A Constructive-Based Level Generator Improved by Stochastic Hill Climber in Mario AI

Chunghao, Ku | Jingye Shang | Yinghao Qin

Queen Mary University of London, UK

ec19485@qmul.ac.uk | ec19336@qmul.ac.uk | ml181215@qmul.ac.uk

Abstract. In this paper, we implemented a level generator in the Mario AI framework and applied two evolution methods to measure its performance. In short, we first set up a constructive generator with Cellular Automata Algorithm, which is based on the Notch generator provided by the framework. Next, we improved the generator using an evolutionary search-based method. We also compared our generator with the Notch generator by using two evolution approaches. Our results show that our generator performs better than the Notch generator on the basis of difficulty and diversity.

1 Introduction

Procedural content generation involves creating methods for generating game content, where its main goal is to automate the game generating process with as little human involvement as possible, but also able to create convincing content [4] .While game content concerns things such as quests, storylines, items, visuals and audio, the most commonly generated types of content are levels [4]. Due to the fundamental differences between every game's rules, mechanics and goals, different PCG methods were developed in order to satisfy the requirements and constraints posed by these fundamental differences**.** Therefore, the question then becomes, what PCG methods should be chosen to use? What metrics can be used to evaluate the fitness of the method as well as the playability of the game? Besides research in PCG, the need for developing metrics to evaluate "good" levels is also important in understanding the differences between different PCG methods through quantitative and qualitative measurements. The purpose of this paper is to focus on proposing one PCG method and two evaluation methods to test and evaluate the quality of levels generated by the newly created generator in *Mario AI.* The choice of game is due to the fact, that for recent papers that have provided benchmarks for research in PCG, are largely associated with the game *Mario AI.* This paper presents a solution for the PCG method: a cellular automata-based generator optimized by stochastic hill climber algorithm. The reason for this approach is that through prior observation of the previously generated levels by the Notch generator, the lack of difficulty and diversity were found. These aspects of the levels cause the gameplay to lack some degree of challenge and fun. Therefore,

the intended purpose of the solution proposed in this paper was improve those two aspects of the game, while ensuring the playability of the level.

2 Literature Review

2.1 Rule-Based Generator

Different procedural content generation techniques have been widely developed to adapt to different types of game. These techniques can be categorized into main groups, namely the constructive-based algorithms and search-based algorithms. The constructive-based algorithms focus on abiding by rules and constraints to assemble different components of the level based on the basic elements defined by the game model. One of the most common examples is the **notch** generator, which has been regarded as the most basic rule-based generator in the research field. A variation of this generator is the **p-notch** generator, which parameterizes the components used for the level. The level generator writes levels from left to right and assign probabilities to the components and add them to the level according to those probabilities [4]. More details regarding notch and its variation can be found in Steve Dahlskog's *"Patterns and Procedural Content Generation in Digital Games."* [3]

2.2 Search-Based Generator

Search-based algorithms [5] are techniques that seek to find the best possible solution throughout a search space. The most popular search algorithm for the research of PCG is the evolutionary algorithm. There have been numerous researches on combining different machine learning methods to train scene representations and evolve solutions of the best level. An example can be taken from Vanessa Volz's *"Evolving Mario Levels in the Latent Space of Deep Convolutional Generative Adversarial Network*" [6]. The paper presents a solution of using CMA-ES to evolve an ideal solution and ~~puts it in~~ the Generative Adversarial Network. Another example of Search-based implementation can be found in Ahmed Khalifa's "*Intentional Computational Level Design,"* [1] and the methods used are Feasible-infeasible two population (FI2POP) and constrained map-elites. Both algorithms are variations of evolutionary search algorithms and seek to explore different behavior of solutions while optimizing their performances.

2.3 Different Evaluation Metrics

Evaluation metrics are needed not only for the analysis of the performance of the PCG algorithms, but also for the differentiation between levels created by different generators. In Britton Horn's *"A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework," two* pattern-based metrics were used to assess the level, namely player experience and level composition [4]. In the paper, player's

experience is characterized by the leniency of the level. The leniency metric is an attempt to capture how difficult a level is for a player. Leniency was calculated by finding all points in the level where an action by the player is needed (e.g. the edge of a platform or the end of a string of blocks), and then determining how lenient that particular challenge would be to the player. The two variables used to define leniency are jump lengths and position of enemies, and the weights assigned to these parameters are based on the how difficult the challenge is for the player (e.g. death possibility). Linearity is a metric used to measure how linear the map is from left to right. The linearity metric is crucial in determining whether a level has variation of heights, which adds more degrees of freedom to the game. In Horn's paper [4], linearity is measured by the R2 goodness of fit, where the endpoints of each platform throughout the level is taken.

3 Techniques Implementation

A constructive-based level generator was first created by improving the notch generator. This was done by implementing the cellular automata algorithm. Next, a search-based evolutionary algorithm was created by defining the search space of the parameters, where the parameters considered were the ones defined in both the cellular automata and notch generator. The implementation of evolutionary algorithm uses the stochastic hill climber method. By considering a range of different possible values the parameters can take in the search space, the values evolve, and a solution was obtained after several iteration of the algorithms. In a nutshell, by applying stochastic-evolutionary algorithm to the parametrized cellular automata notch generator, the best generator can be found, and it is characterized and defined by the values the parameters take. By experimenting this improved level generator with two evaluation metrics, the experimental process follows by providing the end results, player performance and level quality of the level presented. The following presents a detail implementation of the techniques used to construct the generator.

3.1 Constructive Based: Cellular Automata Algorithm

The constructive-based method used is a Cellular Automata algorithm. It is an algorithm designed by Conway for his creation of *Conway's Game of Life*. For the implementation of this algorithm in *Mario AI* framework, it seeks to decide the appearance of a certain tile type by considering the states of its neighbors. The conditions are as follows: if a cell is 1, it becomes 0 if it has less than or equal to one neighbor that is 1. If a cell is 0 and if it has three neighbors that are 1s, the cell becomes 1. The algorithm can be understood by the visualization of the diagram provided in Figure 1 and Figure 2. This then sets the rules for determining what components to appear in a particular position of the level.

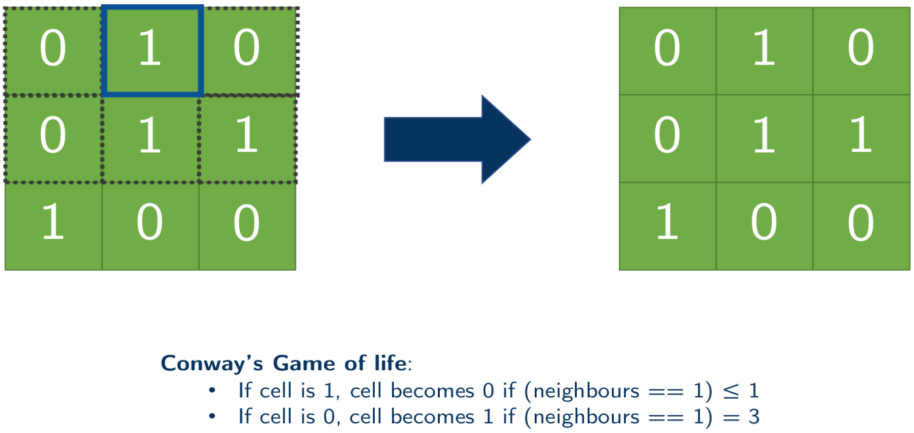

Figure 1 [7]: The algorithm iterates through the space and checks condition. For different number of iterations of the algorithm, the outcome of the values in the cells may be entirely different, but the property of this algorithm ensures a homogenous space, which means that regardless of how many numbers of iterations of the algorithm, a homogenous space remains homogenous.

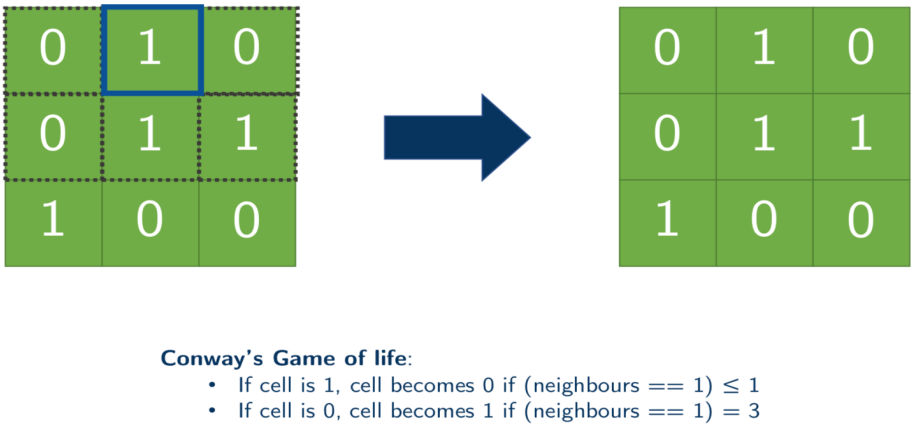

Figure 2 [7]: When one cell is done, it points to the next cell and the process repeats. The algorithm ends when the last cell of the window is reached.

The algorithm implemented is built upon an existing notch generator, using Conway's Game of Life to modify levels. The conditions are the same except the return of cell = 1 takes the parameter ='EMPTY' and cell = 0 takes the parameter "COIN." The number of times cellular automata is performed depends on the number of iterations assigned to it. Other modifications after applying cellular automata includes setting a standing point for Mario, a staircase-like step to Babel and some random tiles in the middle top part of the level. Finally, it also checks whether some parts of the level are empty or not and then add random tiles accordingly. This is because after applying

cellular automata for several iterations, some parts of the level appear to be empty. By adding these components to the level, it seeks to increase the playability of the level by adding a sense of uncertainty. The last part of the level, where a stair is generated before the endpoint, is to allow the player to experience a sense of achievement before arriving the end of the level. Note that the parameterized cellular automata define the parameters "type", "difficulty," "floor," "ceiling," and number of iterations. These parameters are then later used for the evolvement of their parameter search space.

3.2 Evolutionary Search-Based Method

The search-based method used throughout this project is random mutation hill climber. Random mutation hill climber is an evolutionary search algorithm that seeks to evolve the search space of level parameters in order to obtain a better representation of levels. The search space is defined by the possible configuration or placement of certain objects, with each individual representing a tile type. One characteristic of stochastic hill climber is that when it initializes the population of *N* individuals, it randomly chooses the values of genes of each individual. After initialisation, a fitness is used to evaluate the individual (sequence, array) and assigns a score to the individual. The individual with the highest value of fitness score is chosen. Note that the sequence is represented as the level in this context and the chosen individual is the current level. The fitness function used is based on the "leniency" evaluation metrics Mutation is then performed on the current level, which randomly chooses a value in the sequence to "mutate." This indicates that the parameter has a probability to take one of the values in the range search space. Current level then becomes candidate level after mutation. Finally, this candidate level is then compared against current level by their fitness scores (fitness calculation is applied to both levels). The process of mutation and comparison is repeated by a predefined number of iterations.

3.3 A Combination of the Two Methods

The improved version of cellular automata level generator is the final stage of the technique implementation for the level generator in this paper. The search-based evolutionary algorithm was used on the constructive-based method to search for the best level generator. The parameters initially defined for the constructive-based method were used on the search-based method to find the best combination of parameters within the defined parameter search range. This was done by parameterizing the notch generator. In order to parameterize the notch generator, a (1) parameter search space was first created, and an array was assigned to as many parameters defined as there are in the notch generator. These parameters are the odds for which a certain object may appear in the level, and they are assigned their relative indices. Next, (2) some values were added to the parameter search space (the array of each parameter). Finally, the stochastic hill climber algorithm was applied to evolve the values of these parameters in the search space. By specifying a given number of iterations, the evolution of parameter values and comparison of the scores repeat themselves. When the solu-

tions converge, the level generator accepts the evolved parameter values and generate levels based on those values.

3.4 Evaluation Metrics

The two metrics used to evaluate levels are linearity and leniency. The idea of the evaluation technique from Britton Horn's "A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework" is used as a reference to the evaluation technique implemented throughout this project. The techniques used are rather the simplified versions of those proposed in Horn's paper. A feature analysis [2] of *Mario AI* level model was done to extract relevant information regarding the leniency (difficulty) and linearity of the level. These features were then used as some criteria to assess the playability of the game. A fitness function was constructed for each metrics to calculate the total score of a level, which represents the level's playability. The average score was found by conducting an experiment, which involved generating several levels using both the constructive-based and search-based technique and then calculating their relative scores. The criteria formed of the level, including the relative weights assigned to each feature, is subjective, yet informative since it is able to provide a basis for comparisons between different PCG methods.

3.4.1 Leniency

In the context of the *Mario AI* model, the features that concern the difficulty of a level was analysed to be of the following: gaps, edges and enemies to available space ratio. What these three features have in common are that they all require some type of actions by using the game mechanics. When all three features are considered, the leniency scales up with respect to the score from the fitness as a function of these features. The score is then used to determine the difficulty of the level. Gaps and edges can be found by iterating through the string map of the level and then capturing the patterns. The enemy to available space ratio calculates the ratio between distance or space Mario is able to move within while being at the enemy moving range. This feature was found to be an important feature to be considered in determining the difficulty with prior experiment of the game.

3.4.2 Linearity

Linearity is another concept introduced to express the level's height variations. This metrics is important as it tells whether a level is flat for the most part or contains fluctuations caused by height differences. The method suggested was to calculate the sum of all the height differences from the mean height. This was done by calculating the individual heights, obtaining the mean heights and then calculating the difference between mean and each individual height of the platform and sum them. With this method, the score of the function is able to tell how spread out the platform or objects are from the mean height. If the score is too low, the map is more linear, and this may playthrough of the game to be monotonous. If the score is high, the variation of heights is high and thus adds more possibilities of different mechanics in the playthrough, which makes the game more fun and challenging.

4 Experimental Study

The experiments carried out in this paper involves assessing the (1) end results of the newly created generator, an agent's (2) player performance as a result of playing the level and using the evaluation metrics to (3) evaluate level quality. The end results, which are the actual levels generated, were compared against the criteria based on the characteristics of PCG methods. The difference in the player performance between the original level and the newly created level uses several statistics for comparison. This includes win rate, percentage completion, coins, total kills and jumps. The level quality was assessed by applying the evaluation metrics, with one metric evaluating the leniency and the other evaluating the linearity of the level.

The experiment was conducted to gather data of the player performance by using an A* agent to play a level 5 times for 20 different random levels. The scores and other data of the agent's performance were then recorded. This was done for both the new generator and the old generator (Notch) in order to compare the agent's performances from two different level generators. The leniency and linearity scores were also recorded separately in order for the level quality evaluation.

5 Discussion

5.1 End Results

The end results of the new generator can be described by the figures below. The criteria for the end results involve higher difficulty and higher distribution of level elements. In Figure 3, the level generated with respect to leniency evaluation, shows that the level difficulty has scaled up with regards to the features defined. The distribution of elements is high, which can be observed from the distribution of question blocks and coins. To conclude, the end results correspond to the criteria formed by using the leniency score evaluation method.

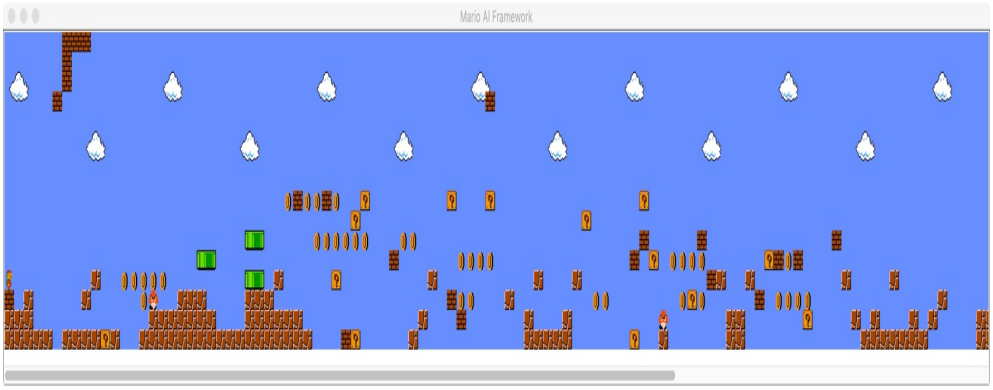

Figure 3: Level generated with respect to fitness evaluation 1 (leniency score)

In Figure 3, it shows a level generated by the new generator with respect to the second evaluation, which is the linearity score. It can be observed that the tiles are placed high up at the top of the level. It was concluded that it would not make sense to use the linearity score for the new generator since it did not correspond to the criteria for the level. As can be seen from the figure, the level was not improved according to the criteria. It was realised that the linearity evaluation method can only be used for evaluation of level quality. **It cannot be used as a fitness function for the evolutionary search algorithm**.

Figure 4: Level generated with respect to fitness evaluation 2 (linearity score)

5.2 Player's Performance

	Notch Generator	groupC Generator-Leniency Evaluation	groupC Generator-Linearity Evaluation
Win Rate	1.0	0.83	0.95
Percentage Completion	100.0	97.1560134887695	95.3061065673828
Coins	19.45	13.32	10.43
Remaining Time	13.85	12.62	12.83
Total Kills	0.86	1.3	1.2
Stomp Kills	0.86	0.6	0.85
Fall Kills	0.0	0.7	0.35
Jumps	8.65	17.31	15.46
Max X Jump	184.36357421875	183.05865234375	173.89833984375
Max Air Time	16.1	17.0	15.94
Question block bump	0.0	1.62	0.7

Table 1: Comparison of A agent's performance between Notch and the new generator*

In terms of winning rate, the winning rate of the new generator (groupC) is generally smaller, but the remaining time and total kill and jumps are higher, which shows that the new generator generates levels more difficult than Notch. In addition, there are certain statistics that the new generator has while Notch does not have, such as Fall kills and Question block bump. This again shows that new characteristics of the level generated using the new generator. However, for the two evaluation methods in our generator, it is clear that leniency evaluation has more impact on the generator than the linearity evaluation does. This shows that when the fitness function of leniency was used for the comparison of fitness scores between the candidate and current levels, the generation of level was affected, and the linearity should have only been used as an evaluation metric for the level quality, instead of using it as a fitness function in the evolutionary algorithm.

5.2 Level Quality

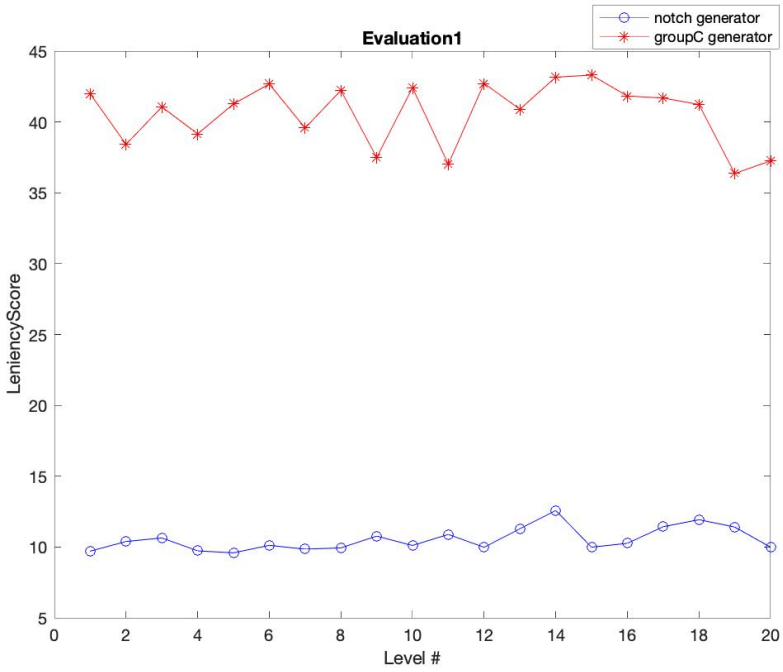

Figure 5: Plot of Leniency Score for Notch and New Generator

In Figure 5, it can be observed that Notch has a lower trend than that of the new generator, which shows that the overall difficulty of levels generated by the new generator is much higher. This also shows that the assumption made for what features to consider for the evaluation of level difficulty corresponds to the expected outcome,

which is that the level generated by the new generator should be more difficult. Another trend can be observed from the regularity of the curves for Notch and new generator. Notch has rather small fluctuations over Nth levels, whereas new generator has relatively larger fluctuations. Even though there is no relation between the corresponding points (Nth level) between the two generators, it can be concluded that the Notch generator generates levels of similar patterns, whereas the new generator generates a larger variety of levels. These two aspects of the game correspond to the initial goal of the design of algorithm, which is the increase of level difficulty.

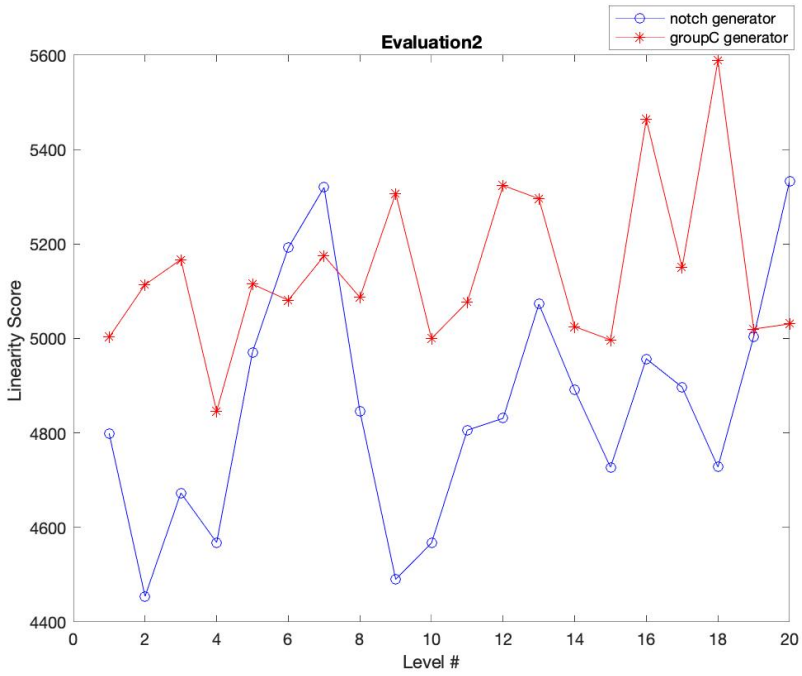

Figure 6: Plot of Linearity Score for Notch and New Generator

Figure 6 shows the levels linearity for both Notch and the new generator. It can be observed that linearity scores are generally higher for the new generator, which is an expected outcome as a result of higher distribution of elements in the level. Both generators show high fluctuations from level to level, showing that the distribution of heights vary hugely from level to level. However, there are certain levels where the linearity score for Notch is higher than that of the new generator. These are the unexpected results since levels generated by the new generator was assumed to have larger linearity scores than those generated by Notch. This can be explained by two possible reasons: first, for rule-based generators, levels are generated randomly according the probabilities assigned to each parameter, and this explains the random fluctuations

from level to level. Second, the heights sampled from the level which are used to calculate the mean may contain noise. This may lead to higher linearity scores irrespective of the degree to which elements are distributed. Therefore, although linearity score may be a good metric for specifying the degree of element distribution of a level, it may not be the best metric for random generators.

6 Conclusions and Future Work

This paper provides a way to implement a level generator in the *Mario AI* framework by focusing on improving the notch generator to increase a level's difficulty and diversity. Results have shown that difficulty and entertainment value can be measured by using the leniency and linearity evaluation methods. Through these evaluation methods, a level generator can be improved, thereby enhancing the desired features of the game. The constructive method provides as a basis for the evolutionary algorithm to optimize the features promoted according to the fitness functions. Further work may be to experiment using different evaluation metrics in the evolutionary search algorithm and compare the generated levels in terms of their quality. Work based on this premise is to assess game's playability from different perspectives as a result of different gameplay experience.

References

[1] Khalifa, Ahmed & Green, Michael & Barros, Gabriella & Togelius, Julian. (2019). *Intentional computational level design*. 796-803. 10.1145/3321707.3321849.

[2] Shaker, N., Yannakakis, G. N., & Togelius, J. (2011). Feature analysis for modelling game content quality. In *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011* (pp. 126-133). [6031998] https://doi.org/10.1109/CIG.2011.6031998

[3] Dahlskog, S. (2019). Patterns and procedural content generation in digital games: automatic level generation for digital games using game design patterns [pg. 1-5]. Retrieved 13 December 2019, from http://muep.mau.se/handle/2043/20371.

[4] Dahlskog, Steve & Horn, Britton & Shaker, Noor & Smith, Gillian & Togelius, Julian. (2014). A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework.

[5] M. Kerssemakers, J. Tuxen, J. Togelius and G. N. Yannakakis, "A procedural procedural level generator generator," *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Granada, 2012, pp. 335-341.doi: 10.1109/CIG.2012.6374174.

[6] Volz, Vanessa & Schrum, Jacob & Liu, Jialin & Lucas, Simon & Smith, Adam & Risi, Sebastian. (2018). Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network.

[7] Raluca D. Gaina (2019). Lecture 7: Procedural Content Generation [pg.36-37]. Retrieved from https://qmplus.qmul.ac.uk/