Research Article

Pegasus: A Simulation Tool to Support Design of Progression Games

Marcelo Arêas R. da Silva ¹ and Geraldo Bonorino Xexéo^{1,2}

¹COPPE/UFRJ, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

²Department of Computer Science, Institute of Mathematics (DCC-IM), Rio de Janeiro, RJ, Brazil

Correspondence should be addressed to Marcelo Arêas R. da Silva; mareas@gmail.com

Received 8 February 2018; Revised 22 October 2018; Accepted 28 October 2018; Published 2 December 2018

Academic Editor: Michael J. Katchabaw

Copyright © 2018 Marcelo Arêas R. da Silva and Geraldo Bonorino Xexéo. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The process of designing a game involves many phases. We can summarize the work of the game designer as satisfactorily converting the idea in their mind to a digital game, which is not a simple task. Therefore, game designers should have a variety of tools to assist them. However, there are not that many specialized tools to support the game design process. Herein, we describe the experience of using Pegasus to design a part of a game. We propose an environment to simulate progression games based on game design patterns. Thus, we described the interaction of the game designer with Pegasus in such an environment, in order to support the process of creating, testing, and refining game elements before proceeding to the programming phase. Each configuration of the game elements corresponded to a simulation that could be performed multiple times, like in discrete event simulation. The results showed that Pegasus has the potential to support game design. Additionally, we presented some support components that were created to facilitate the use of the tool.

1. Introduction

The digital games industry is growing fast. A survey conducted annually in the United States indicated that consumer spending on digital games increased 42% from 2010 to 2016. The same research indicated that there is at least one person in every United States household who plays three or more hours of digital games a week [1]. There was a time when people thought that videogames were aimed only at children, but that is a fallacy; there are studies that characterize the profile of older players [2].

Despite such growth, the process of creating a digital game can be as complex as creating an information system. There are many steps—from designing the initial idea to reaching the final product—involved in conceiving a digital game. Several authors have dealt with this subject [3–7]; some for specific types of games (e.g., serious games [8, 9]), but few have attempted to support the design process by using formal methods [10]. In fact, as stated by Neil [11] and Koster [12], there is a lack of tools to support game design.

If the game designer could test scenarios and refine the game prior to the programming phase, it is reasonable to say

that product quality would improve and time would be saved. Thus, this paper is part of a thesis [13] and we propose an environment to simulate games that have a progression-based structure.

This paper is organized as follows: Section 2 describes some related works in this research area; Section 3 presents our proposal of an environment to simulate progression games; Section 4 introduces the Pegasus simulation tool; Section 5 reports on the experience using the simulator to support the design of part of a game; and finally, Section 6 presents the conclusions and future work.

2. Literature Review

The *Machinations* framework is a way to represent games by highlighting their internal economy, focusing on the relationships of the game mechanics and the dynamic gameplay that emerges from these relationships [14]. It works as a visual language, composed of diagrams and components, in which the game designer can create, simulate, and test the internal game economy. *Machinations* is satisfactory for simulating

TABLE 1: Patterns in game design and their relationships.

Pattern	Instantiates	Instantiated by
Game world	-	Levels
Levels	Game world	-
Inaccessible areas	Traversing	-
Enemies	Combat	Boss monsters, Avatars, Units, Elimination
Boss monsters	Enemies	Elimination
Obstacles	Inaccessible areas	-
Avatars	Enemies	-
Units	Enemies, Resources	-
Tools	Improved abilities, Rewards	Pick-ups
Pick-ups	Tools	Power-ups, Resources
Power-ups	Pick-ups, Improved abilities	-
Lives	Resources	-
Resources	Pick-ups, Rewards, Penalties	Units, Lives
Combat	Randomness, Resources	Elimination, Enemies
Movement	-	Traversing
Improved abilities	Rewards	Power-ups, Tools
Rewards	Balancing	Improved abilities, Tools
Penalties	-	Elimination, Resources
Elimination	Combat, Penalties, Enemies, Boss Monsters	-
Traversing	Movement	-
Randomness	Balancing	Combat
Balancing	-	Rewards, Randomness

emergence in games, but is not suitable for progression games. Dormans [15] tried to shape emergent mechanics to produce progressive experiences; however, due to limitations of *Machinations*, only an evolution of the same structure of the emergence pattern was achieved to represent the constant increase in difficulty.

The simulation technique is often used in several areas, for example, analysing production processes, optimizing transport networks, evaluating user behaviour, and learning [16, 17]. In the study of Nummenmaa et al. [18], the authors proposed simulating gameplay as a design tool. They described a case study with implementation of a simulation model for a game using a simulation software package.

Evaluating player enjoyment is also a common theme in this area of research; there are works presenting models for analysing game playability [19, 20]. Syriani and Vangheluwe [21] proposed an approach to improve game playability by simulating the movement of characters with simple mechanics as well as by evaluating player behaviour.

Even if it was just a theoretical foundation, Grünvogel [10] introduced formalism for designing games from small sub-models, due to complex games being difficult to simulate. Gradually, those models would be combined into more complex systems to represent the game as a whole.

2.1. Progression and Emergence in Games. Among the many genres and subgenres of digital games, a game is classified into these different genres depending on the observer [22]. However, according to Juul [23], we can say that there are two basic structures of games: emergence and progression.

In a game structured by emergence, a small set of rules combine to produce large game variations, and players then create strategies to deal with the random world [23]. We found emergence in strategy, card, and sports games, as well as in chess and tournaments. On the other hand, progression deals with a controlled sequence of events and often tells a story. The game designer plans each level and its challenges. The player must overcome the events in a given sequence to advance in the game [23]. In general, adventure and actions games are based on progression structure.

Most modern digital games fit the two definitions; that is, they contain elements of emergence and progression [14]. For example, *Assassin's Creed*, *Red Dead Redemption*, *GTA*, *The Division*, and *Destiny* are open world games, in which a player controls a character with freedom to explore and perform various activities. However, there are optional and main missions; the latter usually unlock new places when completed. These games contain a progression structure and tell a story, but despite this, each mission has its own emergent elements.

3. Proposal of Environment to Simulate Progression Games

Björk and Holopainen [24] presented a large collection of patterns in game design. We studied patterns and their relationships in order to propose a favourable environment for simulating progression games. Table 1 shows the relationships among some patterns commonly found in games with a progression-based structure.

FIGURE 1: The simulation environment.

FIGURE 2: Actions of game designer.

3.1. Types of Progression Games Covered. The proposed environment supports games with elements that are typical of role-playing and adventure games and could involve single-player or multiplayer games. The player controls a character or a team, starts from point A, faces some challenges, and tries to reach point B alive. During the journey, the player visits several places, which may contain enemies, items, and/or obstacles. Places can be mandatory or optional, which results in many possible ways to reach the end. The degree of complexity depends on the player's creativity in combining elements to create the game world.

Games of the *Dungeon Crawl* genre—such as *Rogue*, *Gauntlet*, *Dungeonlike*, and *Diablo*—fit the described characteristics very well. Furthermore, the proposed environment is also suitable for any game which involves entering a place, resolving conflicts, collecting items, and moving on.

3.2. Description of the Environment. The environment is composed of a *Simulation Engine*, *Input Artifacts*, and *Output Artifacts*; see Figure 1. The *Input Artifacts* are *Team*, *World*,

and *Simulation Settings*. *Team* is made up of *Characters*, which can be *Heroes* or *Foes*. *Places* are arranged inside the *World* and may contain *Teams* of *Foes* and collectible *Items*.

The *Simulation Settings* should allow the game designer to adjust and test distinct scenarios to obtain different results. *Output Artifacts* are data collected—in an easy-to-read format—from all of the simulation events performed. Thus, data could be processed to generate valuable information, reports, statistics, summarization, and graphs.

The diagram in Figure 2 illustrates how the game designer should act in the environment. The inspiration of a game designer remains fundamental. The process starts after the game designer constructs the *Input Artifacts* to feed the simulator; we call this *Initial Design*. The designer then starts a test cycle, performing simulations (*Simulations*), analysing the results (*Analysis*), and, if applicable, adjusting some elements and settings to start new simulations (*Refinement*). Once the game designer approves the results, the participation of the simulator is concluded (*End*), and everything is set to move forward to the programming phase of the creation process.

FIGURE 3: Simulation Engine requirements.

3.3. Simulation Engine Requirements. The operations of the *Simulation Engine* must fulfill certain requirements to ensure the proper functioning of the proposed environment (Figure 3). The first step is to receive the *Input Artifacts*, then to restore all of them to their initial conditions. Thereafter, with the *Hero* or *Team of Heroes* at the first place, the conflict should be solved in the following order:

- (i) *Preconflict*: preparation for the conflict; *Heroes* can use *Items* to improve or restore attributes.
- (ii) *Conflict*: resolve the threats—such as battles—at the particular place.
- (iii) *Postconflict*: *Heroes* collect pick-up *Items* as rewards, if they exist.

Heroes then move to the next place, and the aforementioned process starts again until one of the following happens:

- (a) *Heroes* are in the final place or
- (b) all *Heroes* are dead.

In both cases the simulation ends.

During a simulation, the *Simulation Engine* must record all data from each event as it happens. Such events can be battle logs; how the values of each attribute of the characters

are modified, items collected, items used, keys found, etc. At the end of a simulation, all the collected data should be summarized.

The described process comprises a single simulation. At this point, the next simulation can be performed or, if it was the last one, the *Simulation Engine* consolidates the results from all previous simulations and provides the data. In the subsequent sections we show some examples of processed data.

4. Pegasus: Progression Game Simulator

We built Pegasus to implement our proposal. It was constructed in the Python programming language [25] to comply with best practices for software design patterns [26]. In this section, we present the architecture, related artifacts, and support components.

4.1. Architecture and Input Artifacts. Pegasus architecture is divided into layers (see Figure 4), as follows:

- (a) *Interface*: where all the processes are initiated. The *Input Artifacts* are forwarded to the subsequent corresponding layers.
- (b) *Component*: collection of essential and nonessential modules that facilitate usage and give new configuration options to the game designer. It is referred to as the *Component* layer because new modules can be built and incorporated into it.
- (c) *Simulation*: the simulation process core was developed respecting the guidelines of our proposal.
- (d) *Result Storage*: it stores all simulation results; it works concurrently with the *Simulation* layer.
- (e) *Result Processing*: it generates reports and graphs from the collected data.

Grey arrows represent the sequence of events, whereas white arrows indicate that a layer acts upon another.

The *Input Artifacts* are class objects representing the game elements and simulation settings. Figure 5 shows a class diagram with the relationships.

A *Character*, which can be either a *Hero* or *Foe*, is part of a *Team* and has a set of *Attributes* and *Items*. The *Attributes* define the kind of action a *Character* is capable of. There are four distinct types of *Attributes*, all of which affect and are affected by battles: *life*, *attack*, *defense*, and *agility*. The *life* attribute defines if a *Character* is still alive; thus, the absence of this attribute means that a *Character* does not participate in battles. The *attack* attribute is essential to a *Character* inflicting damage on others. The *defense* attribute is the capacity to reduce damage suffered. Finally, the *agility* attribute influences how fast a *Character* can perform actions when in battle.

The *Places* can have *Teams*, *Items*, or nothing. The *Places* connect to each other; a *Team* of *Heroes* tries to find a way from the first to the final *Place*. The *Items* may be weapons or potions for temporarily or permanently improving some of the *Attributes*; for example, a key to a *Place* that would

FIGURE 4: Pegasus architecture.

FIGURE 5: Class diagram for Input Artifacts.

otherwise be inaccessible. This key/lock system is frequently found in a large range of games.

Once these elements are prepared, they are all encompassed by the World.

The *Simulation Settings* comprise a set of options to guide the behaviour of some components. More details are introduced in the subsequent subsections.

4.2. Output Artifacts. The *Result Storage* layer records all events that have occurred in the simulations, thus satisfying

what was described in our proposal. Subsequently, the *Result Processing* layer generates summary and detailed reports. Figure 6 contains snippets of some reports as examples.

Both the *Summary Report* (a) and the *Detailed Report* (c) contain data from each simulation, composed of the places entered, number of turns in each battle, and items found. At the end of each simulation there is a summary with the following: total number of places visited, names of all items found, number of the place where the simulation ended, total number of turns, average number of turns, and the order of the places visited. The *Detailed Report* (c) also contains logs of each battle; the values of the attributes vary according to each battle.

In each report, the consolidated results (b) contain information on all simulations and are shown after all simulations, at the end of the report. They state the number of times the *Heroes* concluded a simulation, average number of places visited, average number of items found, average number of turns per simulation, average number of turns per place, and the most common order of the places visited.

One of the modules developed in the *Component* layer acts on the *Result Storage* layer to generate graphs from the collected data. Currently, the following graphs can be generated:

- Bar chart and pie chart for the number of times a simulation ends in a place.
- Scatter plot and Box plot for the variation in a *Hero* attribute at each place, for all simulations.
- Line chart for the average value of the attributes of every *Hero* at each place.
- Scatter plot for the variation in number of turns at each place, for all simulations.
- Line chart for the most common order of places visited.

```

***** STARTING SIMULATIONS *****

*** Starting Expedition #1 ***

Entering Place #1
Battle in Place 1 ends after 18 turns

Entering Place #2
Battle in Place 2 ends after 67 turns
Item(s) found:[keyPlace4 (KEY).
Qty:1]

Entering Place #3
Battle in Place 3 ends after 4 turns

End of Expedition #1. Summary:
Heroes LOST
Places visited: 3
ItemsFound: keyPlace4
Ended in place: 3
Total turns: 89
Turns avg: 29.67
Visit order: 1, 2, 3

```

(a)

```

-----Simulation report after 5000 expeditions:-----
Heroes successes: 1600 (32.0%)
Heroes failures: 3400 (68.0%)
Avg visited places: 4.05
Avg items found: 1.49
Turns avg per expedition: 85.88
Turns avg per places: 24.5
Most common visit order: 1, 3, 5. Per 1250 times (25.0%)

```

(b)

```

Entering Place #3
Heroes data:
Name: Knight. Life:71. Attrib: Sword(ATK):10. Shield(DFS):8. Agi(AGI):10.
Name: Wizard. Life:25. Attrib: Wand (ATK):12. Wand(DFS):6. Agi(AGI):7.
Enemies data:
Name: Goblin 1. Life:25. Attrib: Knife(ATK):6. Dfs(DFS):2. Agi(AGI):10.
Name: Goblin 2. Life:25. Attrib: Knife(ATK):6. Dfs(DFS):2. Agi(AGI):10.
Name: Hunter. Life: 30. Attrib: Axe(ATK):15. Shield(DFS):6. Agi(AGI):7.
Hunter attacks Wizard. Atk pw:20. Dfs pw:7.6 Dam:12.4. Wizard life:12.6
Knight attacks Goblin 2. Atk pw:16. Dfs pw:3.6. Dam:12.4. Goblin 2 life:7.6
.
.
.
Battle in Place 3 ends after 29 turns
No Items

```

(c)

FIGURE 6: Simulation reports: (a) summary report, (b) consolidated results, and (c) detailed report.

- (f) Bar chart for the order in which the different places were visited.
- (g) Pie chart with the percentage of successes and failures of the *Team of Heroes*.

4.3. Support Components. We created an XML format in order to create the elements *World*, *Place*, *Character* and

Item, as well as an XML reader to read files with this format and to create the corresponding objects. As this reader was not a part of the simulator, we preferred to put it here. The next components are part of the *Component* layer. One of them is the module to support graph generation, which was introduced in the previous subsection. The other modules operate according to the options set in *Simulation Settings*.

TABLE 2: Foes and items at each place.

Place	Foes	Items
1	4 goblins	-
2	5 goblins, 1 orc	Silver key
3	3 goblins, 1 troll	-
4	6 goblins	Defense shield
5	2 goblins, 1 orc, 1 troll	Life potion
6	2 orcs	Silver lock, life potion, and sword
7	Warlock (Boss)	-

Here we show a list with the names and options available for each component developed so far:

- Movement strategy*: next place, random place, farthest place, prioritized unvisited places.
- Strategy for choosing character to attack in battle*: pick randomly, pick character with highest value for a particular attribute, and pick character with lowest value for a particular attribute.
- Strategy for choosing character to benefit from item*: same as previous.
- Battle mechanics*: alternating turn system; active time battle system.
- Randomness generator*: choose minimum and maximum bonus value for attacking and defending.
- Output settings*: set the path on the computer to save the *Output Artifacts*.
- Graph generator*: inform the relationship of the graphs to be generated.

Pegasus's architecture was designed to permit the addition of new options. These options are relatively simple and only the first ones created. There are other studies in research areas related to the topics covered in these options that could be adapted and incorporated into our simulation system.

5. Results

In this section we report on the experience using Pegasus to support the design of a level of an adventure type of game. In this game, a group of three heroes with different abilities need to reach the end of the map and defeat the boss who is there. On the way they may encounter some foes and collectible items.

The imagined level has seven different places, and all of them have some enemies to confront. Place 2 has a key. If the heroes do not possess this key it is not possible to enter place 6. Place 4 has a shield to improve defense. Places 5 and 6 have potions to restore life, and place 6 has a sword to improve attack. These places are part of the world, as Figure 7 shows. Table 2 shows the enemies and items at each place, and Table 3 contains the chosen settings for the simulations. The attributes of the characters were not included in order to keep the tables brief.

We performed 10,000 simulations in a computer with an Intel Core i5 processor (5th generation, 8 GB RAM), which

TABLE 3: Simulation settings.

Setting	Option chosen
Movement strategy	Random place
Character to attack in battle	Randomly picked
Character to benefit from item	Randomly picked
Battle mechanics	Active time battle system
Randomness generator for attack	Min 0% and max 60%
Randomness generator for defense	Min 0% and max 30%

FIGURE 7: World map for the adventure game.

took 9 min 48 s. The following are the consolidated results of the Summary Report: heroes' successes: 985 (9.85%); heroes' failures: 9015 (90.15%); average number of places visited: 6.31; average number of items found: 3.37; average number of turns per simulation: 205.09; average number of turns per place: 35.47; and most common order of places visited: 1, 3, 4, 5, 7—1292 times (12.92%). We assembled some interesting graphs in Figure 8.

Upon analysing (d), we could see that the heroes arrived at the final place 70.5% of the time, which seems to be a good number. However (a) and (c) indicate that they arrived with a low life value.

There are several ways to change the balance of the game. Due to the heroes being successful only 9.85% of the time, we wanted to increase the success rate, but without changing the attributes of the final boss. In order to do this, we thought of some options like adding new power items to the map or decreasing some attribute of the enemies. In the end, we chose to decrease the number of enemies at Place 5, due to it being the place where the battles lasted the longest—see (b)—and where a big drop in life value occurred; see (a) and (c).

Thus, in the new configuration, we removed one goblin and one troll from Place 5. After another 10,000 simulations (in 10 min 4s), the new results were as follows: heroes' successes: 4469 (44.69%); heroes' failures: 5531 (55.31%); average number of places visited: 6.65; average number of items found: 4.11; average number of turns per simulation: 199.65; average number of turns per place: 32.56; and most common order of the places visited: 1, 3, 4, 5, 7—1280 times (12.8%). The new graphs are shown in Figure 9.

Now the heroes reached the final place almost every time (97.4%); see (d). We can see in (a) and (c) that the curve from place 4 to place 5 became smoother. Also, the number of turns at each place decreased, as we had planned.

We considered the results of this iteration to be satisfactory; therefore, we finished the refinement phase. However, a

FIGURE 8: Simulation results from the first iteration: (a) average life at each place, (b) number of turns at each place, (c) life of the Amazon Hero at each place, and (d) place where simulation ended.

FIGURE 9: Simulation results from the second iteration: (a) average life at each place, (b) number of turns at each place, (c) life of the Amazon Hero at each place, and (d) place where simulation ended.

new cycle could be started in order to test other configurations. This process can be repeated extensively until an ideal configuration producing the expected results is achieved.

6. Conclusions and Future Work

In this paper, we proposed an environment to support the design of games with a progression-based structure. We studied the well-known patterns in game design, and highlighted the ones typically found in the kind of games in which we were interested. We introduced Pegasus—a simulation tool created according to our proposal—and described its architecture and how it operates. We also developed some support components that can be improved and expanded.

Pegasus showed potential to be a valuable tool in assisting game designers with their tasks. The ability to test the variation of scenarios—even during the design phase of a game's production—is a great benefit.

In our tests, we noticed that varying the strategy for choosing the character to attack in battle can produce different results to those using characters picked randomly. We believe that the *random* strategy is the most appropriate for evaluating what the behaviour of an ordinary player would be. Moreover, adjusting the strategy gives an idea of how players with more skill could behave. Similarly, it is possible to test different strategies for the behaviour of enemies. Thus, this is a useful feature to discover the best configurations for multiple difficulty levels of the same game.

For future work, we intend to create new modules as well as new options for the existing modules of the *Component* layer, starting with a new battle system. We also intend to make some improvements so that the designer could prepare more than one scenario, which would lead to the simulations producing collections of results.

Data Availability

The complete results are in a compact format, available to download at <https://www.dropbox.com/s/amg0bdllv190s3l/Pegasus%20experiment%20data.rar?dl=0>.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)), Finance Code 001.

References

- [1] Entertainment Software Association, *Essential Facts About the Computer and Videogame Industry*, Washington, DC, USA, 2017.
- [2] B. De Schutter, "Never too old to play: The appeal of digital games to an older audience," *Games and Culture*, vol. 6, no. 2, pp. 155–170, 2011.
- [3] C. Crawford, "The art of computer game design," *Osborne/McGraw-Hill*, 1984.
- [4] J. Schell, *The Art of Game Design: A Book of Lenses*, vol. 54, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
- [5] A. Rollings and D. Morris, *Game Architecture and Design: A New Edition*, New Riders Publishing, Indianapolis, Indiana, 2004.
- [6] E. Bethke, *Game development and production*, Wordware Publishing, Inc., Plano, Texas, USA, 2003.
- [7] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*, vol. 37, The MIT Press Cambridge, Cambridge, UK, 2004.
- [8] A. F. S. Barbosa, P. N. M. Pereira, J. A. F. F. Dias, and F. G. M. Silva, "A New Methodology of Design and Development of Serious Games," *International Journal of Computer Games Technology*, vol. 2014, Article ID 817167, 8 pages, 2014.
- [9] F. Bellotti, R. Berta, A. De Gloria, A. D'ursi, and V. Fiore, "A serious game model for cultural heritage," *Journal on Computing and Cultural Heritage*, vol. 5, no. 4, pp. 1–27, 2012.
- [10] S. M. Grünvogel, "Formal models and game design," *Game Studies: The International Journal of Computer Game Research*, vol. 5, no. 1, 2005.
- [11] K. Neil, "Game design tools: Time to evaluate," in *Proceedings of the 2012 International DiGRA Nordic Conference*, 2012.
- [12] R. Koster, "A Theory of Fun 10 years later," in *Proceedings of the Game Developers Conference*, 2012.
- [13] M. A. R. da Silva, *PEGASUS: Uma Ferramenta de Simulação para Apoio ao Design de Jogos de Progressão*, Universidade Federal do Rio de Janeiro, 2018, <https://www.cos.ufrj.br/index.php/pt-BR/publicacoes-pesquisa/details/15/2850>.
- [14] J. Dormans, *Engineering Emergence: Applied Theory for Game Design*, Universiteit van Amsterdam, 2012.
- [15] J. Dormans, "Integrating emergence and progression," in *Proceedings of the 2011 DiGRA International Conference: Think Design Play*, pp. 1–17, 2011.
- [16] J. A. Sokolowski and C. M. Banks, *Principles of Modeling and Simulation: A Multidisciplinary Approach*, John Wiley & Sons, Inc, 2008.
- [17] B. D. Ruben, "Simulations, games, and experience-based learning: The quest for a new paradigm for teaching and learning," *Simulation & Gaming*, vol. 30, no. 4, pp. 498–505, 1999.
- [18] T. Nummenmaa, J. Kuittinen, and J. Holopainen, "Simulation as a game design tool," in *Proceedings of the International Conference on Advances in Computer Entertainment Technology - ACE '09*, pp. 232–239, Athens, Greece, October 2009.
- [19] P. Sweetser and P. Wyeth, "Gameflow: A model for evaluating player enjoyment in games," *Computing and Entertainment*, vol. 3, no. 3, p. 3, 2005.
- [20] J. L. G. Sánchez, F. L. G. Vela, F. M. Simarro, and N. Padilla-Zea, "Playability: analysing user experience in video games," *Behaviour & Information Technology*, vol. 31, no. 10, pp. 1033–1054, 2012.
- [21] E. Syriani and H. Vangheluwe, "Programmed graph rewriting with time for simulation-based design," in *Theory and Practice of Model Transformations*, A. Vallecillo, J. Gray, and A. Pierantonio, Eds., vol. 5063, pp. 91–106, Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 2008.
- [22] A. Järvinen, *Games without Frontiers: Theories and Methods for Game Studies and Design*, University of Tampere, Finland, 2009.

- [23] J. Juul, “The Open and the Closed: Games of Emergence and Games of Progression,” in *Proceedings of the Computer Games and Digital Cultures Conference*, 2002.
- [24] S. Björk and J. Holopainen, *Patterns in Game Design*, vol. 54, no. 3, Charles River Media, 2005.
- [25] Python, *Python Software Foundation*, 2018, <http://www.python.org>.
- [26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1995.