Automated Transplantation for Procedural Content Generation in Video Games

Mar Zamorano, Carlos Cetina, Federica Sarro

***Abstract*—This document describes the most common article elements and how to use the IEEEtran class with LaTeX to produce files that are suitable for submission to the IEEE. IEEEtran can produce conference, journal, and technical note (correspondence) papers with a suitable choice of class options.**

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

I. Introduction

THE video games industry grows exponentially every year [1]. With every passing day, the demand for video games content keeps growing, with players requesting - and more often than not, expecting - more content than developers can produce. Content generation is a generally slow, tedious, costly, and error-prone manual process. In order to cope with the growing demand for novel, original content for video games, researchers alike are working towards procedural content generation. Procedural content generation (PCG) refers to the field of knowledge that aims at the automatic generation of new content within video games [2]. To this extent, PCG has explored different techniques that Barriga *et al.* [3] grouped in three categories: Traditional methods that generate content under a procedure without evaluation; Machine Learning methods that train models to generate new content; and Search-Based methods that generate content through a search on a predefined space guided by a meta-heuristic using one or more objective functions.

In this paper, we propose a new angle to tackle PCG inspired by transplantation techniques [4], that we named as Procedural Content Transplantation (PCT). By definition, *transplantation* is a procedure in which cells, tissues, or organs of an individual are replaced by those of another individual or the same person [5]. The transplantation core idea has been adapted to other research areas such us the Software research community. In Software we understand transplantation as a procedure in which a fragment of code (organ) of a program (donor) is transferred into another program (host). Within the Software research community, transplantation has been used in research areas such as, program repair, testing, security, or functionality improvements. We propose its novel use on the area of PCG.

Current approaches of PCG work as follow, developers provide initial content (usually human-generated content) into an algorithm to work with. Then the algorithm (Traditional, Machine Learning, or Search-Based methods) will generate new content. Only few traditional methods have succeeded in providing a tool used by the industry to generate (mainly random) vegetation (e.g., SpeedTree in Unreal and Unity). Our PCT proposal introduce the transplantation metaphor into this process. Developers from the human-generated content will select an organ (from a donor) and host that will receive the organ. The host and the organ will pass into the transplantation algorithm that will generate new content combining them. Our hypothesis is that transplantation provides more control to developers (identifying host and organ) leading to results that are closer to developers' expectations.

Our work propose a PCT algorithm that works with Search-Based Software Engineering. Search-Based Software Engineering has a better shot at exploring large search spaces than humans. In the state-of-the-art of software transplantation, the search evolves by genetic operations (crossover and mutation) and guided by a test objective function. On other hand, in the field of Search-Based Procedural Content Generation (SBPCG) in a common practice to guide the search is guided a simulation objective function. Taking into account the differences in software transplantation and SBPCG, we propose leveraging simulations to guide the transplantation algorithm, and we compare two different objective functions (tests and game simulations).

Another difference between the state-of-the-art of software transplantation and SBPCG is the representation of the problem. Software transplantation works directly over the source code of a program, and due to the nature of video games, in SBPCG is common the use of software models to treat the content. In this paper, we work with a representation closer to SBPCG using in our representation software models.

Our approach, called Imhotep[1], transplant software models through a search-based approach which uses crossover and mutation as genetic operations, and is guided by two objective functions (Test Suite and Simulation). Imhotep Test-based and Simulation-based variants are compared between them and a PCG Baseline [6]. We have done our evaluation using the Kromaia case study. Kromaia is a video game about flying and shooting with a spaceship in a three-dimensional space[2]. It was released on PC, PlayStation, and translated to eight different languages.

To evaluate Imhotep, 129 different organs extracted from the scenario of Kromaiaare transplanted into 5 of the video game bosses that act as hosts, generating new video game bosses in the process. In total, our approach works with 645 transplants. We evaluate the generated content through simulation that measures the quality of the generated content by means of

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

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

the duration of a match between the generated boss and a simulated player, a metric from the literature [7].

The results show that, out of the 3 objective functions, the content generated through game simulations obtains the best results for all the generation scenarios (that is, for the 5 bosses that act as hosts). The tests-based fitness obtains the second place, with the baseline fitness obtaining worse results than the other two in all scenarios. The generated results are promising, since they can be used as a starting point for the work of the developers. From the generated content, the developers can either include the generated content in the game, modify the generated content to better suit their needs, or inspect the generated content to find novelties from which they can create more original designs.

Our contributions are as follows:

- Novel application of Software Transplantation on Procedural Content Generation or PCT
- Software Transplantation of models instead of code
- Comparative of two objective functions, one based on Software Transplantation and the other PCG

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

II. Background

A. Game Software Engineering

Video games present characteristics that differentiate their development and maintenance from the development and maintenance of classic software; for example, how developers contribute to video games vs. non-games by working on different kinds of artifacts (e.g., shaders, meshes, or prefabs). In addition, game developers perceive more difficulties than other non-game developers when locating bugs as well as reusing code [50].

Nowadays, most video games are developed by means of so called game engines. A game engine refers to a development environment that integrates a graphics engine and a physics engine as well as a set of tools that wraps around them in order to accelerate development. The most popular ones are Unity [3] and Unreal Engine [4], but it is also possible for a studio to make its own specific engine (e.g., CryEngine [5]).

A key artifact of game engines are software models. Developers can create video game content directly using code (e.g., C++) or the software models of the engines. On the one hand, the code allows developers to have more control over the content. On the other hand, software models are much less bound to the underlying implementation and technology and raise the abstraction level using terms that are much closer to the problem domain. This means that developers are liberated 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). Unity and Unreal propose their own modeling language, and a recent survey in Model-Driven Game Development [51] reveals that UML and Domain Specific Language (DSL) models are also adopted by development teams.

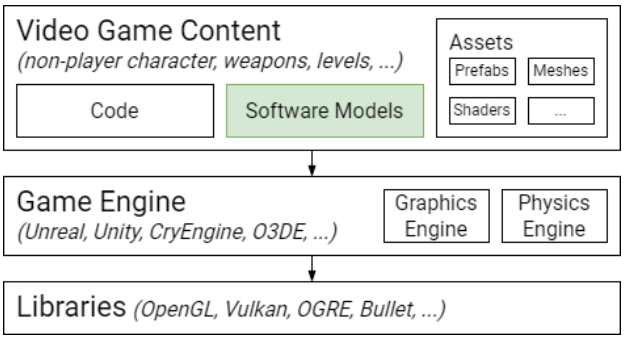

Fig. 1: Overview of video game artifacts. Highlighted in green Software Models, where this work focus.

B. Case Study: Kromaia

The case study that we use to evaluate the work presented here is performed using the bosses of the video game Kromaia. The game in Kromaia takes place in a three-dimensional space. Each of the levels involves a player's spaceship flying from a starting point to a target destination reaching the goal before being destroyed. The level involves exploring floating structures, avoiding asteroids, and finding items along the route, while protected by basic enemies that try to damage the player's 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.

The bosses are specified with the Shooter Definition Model Language (SDML). SDML is a DSL model for the video game domain. This DSL follows the main ideas of MDE using models for Software Engineering. The models are created using SDML and interpreted at runtime. Specifically, SDML defines aspects that are included in video game entities: the anatomical structure (including which parts are used in it, their physical properties, and how they are connected to each other); the amount and distribution of vulnerable parts, weapons, and defenses in the structure/body of the character; and the movement behaviours associated to the whole body or its parts. This modeling language has concepts such as hulls, links, weak points, weapons, and AI components. Examples of the models, the metamodel, and an online visualizer to show the models as they would be seen in the Kromaia video game can be found at the following URL: https://svit.usj.es/models22/bl-in-mgse. TODO *Modify link*

The simulations used in this work simulate a duel between a boss and a human player. The simulated player is an algorithm that is able to act like a human player. It was created by the developers of the Kromaia video game. We used their algorithm for our approach. During the simulation, the simulated player

[3] https://unity.com/
[4] https://www.unrealengine.com/
[5] https://www.cryengine.com

faces the boss in order to destroy the weak points that are available at that moment, whereas the boss acts according to the anatomy, behaviour, and attack/defense balance that is included in its model, trying to defeat the simulated player. In the simulation, both the boss and the simulated player try to win the match and do not avoid confrontation, try to prevent draw/tie games, and try to ensure that there is a winner. The algorithm can fight a boss by applying different strategies. Hence, the algorithm can be parametrized to define the fighting strategy. The simulation parameters were provided by the developers based on the analysis of battles between human players and bosses.

It is important to clarify that bosses can be built either using SDML software models or directly with C++. The intuition of the developers is that they make fewer mistakes and are more efficient working with the models than with the code, and an experiment confirmed this [15]. MAR *check if the clarification of models and code is in other part, if not adapt it. The part of the mistakes should not be relevant to our work.*

III. Our Imhotep Approach

This section will explain how our approach make use of evolutionary computation to transplant elements within video games content. Figure 2 shows our Imhotep approach that we will explain in detail during this section. We will also make use of a theoretical example of auto-transplantation of content within a simplified version of ”bosses” of the video game Kromaia.

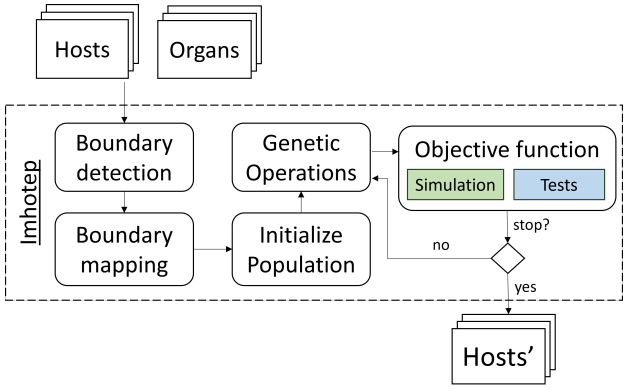

Fig. 2: Overview of Imhotep

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-B). The donor and host from the example are different from the donor and host used in the evaluation, but we think they will help to understand the approach. The donor, organs and host used by Imhotep

For the theoretical example we present a simplified version of the metamodel, and its corresponding concrete syntax of the model (Fig. 3) from the video game Kromaia. In this example we use a graphical representation to help the comprehension of the reader. Notice that 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.

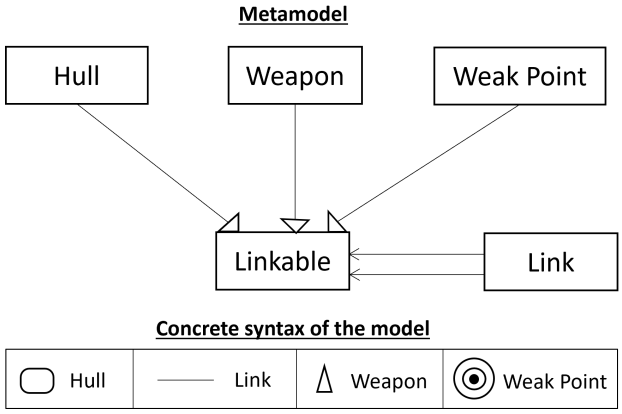

Fig. 3: Simplified metamodel with its corresponding concrete syntax of the model

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. 4). The original model is a SDML model written on a XML file with approximately 1700 lines of code. Figure 4 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 on Kromaia following the metamodel (Fig. 5.

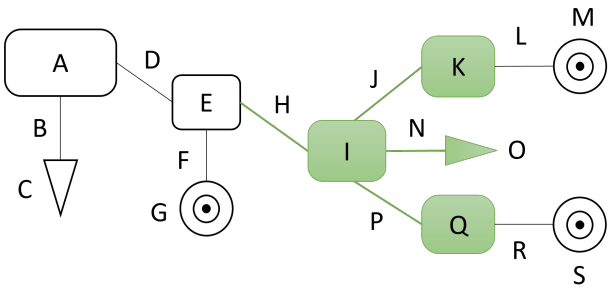

Fig. 4: Donor with organ selection in green. The letters represent each element of the model.

B. Boundary detection

To transplant an organ into a host we need to find a way to connect them. To do that we will use the boundaries of the organ and the host. A boundary is a connection point on an element capable of connect two elements within a model. The connection is restricted by the rules of the metamodel. Imhotep identifies the boundaries of the selected organ, and all

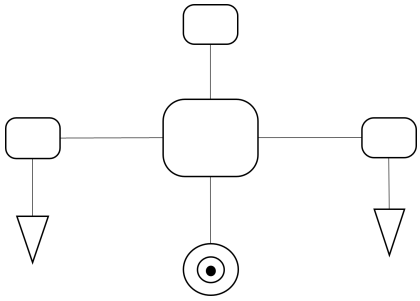

Fig. 5: Host

the boundaries of the host before initializing the evolutionary algorithm.

Following on 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 6 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.

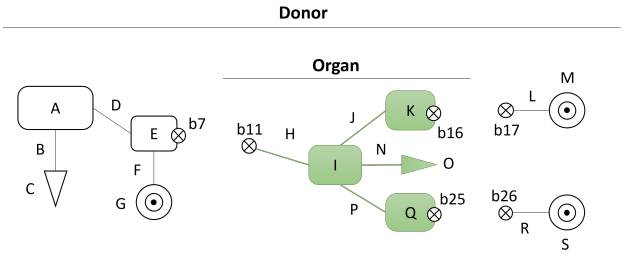

Fig. 6: Definition of organ boundaries. The boundary is represented by a circle crossed.

On the other hand, the boundaries of the host will be all the points where its elements connect. Figure 7 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.

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 the 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.

D. Encoding

Te adjunto el encoding que había originalmente, al que habría que añadir una fila nueva ”transplant root hull index”:

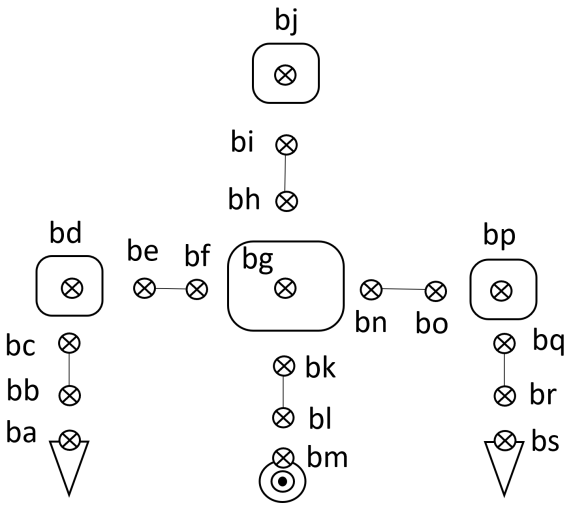

Fig. 7: Definition of host boundaries. The boundary is represented by a circle crosed.

Organ boundary	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

en ella todo son guiones ”-” (representa -1), salvo en los cascos que están ahí porque han sido implantados. En todos ellos aparecerá un número/carácter que representa el índice del casco del receptor por el que el empalme se ha realizado, mediante link. En el otro archivo que te paso, el gusano (64 cascos originalmente) tiene trasplantado un brazo que tiene 13 cascos. El primero de esos trece está unido al número 14 del gusano (en la fila de links, el casco 65, el primero correspondiente al contenido trasplantado, tiene una ”e”(14). Ésa ”e” es la que luego ves en la última fila, en las columnas correspondientes al contenido trasplantado.

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 crossover between them.

TODO *how crossover work?*

The new individual then have 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. 8 shows handmade new individuals that could results from our example.

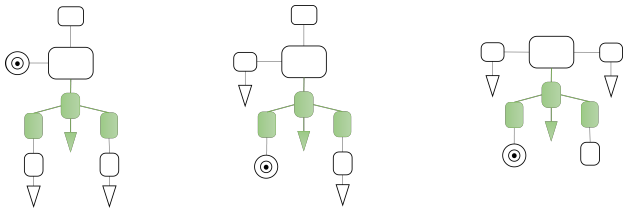

Fig. 8: Population individuals to be evaluated by the objective function

F. Objective function

The objective function in Imhotep assess the quality of each individual as a model. First, as done in previous work of that use the case study Kromaia [40], 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 passes the validation process will then be assess by Test-based and Simulation-based objective function. 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 a total of 243 test selected under they domain knowledge. The objective value will be retrieved when each individual pass through the 243 tests normalized in a scale of [0, 1] written this limitation. An individual which pass the 243 tests will obtain 1.0, and on the contrary if it does not pass any test will obtain 0.0.

On the other hand, the Simulation-based objective function as in Blasco *et al.* [40] 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.* [40]:

$F_{Victory}$ 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 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. In the end, $F_{Overall}$ is a value in [0, 1] that 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. Evaluation

In this section we explain how we evaluate the feasibility of automated transplantation in video games through our Imhotep approach. Our experiment compares Imhotep with the two different objective functions, Test-based and Simulation-based, and a PCG baseline.

The PCG baseline that we use is... **TODO** *explain the baseline*

We transplant on 5 different host, original bosses from the video game Kromaia. We have 129 donors, that are elements from the scenario of the game, called "Totems". We obtain a total of 645 new host' from Simulation-based and 645 new host' from Test-based. **TODO** *for the baseline we force 129 repetitions per initial boss, so we obtain 645 new host as well*

The execution time for each transplantation is 2 minutes and 30 seconds.

We run the experiments in two laptops with the following specifications; Intel Core i7-8750H, 16GB, 2.2GHz; and 2x Intel(R) Xeon(R) CPU X5660, 64GB, 2.80GHz.

We evaluate each host' with the following quality measures:

A. Quality measurements

In a recent research done by Browne et al., the experimentation with game users showed that the following criteria stand out as being the most important: Completion, Duration, Uncertainty, Killer Moves, Permanence, and Lead Change [7]. Our evaluation measures these criteria with values in the interval [0,1].

Completion (Viability): A game against a boss unit should end with more conclusions (victories for either the player or

the boss) than draws/ties. The criterion $Q_{Completion}$ calculates a ratio of conclusions over total duel count:

$$Q_{Completion} = \frac{Conclusions}{Duels} \quad (4)$$

Duration (Viability): 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 ($T_{Optimal}$):

$$Q_{Duration} = 1 - \frac{\sum_{d=1}^{Duels} \frac{|T_{Optimal} - T_d|}{T_{Optimal}}}{Duels} \quad (5)$$

Uncertainty (Quality): In order to keep players engaged with a duel, neither the player nor the boss unit should get extremely close to victory or defeat too early before the duel is settled, with (T_d) being its duration. Therefore, a duel is considered to be more uncertain the longer the time until the player's or the boss unit's health levels reach a dangerous/critical status (P_d and B_d, respectively). For each duel, $Q_{Uncertainty}$ measures the average deviation between the time at which it is detected that one of the contenders is on the verge of defeat and the time corresponding to the duration of the duel.

$$Q_{Uncertainty} = 1 - \frac{\sum_{d=1}^{Duels} \frac{T_d - min(P_d, B_d)}{T_d}}{Duels} \quad (6)$$

Killer Moves: Q_{KMoves} measures the proportion of killer moves by any contender (K), taking into account the moves that are considered to be remarkable highlights (H) but that are less important than killer moves. In the video game case study, the developers considered that a highlight move happens when either the boss unit or the player experiences a decrease in health; killer moves are those that make the difference in health between the contenders reach 30%.

$$Q_{KMoves} = 1 - \frac{\sum_{d=1}^{Duels} \frac{K_d}{H_d}}{Duels} \quad (7)$$

Permanence: Duels with a high permanence value are games in which the advantages given by significant actions or moves by one of the contenders are unlikely to be immediately reverted by the opponent in terms of dominance. In the video game case study, the developers considered every highlight move and killer move to be meaningful actions, with recovery moves (R) being those that quickly cancelled the advantages given by other previous killer or highlight moves. The criterion $Q_{Permanence}$ is measured as follows:

$$Q_{Permanence} = 1 - \frac{\sum_{d=1}^{Duels} \frac{R_d}{H_d + K_d}}{Duels} \quad (8)$$

Lead Change: The lack of lead changes indicates low dramatic value. In the video game case study, the lead is determined at any given moment by considering the contender with the highest health level. This criterion is measured taking into account those highlight or killer moves that cause the lead to change (L) during the course of a duel:

$$Q_{LChange} = \frac{\sum_{d=1}^{Duels} \frac{L_d}{H_d + K_d}}{Duels} \quad (9)$$

$Q_{Overall}$ calculates an average quality value for a model, including all of the quality criterion studied:

$$Q_{Overall} = \frac{\sum_{i=1}^{N} Q_i}{N} \quad (10)$$

V. Results

Figure 9 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 of the host for each objective funtion and the baseline. In addition, the oracle indicates the value obtained by the human-generated final boss models that were obtained from the 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 9 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

VIII. Related work

This work is about generating original content in video games using an automated software transplantation approach. Our evaluation is in the context of the video game content of Kromaia. Therefore, our approach generates new content of Kromaia. In this section, we discuss: (1) work that address automated software transplantation; (2) work that address

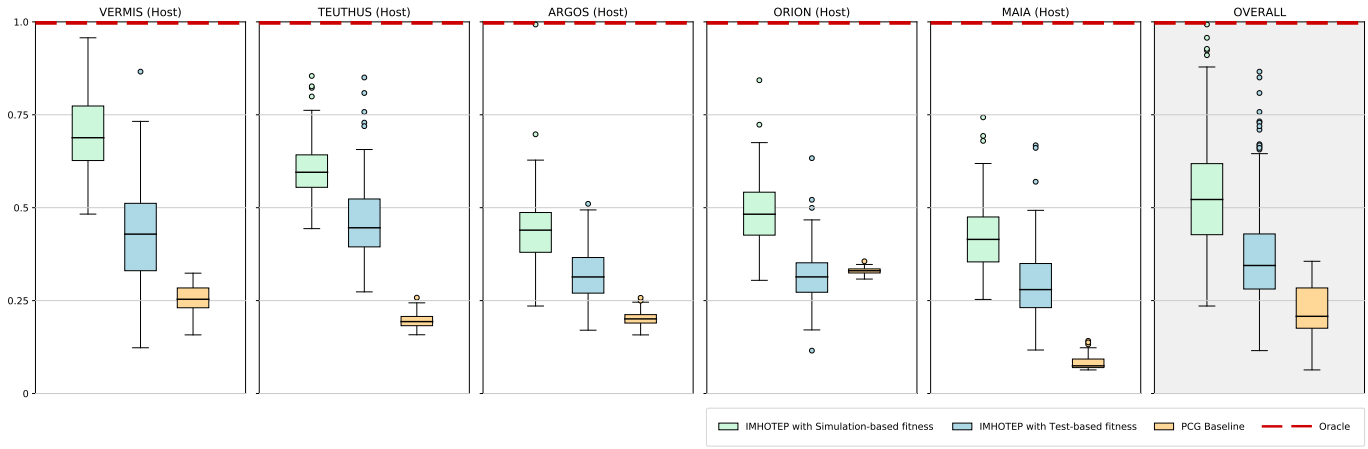

Fig. 9: Results

video game content generation; and (3) work that address game software engineering from the MDE community. Video game content generation is also know as procedural content generation in the literature. Finally, we present an analysis of the research gap.

A. Automated Software Transplantation

Automated software transplantation address the challenge of automatically transferring a fragment of code, an organ, from one program, called donor, into another program, called host.

In the literature, automated software transplantation has been used in different research areas, such as, program repair [8], [9], testing [10], security [11], or functionality improvements [12]. However, most of the literature in software engineering use software transplantation to repair or improve the functionality of the host, with code from the donor that was previously missing in the host [13].

Miles *et al.* [14] or Petke *et al.* [] proposed approaches that transplant in the same program, taking into account that different versions of the programs are considered the 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. In addition, the work from Sidiroglou-Douskos does not use adapters either. Sidiroglou-Douskos *et al.* [15] 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.

On the other hand, Maras *et al.* [16] 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.* [17] 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 transforms the types from the desired type signature into the type signatures of the candidate functions.

Allamanis *et al.* SMARTPASTE [18] presents a different strategy to adapt the organ into the host. SMARTPASTE takes the organ and replace variable names with holes, then using a deep neural network it fills the holes. Allamanis *et al.* [18] use Gated Graph Neural Networks [19] to predict the correct variable name in an expression and to identify when a variable name is missuses.

In 2018, Lu *et al.* [20] introduced program splicing, a framework to automate the process of copy, paste, and organ modification. In their approach, unlike Allamanis *et al.* which puts holes into the organ, the host is provided with a draft of code with holes, or natural language comments. Like Wang *et al.*, Program splicing looks into a database of programs to identify a relevant code to the current transplant task. Finally, the approach select the more suitable result found to fill the holes in the draft.

μSCALPEL [4] is an automatic code transplant tool that uses genetic programming and testing to transplant code from one program to another. μSCALPEL using test cases to define and maintain functionality, 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 [13] allows the transplantation between not only different programs but also between different programming languages.

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

B. Procedural Content Generation

Procedural Content Generation (PCG) refers to the automation or semi-automation of the generation of content in video games [2]. The types of content generated by PCG are diverse, such as vegetation [23], sound [24], terrain [25], Non-Playable Characters (NPCs) [26], dungeons [27], puzzles [28], and even the rules of a game [29].PCG is a large field spanning many algorithms [30], which can be grouped in three main categories according to the survey of PCG techniques by Barriga et al. [3]: Traditional methods [31] that generate content under a procedure without evaluation; Machine Learning methods (PCGML) [32], [33], [34] that train models to generate new content; and Search-Based methods (SBPCG) [2], [35] 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 [35]: 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 by 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 different version of the NPC. From the best of our knowledge we are the first work applying transplantation in PCG. Guarneri *et al.* [36] or Norton *et al.* [37] 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.* [38] 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.* [39] and later extended by Viana *et al.* [26] instead of diversity seek for generating enemies that meet a difficulty criteria.

Our work uses the same case study as Blasco *et al.* [40] 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.* [6] used a combination of Lindenmayer systems [41] and evolutionary algorithm.

C. MDE and Game Software Engineering

One of the challenges in software development is the environment used, as each environment and programming languages has unique characteristics. Software models, and more precisely Model Driven Engineering, study how to alleviate this problem by approaching software development from a platform-independent perspective through models. Video game developers must deal with this challenge as well and has motivated the research that combine software models and the domain of video games.

The 2010 survey of Software Engineering Research for Computer Games [42] identified only one work that applied Model-Driven Development to video games [43]. That work coined the term "Model- Driven Game Development" and presented a first approach to 2D game prototyping through Model-Driven Development. Specifically, they used UML classes and state diagrams that were extended with stereotypes, and a model-to-code transformation to generate C++ code.

More recent work presents work that intended to minimize errors, time, and cost in multi-platform video game development and maintenance [44], [45], [46], or suggest the use of business process models as the modelling language for video games [47].

In the intersection between software models and evolutionary computation, Williams *et al.* [48] use an evolutionary algorithm to search for desirable game character behaviours in a text-based video game that plays unattended combats and that outputs an outcome result. The character behaviour is defined using a Domain-Specific Language. The combats are managed internally and are only driven by behaviour parameters, without taking into account a spatial environment, real-time representation, or visual feedback (which takes into consideration the physical interaction of the characters, variation in the properties, etc.).

Another work that focuses on the intersection between software models and evolutionary computation is Avida-MDE [49], which generates state machines that describe the behaviour of one of the classes of a software system (Adaptive Flood Warning System case study). The resulting state machines comply with developer requirements (scenarios for adaptation). Instead of generating whole models, Avida-MDE extends already existing models (object models and state machines) with new state machines that support new scenarios. The work in Goldsby and Cheng *et al.* [49] does not report the size of the generated state machines; however, the ones shown in the paper are around 50 model elements, which is significantly smaller than the more than 1000 model elements of the models of a commercial video game such as Kromaia.

The work mentioned above focus on generating new content from models, which differs with our proposal of using MDE to transplant model fragments between models.

D. Conclusion

Our work open differs from previous work in different aspects. To the best of our knowledge is the first paper

addressing automated software transplantation if the field of video games. Our proposal allows the transplantation different types of content. More precisely, we transplant elements from a scenario to an NPC. The use of MDE separate the problem from the platform, and even the specific video game, as a same Domain Specific Language can be used in different video games.

IX. Conclusion

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] 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.

[3] 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.

[4] 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.

[5] 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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: A systematic review," *Information and Software Technology*, vol. 52, no. 9, pp. 888–901, 2010.

[43] E. M. Reyno and J. Á. Carsí Cubel, "Automatic prototyping in model-driven game development," *Computers in Entertainment (CIE)*, vol. 7, no. 2, pp. 1–9, 2009.

[44] E. R. Núñez-Valdéz, V. García-Díaz, J. M. C. Lovelle, Y. S. Achaerandio, and R. G. Crespo, "A model-driven approach to generate and deploy videogames on multiple platforms," *J. Ambient Intelligence and Humanized Computing*, vol. 8, no. 3, pp. 435–447, 2017. [Online]. Available: https://doi.org/10.1007/s12652-016-0404-1

[45] E. R. Núñez-Valdéz, O. S. Martínez, B. C. P. García-Bustelo, J. M. C. Lovelle, and G. Infante-Hernandez, "Gade4all: Developing multi-platform videogames based on domain specific languages and model driven engineering," *IJIMAI*, vol. 2, no. 2, pp. 33–42, 2013. [Online]. Available: https://doi.org/10.9781/ijimai.2013.224

[46] M. Usman, M. Z. Iqbal, and M. U. Khan, "A product-line model-driven engineering approach for generating feature-based mobile applications," *Journal of Systems and Software*, vol. 123, pp. 1–32, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2016.09.049

[47] J. Solís-Martínez, J. P. Espada, N. García-Menéndez, B. C. P. García-Bustelo, and J. M. C. Lovelle, "VGPM: using business process modeling for videogame modeling and code generation in multiple platforms," *Computer Standards & Interfaces*, vol. 42, pp. 42–52, 2015. [Online]. Available: https://doi.org/10.1016/j.csi.2015.04.009

[48] J. R. Williams, S. M. Poulding, L. M. Rose, R. F. Paige, and F. A. C. Polack, "Identifying desirable game character behaviours through the application of evolutionary algorithms to model-driven engineering metamodels," in *Search Based Software Engineering - Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings*, 2011, pp. 112–126. [Online]. Available: https://doi.org/10.1007/978-3-642-23716-4_13

[49] H. J. Goldsby and B. H. C. Cheng, "Automatically generating behavioral models of adaptive systems to address uncertainty," in *Model Driven Engineering Languages and Systems*, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 568–583.

[50] 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.

[51] 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.