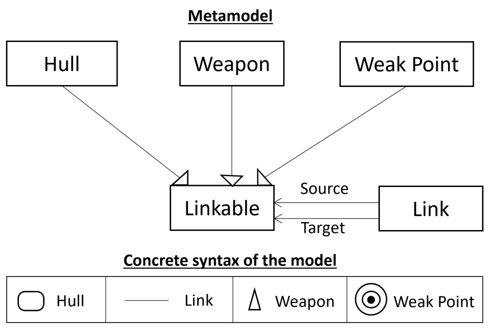


Fig. 4: Simplified metamodel with the corresponding concrete syntax of the model.

In the running example, the source donor model is a simplified version of an original 'boss' from Kromaia, called 'Serpent'. Fig. 5 shows the graphical representation of the donor model, differentiating each element of the model with a letter from A to S. It also shows with dashed lines the elements selected as organ (the elements H, I, J, K, N, O, P, Q). This simplified example is inspired by the boss shown in Fig. 2 with letter B. In the running example, the host is a model of a regular enemy that could appear in Kromaia. Fig. 6 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 do that we use the boundaries between the model elements of the organ and the host. A boundary is a connection point capable of connecting two distinct model elements within a model. The connection is restricted by the rules of the metamodel. In the simplified example in Fig. 4, the Source and Target meta-relationships are the boundaries

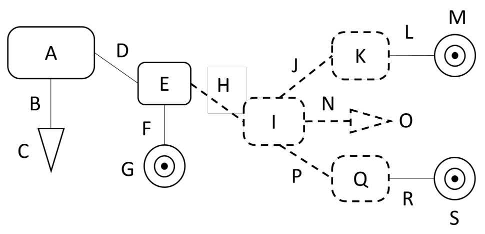


Fig. 5: Donor model with organ selection in dashed lines.

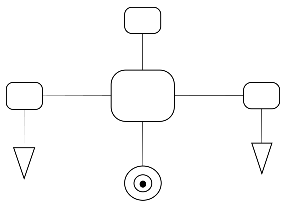


Fig. 6: Host

between the model elements of the models conforming to that metamodel. In other metamodel languages, there will be other meta-relationships with other names that will be the boundaries.

Imhotep automatically identifies the boundaries of the selected organ, and all the boundaries of the host. In the running example, the boundaries of the organ are the connection points between donor and host. The elements that connect with the rest of the donor are H, K, and Q. Fig. 7 shows the donor, the organ, and the boundaries (boundaries are represented by a circle crossed). The boundaries of the organ are as follows: b11 for the H element; b16 for the K element, and b25 for the Q element.

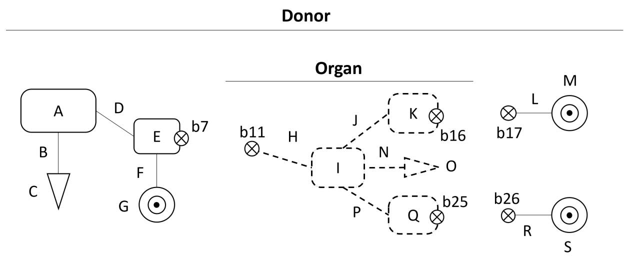


Fig. 7: Donor model boundaries. The boundary is represented by a circle crossed.

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

C. Boundary mapping

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

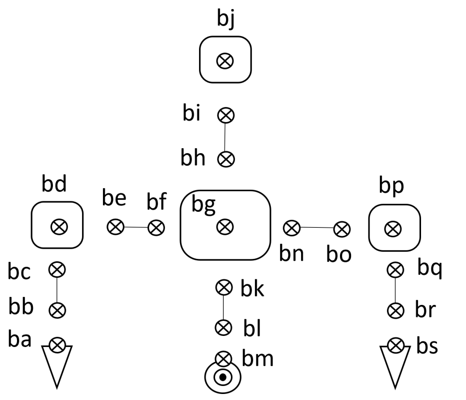


Fig. 8: Host model boundaries. The boundary is represented by a circle crossed.

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

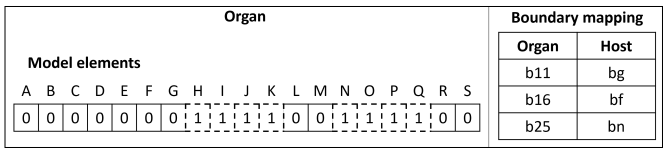
Organ boundaries	Host boundaries	
b11	ba	bm
	bd	bp
	bg	bs
	bj	Not connected
b16	bb	bc
	be	bf
	bh	bi
b25	bk	bl
	bn	bo
	Not connected	

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, the encoding requires a binary vector that represents the organ in the donor, and the boundary mapping (see Fig. 9). In the binary vector, each element from the model is a position from the vector. If a position in the vector has a ‘1’, it means that the element from the model is part of the organ. On the other hand, each boundary from the organ gets assigned a compatible boundary from the host. The initial population of Imhotep contains individuals composed by the host and the organ placed in a random position (a random mapping between the organ boundaries and the compatible organ boundaries).



Organ

Model elements

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0

Boundary mapping

Organ	Host
b11	bg
b16	bf
b25	bn

Fig. 9: Example of encoding.

E. Genetic Operators

Imhotep has genetic operations (selection, crossover, and mutation) to generate new individuals of the population. To select the individuals, we use the ranking selection, which ranks the population by the objective function and takes the top individuals in the current population.

We use a single, random, cut-point crossover. 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. Finally, the new individual has a probability to mutate any value of the encoding.

Fig. 10 shows example of new individuals that could results from the running example. For simplicity, these individuals have unaltered organs but illustrate different boundary mappings between organ and host.

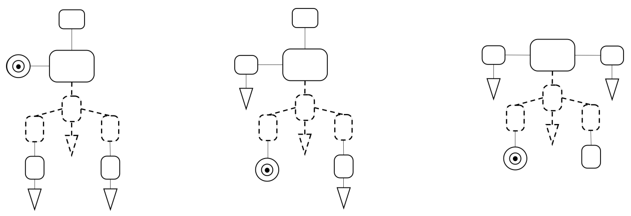


Fig. 10: Example of individuals.

F. Objective function

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

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

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

In this work, the Simulation-based variant of Imhotep assesses the transplants through a simulation of a game battle between the boss (Host') and a NPC spaceship. The information retrieved from the simulation is the data that the developers regard as relevant, using their domain knowledge. Hence, our approach takes into account the percentage of simulated player victories ($F_{Victory}$) and the percentage of simulated player health left once the player wins a duel (F_{Health}). The calculation of $F_{Victory}$ and F_{Health} is performed in the same way as Blasco *et al.* [20]:

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

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

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

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

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

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

IV. Experimental Design

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

A. Research Questions

We aim to answer the following research questions:
RQ1: *What is the quality of the content generated by Imhotep in contrast to the oracle?*
RQ2: *What is the quality of the content generated whith each variant of Imhotep (Simulation-based and Test-based)?*
RQ3: *Is there a significant quality difference between the traditional PCG approach (baseline) and our transplantation approach?*

B. Planning and execution

Fig. 11 shows an overview of the evaluation. The white background part at the top shows the assets of the game itself (content) and the game development (test suite) that are used by the approaches. The grey background part in the middle shows the inputs and outputs of the three approaches (the two Imhotep variants and the baseline). Finally, the white background part at the bottom shows the evaluation of the results of the approaches.

We used the work by Gallota *et al.* [15] as PCG baseline. Gallota *et al.* propossed a hybrid Evolutionary Algorithm for generating NPCs. Specifically, their approach combines an L-system with a Feasible Infeasible Two Population Evolutionary Algorithm. We choose Gallota *et al.* as PCG baseline because (1) that work is specific for spaceships that can play the role of bosses which is comparable to the content of the case study, and (2) Gallota *et al.* achieves the best results of the state of the art for that content.

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

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

Each organ was transplanted individually to each boss. Each variant of Imhotep provided a total of 645 new individuals (5 hosts * 129 organs) as output, 645 new individuals from Simulation-based and 645 individuals from Test-based. In the case of the baseline (which relies on the L-system to generate the content instead of transplanting organs), to make it a fair comparison, the baseline was executed 129 times with each one of the 5 different hosts to obtain 645 new individuals. In addition, we executed 30 independent runs (to avoid not accounting for random variation as suggested by Arcuri and Fraser [21]). Hence, we obtain a total of 58050 independent runs (645*3*30).

To compare the solutions of the two variants of Imhotep (Simulation-based and Test-based) and the baseline, we rely on the concept of game quality and its automated measurement through simulated players. The results of Browne et. Al. demonstrated the validity of the approach which is accepted by the game research community [22]. Therefore, we need two ingredients: the simulated player and the automated measurement.

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

The automated measurement is QDuration which has proven to achieve good results [22]. The duration of duels between simulated players and bosses 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 ($T_{Optimal}$):

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

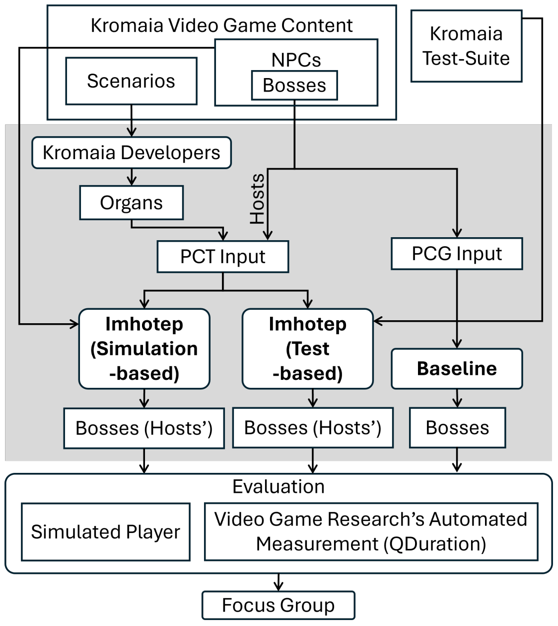


Fig. 11: Overview of the evaluation process.

C. Implementation details

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

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

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

In this section, we present the results obtained in the baseline and in the two variants of our approach (Imhotep) in Kromaia.

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.* [23].

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. In order to address the *lack of good descriptive statistics*, we present the standard deviation, min-max range and a box-plot from the results of the experiments realized.We also applied statistical significance (the Quade test and Holm's post-hoc analysis) and effect size measurements (Cliff's Delta) following accepted guidelines [21]. We tackled the *lack of a meaningful comparison baseline* by comparing our two variants of Imhotep with a recent PCG approach as baseline.

2. Internal Validity Threats. 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 a metric for the evaluation that is adopted and *validated* by the video game research community [22].

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. We can apply our approach to other video games where NPCs are available. These NPCs are 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. 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.

VIII. Related work

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

A. Automated Software Transplantation

On functionality transplantation, Miles *et al.* [24] and Petke *et al.* [25] 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.* [26] 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.* [27] 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.* [28] 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 [29] 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.* [29] use Gated Graph Neural Networks [30] to predict the correct variable name in an expression.

In 2018, Lu *et al.* [31] 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 [32] achieves the transplantation between different programs and programming languages.

We have seen so far that Automated Software Transplantation transplants within the same platform. However, Kwon *et al.* propose CPR [33] 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 [34] 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

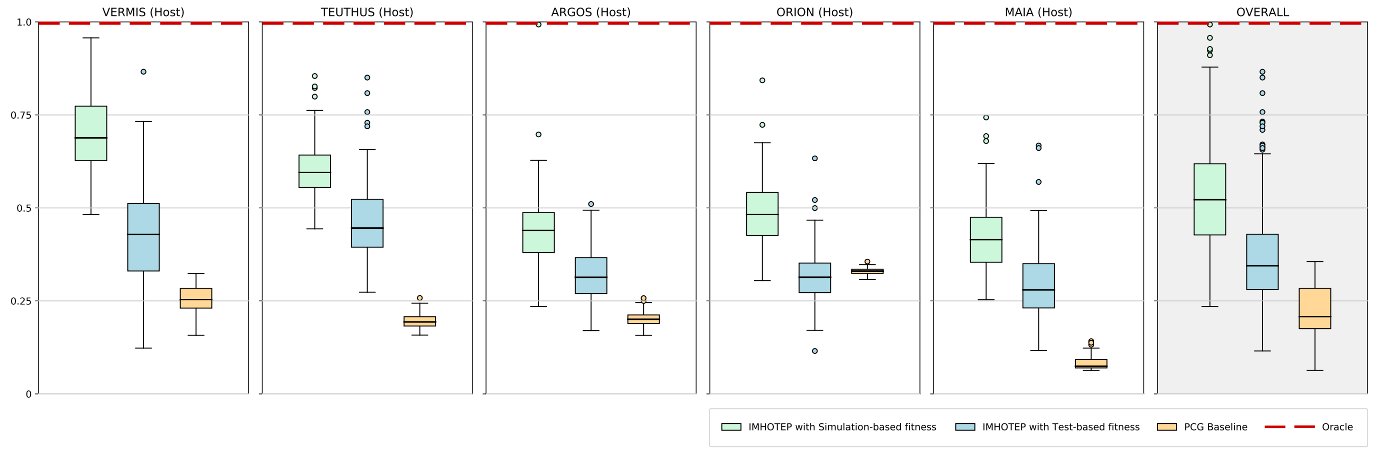


Fig. 12: Results

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

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 categories of content generated by PCG are diverse, such as vegetation [35], sound [36], terrain [37], Non-Playable Characters (NPCs) [38], dungeons [39], puzzles [40], and even the rules of a game [41]. PCG is a large field spanning many algorithms [42], which can be grouped in three main categories according to the survey of PCG techniques by Barriga et al. [43]: Traditional methods [44] that generate content under a procedure without evaluation; Machine Learning methods (PCGML) [45], [46], [47] that train models to generate new content; and Search-Based methods (SBPCG) [6], [48] that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

Our work generates content of the NPC category. In the context of NPC generation using SBPCG, Ripamonti *et al.* [49] developed a novel approach to generate monsters adapted to players, considering the monster with more death rate the preferred by the player. To evaluate the monsters, they recreated an environment with the main aspects from a MMORPG [7] game. Pereira *et al.* [50] and later extended by Viana *et al.* [38] seek for generating enemies that meet a difficulty criteria. Pereira *et al.* and Viana *et al.* use the same research academic game in their experimental designs. Blasco *et al.* [20] focusses on generating spaceship enemies that are comparable to the ones manually created by developers. To generate spaceships, Gallota *et al.* [15] used a combination of Lindenmayer systems [51] and evolutionary algorithm. Gallota *et al.* as well as Blasco *et al.* use a commercial video game in their evaluation.

TODO *cite any recent ML and NPCs, simplifications or academics?*

The motivation of our work comes from the limitations that we detected in previous work. Previous work focused on

[7]Massive Multiplayer Online Role-Playing Games

speeding up development time. However, the influence of the developers on the generated content was limited. The generated content depended on randomness resulting on generated content not aligned with the intention of the developers. As a result, the generated content was either not used or used as secondary content.

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

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] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

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

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.