Mario AI: A Level Generation Framework

Shahriar Kabir Khan (190784331), Bikash Kumar Tiwary (190778031) and Jaemin Kim (180806577)

School of Electronic Engineering and Computer Science, Queen Mary University of London, UK

{md.khan, b.k.tiwary, j.kim}se19.qmul.ac.uk

Abstract. This paper describes the use of procedural content-generating algorithms using Markov chain process. The algorithm is a hybrid of the constructive and learning-based algorithms which generate levels for Mario AI framework. Our algorithm attempts to learn what makes a "good" map from a set of original maps, then uses those learned patterns to generate new maps. Experiments are performed on map generation, simulation-based evaluation and multi-agents performance on multiple levels.

Keywords: game AI, benchmarking, procedural content generation, Mario, platform games

1 Introduction

Procedural content generation (PCG) is generally referred to automatically created content for games e.g. levels, puzzles, art, and animation, etc using algorithms. Some of the example games used PCG are Rouge (1980), Micropose (1991), Diablo (Blizzard 1996), Gearbox (2009), etc[18]. The procedural generation of video games existed for approximately 33 years, but there was no machine learning approached applied to generate levels without specifying rules in the early stage of PG[10]. Academics and researchers have taken different approaches to implement machine learning techniques e.g. planning and constraint solving, human author tile placement[15]. Both of these approaches require a method for evaluating the generated levels or the rules of generation.

Also, while doing computational and/or AI research work, it is important to select the right game to apply the right algorithms. This applies for both, whether the research is conducted for games improvement or to improve AI systems. As a steadily developing field, one AI game will not satisfy all the research criteria/algorithms to experiment e.g. procedural content generation, player satisfaction, interesting enough to play, etc [7]. Additionally, the implemented game benchmark should be developed for major computer platform with future workability in mind and availability for all other platforms. The system should be easy to install, the API and code should be modifiable without having advance programming knowledge, etc [7].

In this study, we have described our experimentation on the Mario AI framework to generate content using a hybrid of constructive and learning-based algorithms along with Markov chains. Besides, we have also discussed previous research work-related to PCG and Mario games used AI techniques, a short introduction on original Mario games, approach for training Markov chains, search-based, and learning-based algorithms, etc, implementation of PCG, evaluation of our work using various methods and future work below in this paper.

2 Literature Review

Super Mario Bros., is a videogame developed by the Nintendo Entertainment System in 1985. It is one of the popular games used in research to develop intelligent agents using an AI system, which also includes generated levels. Mario was hugely popular achieving almost 40 million sales and it has reached the best selling video games rank (fifth) shortly after it released in the game platform. With this huge success, another four versions were developed respectively with various contents. In 2009, a Mario AI competition was introduced for the first time with the intelligent agent to complete difficult game levels based on Infinite Mario Bros (a different version of Mario games). In the Infinite Mario games, levels generated automatically which makes it difficult to complete a level. This championship was continued until 2013, however, it has stopped due to the favour of AI competition platformer[2].

For this research work, the Mario AI framework (also a well-known platform game) has been considered as one of the most suitable benchmarks for game AI research. A brief introduction of platform games, the Mario AI framework and level generated methods are described below.

2.1 Platform Games

Platform games generally described as a type of game where a game character is controlled by the player, the game character jumps or runs to avoid obstacles and defeat enemies. Platform games can be categorized into two types; Scrolling and single-screen platformers.

Single screen platform games displayed on a single static screen to players with multiple levels of varying difficulties. Example of single-screen platformer games is Burgertime, Donkey Kong, The Fairyland Story, etc[19]. Besides, in the scrolling platformer games, the game character moves towards the screen edge to reach the finish flag/sign. It also can have multiple levels of difficulties to increase player interests and create challenges for players. Super Mario Bros is a scrolling platformer game, another example is Castlevania Sonic the Hedgehog. The 1980s, 1990s, and 2000s can be considered as a golden age of platform games, however, in 2010s to onwards the rise of smartphones, 3d technologies, pcs and consoles had faded away from the popularity of platform games. However, for researchers and academics, platform game is very popular for accessibility and open-source platform [19].

2.2 Mario AI framework

Mario AI framework is a clone of 'Infinite Mario Bros.' version of the Mario video game series. Nonetheless, the benchmark is actually cloned from Nintendo's platform game 'Super Mario Bros.', available to play on the web and the source code written in Java also available online. A player can control the game character/Mario in front of a moving 2D sideway objects and moves forwards to reach the goal. Mario moves right and left, can jump when a hole or an obstruction in front to get past[6]. In some states, Mario can also shoot fireballs. Mario can be in three states, begins with 'small' states starting of the game, other states are 'Big' (when crushes an enemy or objects by jumping over them/it), and 'Fireball states'. The main goal of the game is to reach the end of the game, however, optional rewarding goals are scoring high by collecting items, kill enemies, and reach to the end of the game as fast as possible[7]. Although, unlike original 'Super Mario Bros.', Mario AI generates level automatically.

The Mario AI framework is first released in 2009, for this study we have used the most updated Mario AI version known as the tenth-anniversary edition with several new features. Some of the improved features are the better interface than the previous version, 11 different agents in total, faster framework, better interface, human-readable level files, etc [5]. Super Mario Bros and Maria AI framework can be seen in figure -1 below.

Figure 1: Mario AI Framework (left) and Super Mario Bros. (Right)

2.3 Level Generator Methods

PCG refers to the automatic level generation for the games but it is not fully automatic often. The most common PCG approach is the search-based approach, use the stochastic global search/optimization algorithms or use evolutionary algorithms for searcing content space. Often the difference in PCG is between control and Variation. Methods in which output has high variation usually have little afford of a designer. While variation generated artefacts with expressive range, the control uses for style, player experience, game playability or level of difficulty [3].

However, the categories mentioned above are not complete or exclusive as there are hybrid methods available and some methods can not be categorized to any of the methods mentioned above[13]. For instance, an approach was implemented in 2009 to generate a platformer map based on the rhythm and the aim was to achieve all the available action a player can take[11]. A brief description of how the level generator method can be implemented using the Markov chain process is described in the section below with a short description of the procedural map generation.

3 Background

There are several techniques and algorithms available for the Game AI fields with various useability. Some of the most widespread algorithms are Finite state machine, Utility-based AI and behaviour trees under the wing of Ad-hoc learning. Besides, supervised, unsupervised, reinforcement learning, evolutionary algorithm and tree search are some other most renowned approaches available for Game AI[1].

For many reasons, game developers interested in PCG. Memory consumption is one of them, other reasons reduce the expenses of making content manually, the possibility of emerging completely new game type, etc[17].

However, none of the algorithms suitable for one particular game to experiment or there is not a technique implemented using only one algorithm. Therefore, at the beginning of this study, we explored the Markov chain process with LSTM(Long Short-Term Memory recurrent neural networks) sequence generation in our mind[15].

3.1 Markov Chains

A method for modelling probabilistic transition between states. Markov chain is a set of states S = {s1, s2, … sn} and the conditional probability distribution (CPD) P(S(t) |S(t−1)), which represents the transition probability to a state S(t) ∈ S given that previous state was S(t-1) ∈ S [15]. A two-dimensional Markov chain is considered as a Multi-dimensional Markov chain (MDMCs) in this study. Multi-dimensional Markov chains (MDMCs), have also shown to produce qualitative texture outputs, at the same time reducing the sampling complexity remarkably [9].

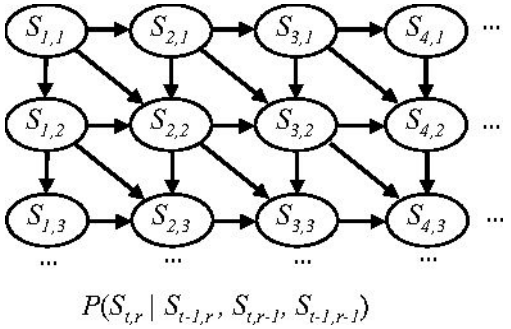

Figure 2: Two-dimensional Markov chain of order three. (three incoming dependence arrows of each node)[14].

Markov chain is a model used by machine-learned platformer to generate levels. Markov chain generally uses two major approaches (i) tile-to-tile transitions and (ii)

vertical slices of the level. In 2014, the vertical slice has been used for the first time by Dahlskog et al. (2014)[4], later a year it was used by Summerville et al. (2015)[15]. In the Summerville et al. (2014) method, the Markov chain state is treated as a vertical slice and transitions are learned from every slice. The advantage of encoding vertical structures directly is that, some tiles are more likely to be up/at the top (e.g. empty tiles) and other tiles will be at the bottom (e.g. in the ground). Nevertheless, it has a serious disadvantage that prevents the possibility of generating new vertical slices. Another problem is that state space can be much larger than the number of tiles, which makes the transition space much smaller than just the transition from tile to tile learning.

On the other hand, Snodgrass et al. (2013, 2014)[13] have used the first approach (i) tile-to-tile transitions in 2013 and 2014 respectively. However, they have faced other issues, for example, placing the bottom tiles at the top and vice-versa. Several attempts have been made to reduce this problem by considering different chains for different heights. The approaches mentioned above also have the risk of no playability. Snodgrass et al. (2013) designated agent cannot complete more than half the production level. Summerville et al. (2015) used the search for Monte Carlo trees as a guide to building Markov chain levels to implement playable levels. In addition, the different application of Markov chains also suffers the same drawbacks of no playability issue[15].

3.2 Procedural Map Generation

Procedural content generation (PCG) is used for creating map contents algorithmically instead of manually as mentioned earlier in section 2.3. PCG methods can be used to create game components like maps [19], textures and other visual game components[18].

Generally, PCG based approaches are classified into these broad categories: Search-based, constructive, learning-based, and tiling. Search based methods generally produce map contents by exploring a defined space using a search technique along with an evaluation method. Few PCG search based methods include Evolutionary Methods, Constraint Solving, Machine learning, and among others. Constructive methods put together different components of maps in terms of respecting rules and constraints, following specific logic and automating map generation. The learning-based approach extract maps from previous maps, generating novel content using those learned models. Tiling approach build map content from smaller parts referred to as “tiles” algorithmically, using different tile sizes and assembly techniques [15].

Procedural content generation is not mutually exclusive to these categories. A hybrid method or approach have been used in this study with constructive and learning-based approaches. Also, methods that do not fit any of the above categories like in the case

of the rhythm-based 2d categories, an approach was developed to generate platform maps based on a rhythm that the player feels with his hands while playing[12].

4 Techniques Implemented

In this section of the report, the techniques implemented in the Mario level generator has been described briefly. The main algorithm used in this study is the Markov chain, as discussed in the previous section. And the program is developed and tested using Python language 3.7.

4.1 Map Representation

To represent the Mario map, the 2-Dimensional matrix is used. Each element of a matrix corresponds to one tile, and it takes the integer value. Because the representation of the input and output level is a character, the process to convert into ASCII (American Standard Code for Information Interchange) integer code is required, and vice versa.

Figure 3 shows the characters and ASCII code to represent the map. The left is game, multiline characters array for input and output of the game (middle) are in the middle of the figures, and the last one is the matrix representation used in the program.

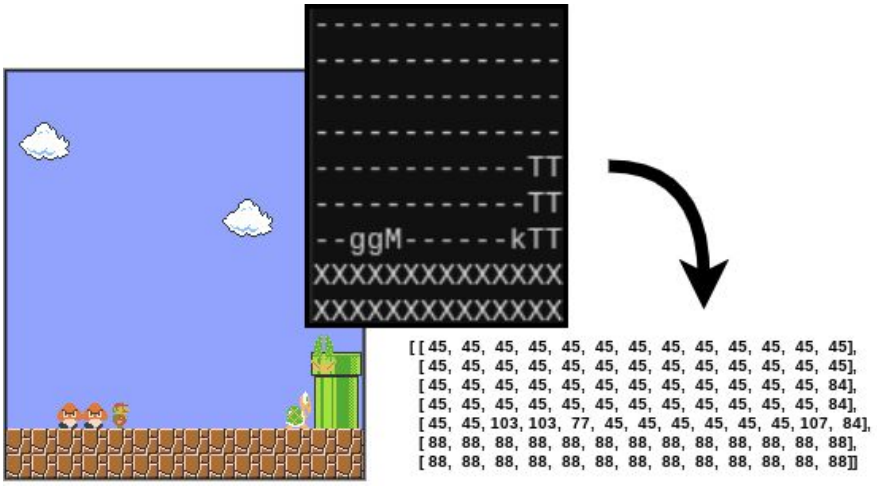

Figure 3: a map presentation in the game (left), and a multiline string array used in the level file (centre), and a 2-dimensional array of map used in the python code (right).

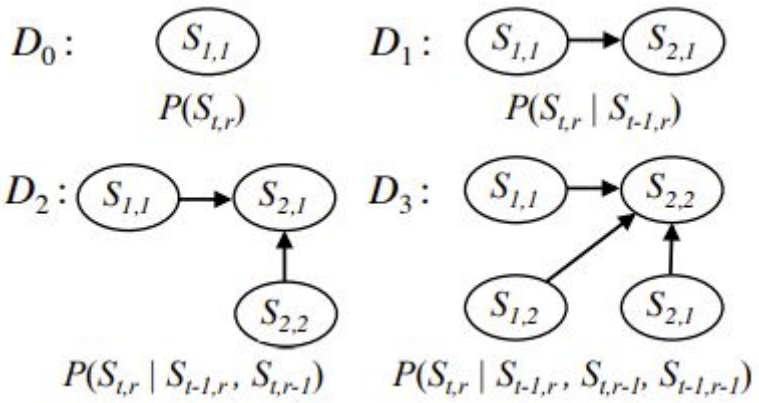

Figure 4: The dependency structures used for the Markov chain. D1, D2, D3 are used for this project. [14].

4.2 Dependency Matrix

The concept of a Markov chain to generate the map was motivated by 2d approach of the map generation[15]. Figure 4 shows dependency structures, which are elaborated in the Snodgrass et al. (2013) paper. Originally, the number of suggested matrices was four, but only three structures are used for this project. The usage of each structure is described in the Learning Phase section.

Firstly, to store the dependency states, the representation is required, which is a string type. Each state is the equal-sized block consisted of tiles, so it needs to be converted into a string type, as shown in Figure 5, before saving. And the next thing is the data structure. Hashtable is suitable to store the dependency information with neighbour blocks.

The previous states of multi-order dependency structures are represented as a combined matrix and this matrix saved as the string type into the hashtable as the key item. The value item of hashtable also should be the same type with key items. However, in order to save value and probability information together, the hashtable is used as the type of the value item instead of a string type.

4.3 Mario Level Generator

The generator is divided into 2 parts. One is the learning phase and another is the generating phase.

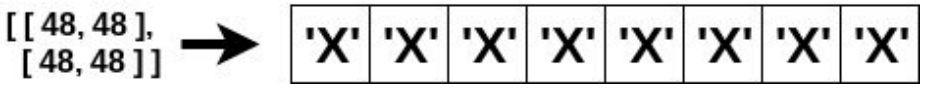

Figure 5: Conversion matrix to string type.

For the learning phase, the bottom line and left border are firstly trained after splitting into equal-sized blocks. This is because the player starts from left and ground is the important tile to play in the game. So dependency states are training and generating from left to right and from bottom to top.

There are five variables of dependency tables. One is for the bottom line, another is for the left border and the rest three variables with different three order dependency structures are for whole blocks. These three order dependency structures have different relations among the neighbour blocks. Weak relation like one order dependency structure implies the uncertainty, which could lead to the diversity of map. However, this structure can generate the odd block. As a result, the map becomes unplayable.

In the generating phase, likewise learning phase, it starts with the left border and bottom line. And , for generating blocks, The strongest tables are first used and if there is no block to produce, move to the next weak table.

4.4 Block size and Diversity

By changing the block size value, the diversity of the map changes. Increasing the value of block size decreases the diversity of the generated maps resulting in close to the original map being generated. So, when the block size = 1, the generated map has high diversity. And big block size means reducing the samples to be learned. Therefore providing enough map samples are important.

5 Experimental Study

In this section, we have described different types of experiments on the implemented system as explained in section 4(Implementation) and documents the outlined results in the section below.

5.1 Experimental setup - 1

Different map parameter setup: Blocksize

When Blocksize = 1

Figure 6 - Generated map when the block size is value is 1

Figure 7 - Generated map when the block size is value is 1

Figure 6 and figure 7 represents the map generated by setting the block size to value 1. Low blocksize setting results in higher creativity and diversity in a level generation. Increase in difficulty level. Additionally, lowering the playability probability of the level generated.

When Blocksize = 3

Figure 8 - Generated map when the block size is value is 3

Figure 9 - Generated map when the block size is value is 3

Figure 8 and figure 9 represents the map generated by setting the blocksize value to 3. This blocksize setting results in considerable freedom in creativity. Lowering the difficulty. Increasing the playing probability of the level generated.

When Blocksize = 7

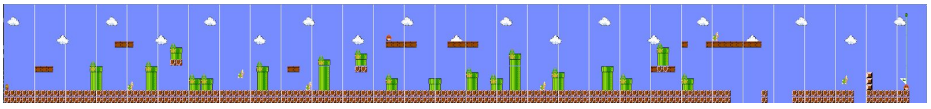

Figure 10 - map generated by setting the blocksize value to 7.

Figure 11 - map generated by setting the blocksize value to 7.

Figure 10 and figure 11 represents the map generated by setting the blocksize value to 7. The block size setting results in very low freedom in creativity keeping the level generated close to original, further lowering the difficulty. And Increasing the playing probability furthermore for the generated level.

As we can notice, the increase in block size decreases the creativity in the generated level map, lowering the difficulty.

5.2 Experimental setup - 2

Simulation-based Evaluation

In this experimental setup, different AI agents are playing on the same level. Based on the agent experience a level is considered good if the smarter agents perform better and less intelligent agents perform worse, with successful random plays being discouraged.

When Blocksize = 4

Figure 12 - Level 1 map generated by setting the blocksize value to 7

Agents -->	Andy	Random	Robin (smart)	michal	sergey Karakovskiy
Level 1	Time out (>20sec)	lose	win	Timeout (>20sec)	lose

Table-1: AI agents performance on the same level (i.e Level 1)

As shown in Table 1, Robin the smarter AI agent has performed well and won, whereas Andy and michal reached the time limit in the level within 20sec. Random and sergey Karakovskiy performed worse, losing in the generated level.

Based on the performance of the agents, this level can be considered as a good level.

When Blocksize = 7

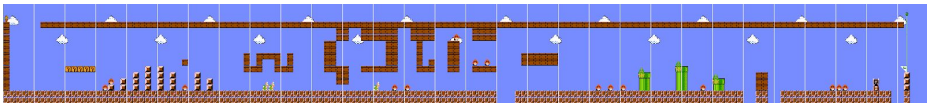

Figure 13 - Level 5 map generated by setting the blocksize value to 7

Agents -->	Andy	Random	Robin (smart)	michal	sergey Karakovskiy
Level 1	win	lose	win	Timeout (>20sec)	win

Table 2: AI agents performance on the same level (i.e Level 2)

As shown in Table 2, Robin the smarter AI agent, andy and sergey Karakovskiy have performed well and won, whereas michal reached the time limit in the level which is 20sec. Random performed worse, losing in the generated level.

Based on the performance of the agents, this level can be considered as a good level. As the performance by smarter agents are better at lower difficulty where the generated level is close to the original.

5.3 Experimental setup 3

Different AI agents playing different levels multiple times, and their completion rate.

When Block size = 4

Agents (10 games)	Win	Timeout	Lose
Robin	9/10	0	1/10
Random	0	1	9/10
Andy	0	4	6/10

Table 3: Performance of AI agents on multiple levels (Level 1 to 10)

Robin, Andy, and Random AI agents are made to play 10 levels, where the completion rate for the smartest agent is high, followed by the less intelligent agent, along with successful random play being the worse.

Hence the performance in table 3, specifies the playability probability of level maps at this difficulty.

Level	Robin	Random	Andy
1	win	lose	lose
2	win	lose	Timeout
3	win	lose	lose
4	win	Time out	Timeout
5	win	lose	lose
6	win	lose	lose
7	win	lose	lose
8	win	lose	Timeout
9	lost	lose	lose
10	win	lose	Timeout

Table 4: Agents performance at each level

As seen in table 4, we can state that generated level 9 is not playable.

6 Discussion

As details of the experiment work have been outlined above in the section (Experimental Study). it can be concluded that most of the experimental outputs fulfilled our expectations. ‘GroupQ’ level generator algorithm generated the levels in Mario AI framework by learning from original maps. Additionally, the algorithm was able to generate playable maps as shown above in section 5(experimentation) with different setups like blocksize.

In terms of underperformance, generated maps had odd unreachable content like bad pipes, misplaced grounds, etc. Furthermore, implementation of real-time inspection did not work which resulted in not being able to avoid unexpected content placements all over the map. Sometimes trapping the agent on generation.

One of the surprising aspects of the ‘GroupQ’ level generator algorithm was the successful generation of playable maps with the smallest blocksize value i.e blocksize = 1, which was unexpected due to high diversity and very low control in undesirable content generation and their placements.

7 Conclusions and Future Work

This report represents a comprehensive discussion and comparative study on the performance of GrougQ level generator experimented on Mario AI framework. The inspiration for GroupQ level generator was derived from '2d video game map generation'[15]. GroupQ algorithm is a hybrid of constructive(rules and constraints-based) and learning-based algorithm which uses Markov chains. The conclusion which can be drawn is that GroupQ algorithm can generate a level in Mario AI by learning from original levels. The undesirable section on maps might affect the playability of a level. Additionally, increasing the block size decreases the creativity in the generated level.

This study was taken under a limited timeframe. Multitudes of possible algorithms were analyzed as an approach for the level generation framework in Mario AI. Finding the right approach and implementation proved to be a hectic task. With uncertainty about code complexity and undesired outcome. Eventually, we decided to implement two-dimensional Markov chains with a hybrid of constructive and learning-based approaches.

In future work, it might be fascinating to use MCTS with two-dimensional Markov chains where using heuristic or evaluation function, we can restructure the maps undesirable part with the help of backtracking.

Alternatively, a hybrid of search-based and learning-based algorithms can be used along with two-dimensional Markov chains which might solve, for example, the undesirable sky blocks or misplacements. It might also solve the undesirable sections of the map produce making it fully playable with access to every part of the level.

Another future task may include implementation of real-time inspection in the current level generator algorithm, with this approach we can backtrack during the map generation to allow for change if an undesirable state is reached.

References

1. Aiolli, F. and Palazzi, C.E. (2019). Enhancing Artificial Intelligence in Games by Learning the Opponent's Playing Style. IFIP *International Federation for Information Processing*, [online] pp.1–10. Available at: https://link.springer.com/chapter/10.1007%2F978-0-387-09701-5_1 [Accessed 30 Nov. 2019].

2. Baldominos, A., Saez, Y., Recio, G. and Calle, J. (2015). Learning Levels of Mario AI Using Genetic Algorithms. *Advances in Artificial Intelligence*, [online] pp.267–277. Available at: https://link.springer.com/chapter/10.1007/978-3-319-24598-0_24 [Accessed 2 Dec. 2019].

3. Dahlskog, S. (2019). *Procedural Content Generation Using Patterns as Objectives*. [online] Proceedings of EvoGames, part of EvoStar. Available at: https://www.academia.edu/6268697/Procedural_Content_Generation_Using_Patterns_as_Objectives [Accessed 3 Dec. 2019].

4. Dahlskog, S. and Togelius, J. (2014). Procedural Content Generation Using Patterns as Objectives. *Applications of Evolutionary Computation*, [online] pp.325–336. Available at: https://link.springer.com/chapter/10.1007/978-3-662-45523-4_27.

5. GAIGResearch (2019). *GAIGResearch/ECS7002P-MarioAI*. [online] GitHub. Available at: https://github.com/gaigresearch/ecs7002p-marioai#features [Accessed 1 Dec. 2019].

6. Green, M.C., Khalifa, A., Barros, G.A.B., Nealen, A. and Togelius, J. (2018). Generating levels that teach mechanics. *Proceedings of the 13th International Conference on the Foundations of Digital Games - FDG* '18. [online] Available at: https://arxiv.org/pdf/1807.06734.pdf [Accessed 28 Nov. 2019].

7. Karakovskiy, S. and Togelius, J. (2012). The Mario AI Benchmark and Competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), pp.55–67.

8. Lefebvre, S., and Neyret, F. (2003). Pattern based procedural textures. In Proceedings of the 2003 symposium on Interactive 3D graphics, 203–212. Available at: https://hal.inria.fr/inria-00537476/file/pattern_based_procedural_textures.pdf [Accessed 9 Dec. 2019].

9. Levina, E., and Bickel, P. J. 2006. *Texture synthesis and nonparametric resampling of random fields*. Ann. Statist. 34 (2006), no. 4, 1751--1773. Available at: https://projecteuclid.org/download/pdfview_1/euclid.aos/1162567632 [Accessed 7 Dec. 2019].

10. Smith, Gillian, and Jim Whitehead.2010. "*Analyzing the Expressive Rangeofa Level Generator*." In Proceedings of the FDG workshop on Procedural Content Generation in Games.

11. Smith, G., Whitehead, J., Mateas, M., Treanor, M., March, J. and Cha, M. (2011). Launchpad: A Rhythm-Based Level Generator for 2-D Platformers. IEEE Transactions on Computational Intelligence and AI in Games, [online] 3(1), pp.1–16. Available at: https://users.soe.ucsc.edu/~ejw/papers/launchpad-smith-tciaig-2011.pdf [Accessed 26 Nov. 2019].

12. Smith, G.; Treanor, M.; Whitehead, J.; and Mateas, M. (2009). Rhythm-based level generation for 2D platformers. In Proceedings of the 4th International Conference on Foundations of Digital Games, 175–182. Available at: http://delivery.acm.org/10.1145/1540000/1536548/p175-smith.pdf?ip=161.23.49.65&id=1536548&acc=ACTIVE%20SERVICE&key=BF07A2EE685417C5%2E53A024A4F41048F4%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1576275860_cc4cd73178d227235f73022d734c2767 [Accessed 8 Dec.2019].

13. Snodgrass, S. and Ontañón, S. (2014). *A Hierarchical Approach to Generating Maps Using Markov Chains*. [online] Available at: https://pdfs.semanticscholar.org/4ebf/85f3f8d1cf31988a30aaf13c28f0011b24fa.pdf [Accessed 4 Dec. 2019].

14. Snodgrass, S., & Ontañón, S. (2015). *A Hierarchical MdMC Approach to 2D Video Game Map Generation*. Available at: https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/viewFile/11518/11380 [Accessed 7 Dec. 2019].

15. Summerville, Adam; Mateas, Michael. "*Super Mario as a String: Platformer Level Generation Via LSTMs*." *ArXiv*, Mar. 2016, p. arXiv:1603.00930, adsabs.harvard.edu/abs/2016arXiv160300930S.

16. Super Mario Bros. (2019). *Super Mario Bros.* [online] Available at: https://supermariobros.io/ [Accessed 23 Nov. 2019].

17. Tietojenkäsittely, T. (2016). *Basics of Platform Games*. [online] Available at: https://www.theseus.fi/bitstream/handle/10024/119612/Thesis%20-%20Toni%20Minkkinen.pdf?sequence=1 [Accessed 7 Dec. 2019].

18. Togelius, J., Kastbjerg, E., Schedl, D. and Yannakakis, G. (2011). *What is Procedural Content Generation? Mario on the borderline*. [online] Available at: https://course.ccs.neu.edu/cs5150f13/readings/togelius_what.pdf [Accessed 1 Dec. 2019].

19. Togelius, J., Yannakakis, G., Stanley, K. and Browne, C. (2011). Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Transactions on Computational Intelligence and AI in Games, [online] 3(3), pp.172-186. Available at: https://ieeexplore.ieee.org/document/5756645 [Accessed 9 Dec. 2019].