

# Procedural Map Generation for 'Splatted': Enhancing Player Experience through Genetic Algorithms and AI Finite State Machines in a Snowball Throwing Game

Lukky Hariyanto<sup>1</sup> and Hendrawan Armanto<sup>1</sup>

<sup>1</sup>Informatics Department, Faculty of Science and Technology, Institut Sains dan Teknologi Terpadu Surabaya, Surabaya, Indonesia

**Corresponding author:** Lukky Hariyanto (e-mail: lukky.h20@mhs.istts.ac.id).

**ABSTRACT** Games, a now extremely prevalent form of global entertainment, have emerged as a leading industry in the entertainment media, surpassing other entertainment media such as books, films, and music. However, game development is a complex endeavor, requiring a diverse set of talents to create a decent game for people to enjoy. Some of the talents needed to create a good game is a game designer, which dictates how a player can interact with the world, a writer, which pours a meaningful story inside said world, and a composer, which uses music to elevate the emotions evoked by the game and its events. With that being said, this research aims to streamline the creation process of the game designers, specifically the level designers by focusing on procedural map generation and artificial intelligence to create a map that is in a playable state for the players to play in. Procedural map generation, facilitated by a genetic algorithm inspired by Darwin's evolutionary theory, expedites the level design process. The research explores two types of map generation—tile-based and template-based, each with distinct advantages and disadvantages. Through user acceptance tests and expert-level analysis, it is evident that the genetic algorithm performs effectively, achieving a noteworthy level of player satisfaction.

**KEYWORDS** Game, Genetic Algorithm, Map, Procedural Map Generation.

## I. INTRODUCTION

Every human being needs entertainment in their life. Forms of entertainment may vary, one of which are games. No less than other types of entertainment such as films or books, games are a type of entertainment that requires a high level of technical complexity in its creation. Various components such as story, gameplay, system balancing, and marketing are needed in designing a game [1]–[3]. The more complex a game, the more complex its constituent components will be. Some of the important components in making games is level design and artificial intelligence for NPCs. Even though these two components are not the top priority in making a game, without a good level of design and believable AI from the NPCs, the game will feel bland.

In making a level, level design requires a lot of effort in terms of time, assets, and the endurance of the designer who designs it. Level creators must consider various things starting from the player's position, enemy position, item position, good path arrangements so that the game is interesting and balanced, and a few other considerations. For

example, Live Service games such as Valorant, Apex Legends, Fortnite, or Overwatch 2 require continuous map updates so that players don't get bored of playing and switch to another game. Because of this complexity, various research was carried out to make it easier for designers to carry out level design which ultimately give birth to Procedural Map Generation [4]–[6]. Some examples of algorithms in procedural map generation are Perlin Noise [7] which is used in the game Minecraft, Fractal Terrain Generation [8] in the game Terraria, or Genetic algorithms [9]–[15] in various existing studies.

This research focuses on creating a snowball throwing game called Splatted. The creation of the level design for this game will not be done manually but automatically using a genetic algorithm. Apart from that, in this game artificial intelligence will also be developed which can influence the behavior of Non Player Characters (NPC) so that the game can be more interesting. It is expected that through this research, similar games can apply the methodology so that the games developed can become more interesting.

## II. "SPLATTED" GAME

This game is developed for the purpose of research and as a case study in testing. This sub-chapter will discuss details related to gameplay and the artificial intelligence being developed.

### A. GAMEPLAY

Splatted is a game in which two teams, each consisting of five players, engage in a snowball war. Each successful hit on an opposing player scores points, with the primary objective being to accumulate the highest points and emerge as the winner of the snowball war. The winning team is either the one that meets the required point target or has more points than the opposing team when the game time ends. Players can perform several actions in this game, including picking up the ball from the ground, throwing the ball with the goal of hitting an opposing player, catching the ball thrown by an opposing player, and executing a "Fakeout" to deceive an opponent attempting to catch the thrown ball. Figure 1 provides an example of Splatted game footage where the white area represents the game area (snow), and the dark gray colors represent rocks (obstacles). Players and snowball throwers are only allowed to move within the snow areas.

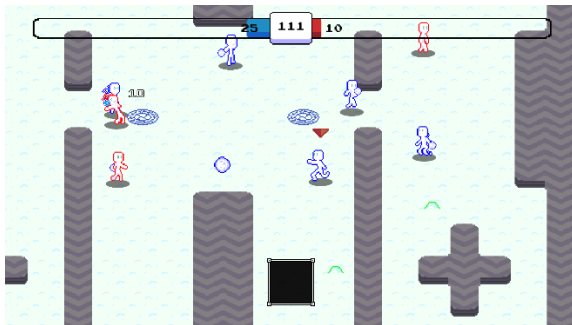







Figure 1. Screenshot of the Splatted game

### B. SPECIAL BALL

Apart from the general snowballs, to add some spice into the game, several special spawners are provided in the levels. Each spawner has a special ball that can be picked up and used to attack opponents. Table 1 is a table of the special balls available and the function of these balls.

TABLE I  
SPECIAL BALLS IN SPLATTED

Name	Display	Function
Ice Piercer		Pierces past opposing players and teammates
Snow-A-Rang		The ball returns to the thrower after hitting someone

Explod-o-Ball		The ball explodes after a set duration and hits all the players in the explosion area
Freezing Winter		The player hit by the thrown ball is slowed down for several seconds
Stone Auger		Penetrates players and walls, then breaks into 3 normal smaller snowballs

### C. ARTIFICIAL INTELLIGENCE (NPC)

In Splatted games, the behavior of Non-Player Characters (NPCs) is governed by a Finite State Machine (FSM). Utilizing an FSM allows NPCs to have several states, each providing different behaviors. Transitions between states in the FSM are influenced by real-time conditions of the NPCs, which could be affected by other NPCs, players, or the surrounding environment. The following section will describe the various states that an NPC can possess:

#### 1) RANDOM WALKING

If the NPC does not have a snowball, it will walk to a randomly selected location, attempting to find a snowball on the ground along the way. However, if the NPC already possesses a snowball, it will start searching for opponents. If the NPC reaches the target location without finding a snowball or an opponent, a new location will be randomly selected, and the search will resume.

#### 2) TAKE THE BALL

If the NPC in "Random Walking" state spots a snowball or a special ball, it will transition to the "Take the Ball" state. In this state, the NPC will walk towards the identified ball to pick it up. After securing the ball, or if the ball is taken by someone else, the NPC will revert to the "Random Walking" state.

#### 3) AIM & THROW

If an NPC in the 'Random Walking' state has a ball in hand and spots a member of the opposing team, the NPC will transition to the 'Aim & Throw' state. In this state, the NPC will cease movement and aim at the opponent. After confirming the aim is accurate, the NPC will throw the ball towards the predicted future position of the opponent. Following the throw, the NPC will revert to the 'Random Walking' state.

#### 4) FOLLOW TARGET

If the target being aimed at in the "Aim & Throw" state disappears from the NPC's view, the NPC will transition to the "Follow Target" state. In this state, the NPC will pursue the opponent with the goal of regaining vision of the target. During this pursuit, if the NPC fails to locate the opponent within a certain timeframe, the NPC will abandon the chase and revert to the "Random Walking" state.

5) CATCH BALL

Specifically, the ‘Catch Ball’ state can be entered from any other state when the NPC spots a ball being thrown in its direction. The purpose of this state is to enable the NPC to attempt to catch balls that are being thrown at it.

**III. IMPLEMENTATION OF GENETIC ALGORITHM INTO SPLATTED GAMES**

In this sub-chapter, we will delve into the details of using genetic algorithms in Splatted games. This includes everything from the representation and fitness values, to the methods employed in each operation of the genetic algorithm.

**A. REPRESENTASION**

In this research, two level generation models are utilized: tile-based generation and template-based generation. Each model has its own unique process and method of representation.

**Tile-Based Generation**

In this model, the individual representation is a 1-dimensional array with a length equal to the size of the level to be created (for instance, for a 10x10 level, the individual length would be 100). Each gene in this individual will hold a value ranging from 0 to 3, where the number 0 represents an empty area, 1 represents a rock/obstacle, 2 represents the position of a special ball spawner, and 3 represents the player’s position. Figure 2 illustrates an example of a tile-based representation converted into a player-understandable level, with the level size being 3x3.”

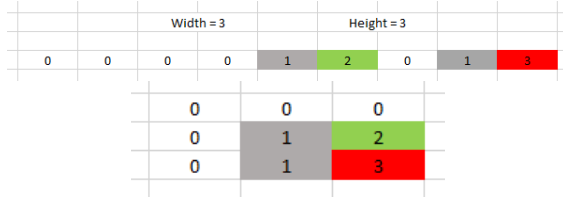


Figure 2. Example of Tile Based Representation into a map

**Template-Based Generation**

Similar to tile-based generation, template-based generation is also represented as a 1-dimensional array. However, unlike tile-based generation, the gene value in this model does not contain the numbers 0-3, which represent objects at the level. Instead, it has values ranging from -n to n, where n is the number of prepared templates. A negative value indicates that there will be one special ball in the middle of the level in the template with ID number x. Apart from the gene value, another difference from the tile-based representation is the length of the individual. Template-based representations have shorter individual lengths because one template consists of 5x5 tiles. For instance, if the level size is 10x10, the individual length used is 100/(5x5), or 4.

This research incorporates three types of templates, each of which has several variations. The three types of templates are as follows:

- Oneway Template**  
A Oneway template refers to a variation of the template that, when rotated by 90°, 180°, or 270°, still produces the same level. For instance, in Figure 3, the level is represented by code 12. This type of template is available in only three variations.
- Two-way Template**  
A Two-way template refers to a variation of the template that, when rotated by 90° or 270°, yields different level results. However, when rotated by 180°, it produces the same level. For instance, in Figure 3, the level is represented by code 5. This type of template is available in eight variations.
- Four-way Template**  
A ‘Fourway’ template refers to a variation of the template that, when rotated by 90°, 180°, or 270°, yields different level results. For instance, in Figure 3, the level is represented by either code 8 or 24. This type of template is available in five variations

In this research, a total of 39 templates were provided. These were derived from 3 oneway templates, 16 twoway templates (8 variations x 2), and 20 fourway templates (5 variations x 4). For twoway and fourway templates, a single variation will yield 2 and 4 different levels respectively when implemented. Figure 3 illustrates an example of a template-based representation converted into a player-understandable level, with the level size being 10x10.

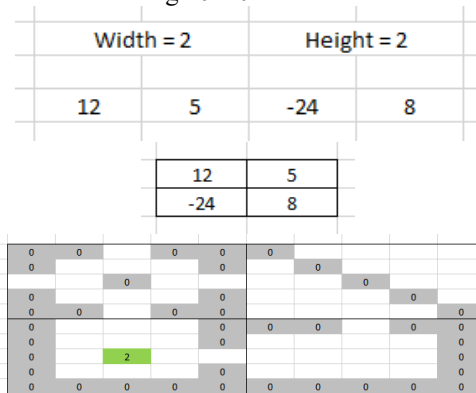


Figure 3. Example of Template-Based Representation Converted into a Map

**B. GENETIC ALGORITHM OPERATOR**

Genetic algorithms encompass several operators, each with a variety of algorithmic choices. This research has explored these algorithms to identify the most suitable and appropriate one for accomplishing level creation, which is the primary focus of this research. The operators and their respective choices are as follows:

1. Parent Selection

The selection method [16][17] used by this research is the Roulette Wheel [18]. This method is suitable for use in this research considering that the better the fitness score, the greater the chance of an individual or level candidate being selected as a parent. Where levels that are not good are generally less playable so that if they are selected it will cause the next generation to also be less playable.

2. Crossover

This research uses the uniform crossover algorithm [19]-[22] as the operator. Where this algorithm will provide a 50% chance for each gene to be exchanged between the 2 selected parents. Figure 4 is an example of uniform crossover visualization. This algorithm is suitable to be applied because in this study 1 gene represents 1 tile/template. So that the levels produced in the next generation will have good variations.

Chromosome 1	10	7	3	2	10	5	2	8	10	8
Chromosome 2	7	4	8	7	4	3	5	8	8	0
Swap?	0	1	0	1	1	1	1	1	0	0
					V					
Chromosome 1	10	4	3	7	4	3	5	8	10	8
Chromosome 2	7	7	8	2	10	5	2	8	8	0

Figure 4. Visualization of Uniform Crossover

3. Mutation

For the same reasons as selecting the crossover algorithm, the mutation algorithm [23][24] partial shuffle mutation or scramble mutation was chosen and used in this study. This algorithm has more opportunities for gene changes in the next generation considering that all genes between the two barriers will be randomized in order and place. Figure 5 is an example of scramble mutation visualization.

			Shuffle						
1	5	1	3	3	2	5	4	1	
				V					
1	5	1	5	3	2	3	4	1	

Figure 5. Visualization of Scramble Mutation

4. Elitism

The elitism method [25][26] used in this research is to combine all offspring (children) and some parents who have the best fitness scores in the previous generation. For example, the minimum population that must be provided is 100 while the previous generation gave birth to 80 units, the remaining 20 are taken from parents who have the best fitness scores.

C. STOP CONDITION

The stopping condition for the genetic algorithm in this research is convergence. If all individuals across the last 100 generations have not shown significant development or have converged, then the iteration of the genetic algorithm is halted. This approach is adopted considering that level formation occurs during the initial game loading and needs to be efficient to prevent long waiting times for players. Although it's undeniable that level formation can sometimes take a considerable amount of time (around 20 seconds), this duration is still within the player's tolerance for waiting time.

IV. FITNESS FUNCTION

The fitness function plays a crucial role in determining the quality of an individual in the genetic algorithm. In this research, five fitness functions will be utilized for tile-based generation, and four fitness functions will be employed for template-based generation. Notably, three out of the four fitness functions for template-based generation are also used in tile-based generation.

A. FITNESS NUMBER OF STONES

This fitness function is utilized to regulate the distribution of stones within a level. The primary objective is to ensure that a level doesn't consist solely of stones, or conversely, lack of it entirely. Certain parameters are initially established, such as MinR (the minimum number of stones in a level) and MaxR (the maximum number of stones in a level).

$$m = \begin{cases} MinR - R, & \text{for } R < MinR \\ R - MaxR, & \text{for } R > MaxR \\ 0, & \text{for } MinR < R < MaxR \end{cases} \quad (1)$$

As can be seen in (1), the value of 'm' will be made negative when the number of stones is either too few (less than MinR) or too many (greater than MaxR). However, if it falls within the range, the value of 'm' will be set to 0, indicating that there are no constraints on the number of stones in that level.

$$M = \begin{cases} A - MaxR, & \text{for } MinR > A - MaxR \\ MinR, & \text{for } MinR < A - MaxR \end{cases} \quad (2)$$

To ensure that the fitness value does not become negative, normalization is performed on the value of 'm'. Equation (2) is used to find the divisor to ensure proper normalization. There are two methods to find the divisor for normalization: subtracting the area with the maximum number of stones, or directly taking the minimum number of stones when the result of subtracting the area and the maximum number of stones is less than the minimum number of stones

$$F = \left(1 - \frac{m}{M}\right)^2 \quad (3)$$

Equation (3) represents the normalization equation utilized in this fitness function. This function ensures that the  $F$  value will never be negative. It's important to note that this fitness function is exclusively used in tile-based generation. This is because, in template-based generation, the number of stones is determined by the composition of the used template. Therefore, applying this function in a template-based context would disrupt the variation occurrences in the existing templates.

### B. GROUP STONE SIZE FITNESS

The fitness function is utilized to calculate the size of stone groups present within a level. The primary objective of this fitness function is to ensure that no stone group is excessively large or too small. Similar to the previous fitness function, (1) and (2) remain employed in computing this fitness function. However, unlike the previous fitness function, which was calculated for one level, in this fitness function, both equations are computed for each stone group encountered.

$$F = \left(\frac{\sum_{i=1}^n \left(1 - \frac{m_i}{M}\right)}{n}\right)^2 \quad (4)$$

To calculate the fitness value of a stone group size, we need to first determine the values of  $m$  and  $M$  for all stone groups. Once we have these values, we will use (4) to calculate the fitness value. It's important to note that the stone group size fitness is only applied in tile-based generation, and not in template-based generation.

### C. ACCESSIBLE AREA FITNESS

This fitness function is employed to calculate the extent of the area accessible to the player. The more interconnected areas within the level, the better the level is considered. The primary objective of this function is to ensure that there are few inaccessible areas within the level, as the player character in the splatted game cannot pass through roofs or destroy obstacles.

$$F = \left(\frac{a_{\text{terbesar}}}{a_{\text{total}}}\right)^2 \quad (5)$$

Equation (5) represents the fitness function used to evaluate the quality of a level based on its area. Here, the largest value of  $a$  corresponds to the largest connected area, while the total value of  $a$  represents the total number of objects in the level that are not stones.

### D. SPECIAL BALL ACCESSIBILITY FITNESS

This fitness function aims to ensure that each special ball present in the level is reachable by a player. However, this accessibility is not for all players but only for the closest player to the ball. Thus, the appearance of special balls in the level ensures that they are reachable by at least the nearest player.

$$F = \left(\frac{P_{\text{akses}}}{P_{\text{total}}}\right)^2 \quad (6)$$

Equation (6) is the fitness value equation that ensures all special balls can be reached by the nearest player.  $P_{\text{total}}$  represents the total number of special balls, while  $P_{\text{akses}}$  is the number of special balls that can be reached without any obstacles by the nearest player. A special ball is deemed accessible if a path search using the A\* algorithm can return a path from the nearest player to the special ball without any obstacles.

### E. SPECIAL BALL RATIO FITNESS

The special ball ratio fitness function is used to ensure the presence of special balls in a level. Several predetermined parameters are defined beforehand, including  $\text{MinP}$  (the minimum number of special balls in a level) and  $\text{MaxP}$  (the maximum number of special balls in a level).

$$m = \begin{cases} \text{MinP} - P, & \text{for } P < \text{MinP} \\ P - \text{MaxP}, & \text{for } P > \text{MaxP} \\ 0, & \text{for } \text{MinP} < P < \text{MaxP} \end{cases} \quad (7)$$

As depicted in (7), it can be observed that the value of  $m$  will be made negative when the number of special balls is insufficient (less than  $\text{MinP}$ ) or excessive (greater than  $\text{MaxP}$ ). However, if the number of special balls falls within the specified range, the value of  $m$  will be set to 0, indicating no constraints on the number of special balls in the level.

$$M = \begin{cases} A - \text{MaxP}, & \text{for } \text{MinP} > A - \text{MaxP} \\ \text{MinP}, & \text{for } \text{MinP} < A - \text{MaxP} \end{cases} \quad (8)$$

To prevent fitness values from becoming negative, normalization is performed on the value of  $m$ . Equation (8) presents the equation to find the divisor for proper normalization. There are two approaches to finding the divisor for normalization: subtracting the area from the maximum number of special balls or directly taking the minimum number of special balls when the difference between the area and the maximum number of special balls is less than the minimum number of stones.

$$F = \left(1 - \frac{m}{M}\right)^2 \quad (9)$$

Equation (9) represents the normalization equation used in this fitness function. Through this equation, it is ensured that the fitness value  $F$  will never be negative.

**F. TEMPLATE VARIETY FITNESS**

The final fitness function utilized in this research is template variety. The objective of this fitness function is to avoid repetitive occurrences of the same template within a level. Consequently, the aim is to generate levels with a high template variety, ensuring that multiple templates are used rather than repeating one or two templates multiple times.

$$x = \begin{cases} 0 & \text{for } t_i - T_l > 0 \\ t_i - T_l & \text{for } t_i - T_l < 0 \end{cases} \quad (10)$$

As shown in (10), it can be observed that the value of  $x$  will be made negative when the number of occurrences of a template exceeds the given tolerance limit. However, if it is within the tolerance, the value of  $x$  will be set to 0. This calculation is performed for each template encountered.

$$F = \begin{cases} 0 & \text{for } \frac{\sum_{i=1}^t (1+x)}{t} < 0 \\ \left(\frac{\sum_{i=1}^t (1+x)}{t}\right)^2 & \text{for } \frac{\sum_{i=1}^t (1+x)}{t} > 0 \end{cases} \quad (11)$$

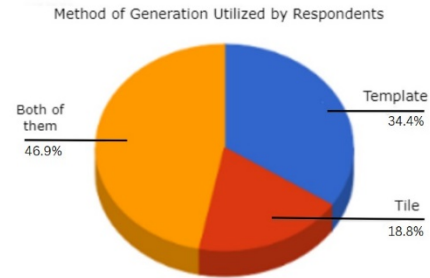
After calculating the occurrences of all templates on a level using (10), (11) is employed to compute the fitness value by calculating the squared average of the  $x$  values. If the resulting value is negative, the fitness value is set to 0 to avoid interference with the values of other fitness functions. This fitness function is specifically utilized for template-based generation.

**V. EXPERIMENTAL TESTING**

This research employs two testing techniques. The first one involves user acceptance testing or questionnaire-based

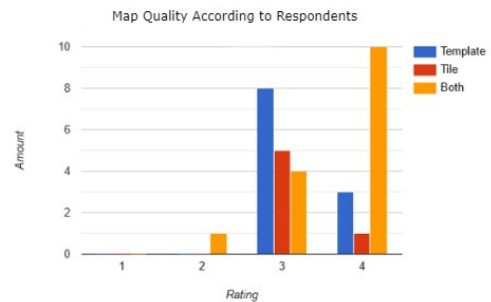
assessment. The second technique involves analyzing the levels generated by game experts.

**A. USER ACCEPTANCE**



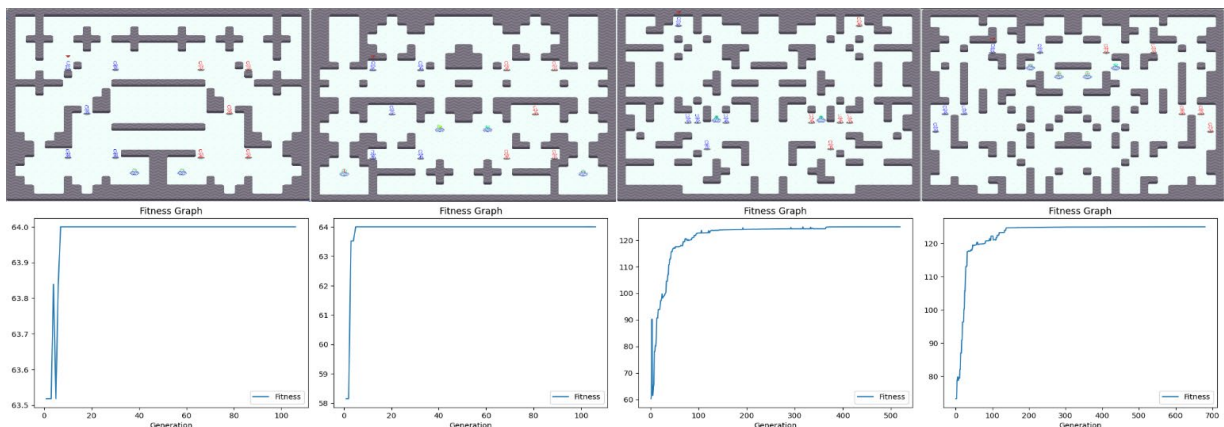
**Figure 6. Level Generation Selection by Respondents**

The questionnaire was completed by 32 individuals whose profile fits that of gamers aged 18–22 years, with a minimum of 2–3 hours of daily gaming, and who have previously played MOBA or 2D real-time strategy games such as Bomberman. Figure 6 depicts the percentage of respondents selecting different level generation modes. 46% opted to try both modes, while the remainder only tried one of the modes.



**Figure 7. Respondents' Given Scores**

Meanwhile, Figure 7 illustrates the ratings provided by respondents who have tried both modes as well as those who



**Figure 8. Visualization of Fitness Flow in Generating a Map**

have tried only one mode. The questionnaire results reveal that only one respondent found the generated levels to be unsatisfactory, while the remaining 31 individuals rated the levels with a minimum score of 3. Through Figure 7, it can be inferred that the generated levels meet players' expectations.

## B. ANALYSIS OF THE LEVELS

In the level analysis conducted by expert evaluation, several levels were generated using both tile-based generation and template-based generation methods. The expert, with a profile matching that of a high-rated MOBA player who spends 5-6 hours gaming daily, was asked to analyze several levels. Figure 8 (from left to right) showcases the best 20 x 30 tile-sized levels for two examples of template-based generation and two examples of tile-based generation. From the four exemplary levels, it is evident that all levels feature open areas conducive to player strategy, with no inaccessible or enclosed spaces, appropriately placed special items, and a visually appealing aesthetic suitable for gameplay.

Regarding the analysis of the levels based on their fitness values, both levels generated by template-based generation exhibit a high speed in achieving maximum fitness or generating good levels. Meanwhile, the two levels produced by tile-based generation, although capable of attaining high fitness values, require more time due to the gradual changes that occur per tile in tile-based generation.

## VI. CONCLUSION

In this study utilizing the splatted game, it can be concluded that genetic algorithms perform well in generating levels that are enjoyable, playable, and meet player expectations. However, the main drawback of level generation using genetic algorithms lies in the significant dependency on the fitness function employed. The quality of generated levels improves when the fitness function aligns accurately with the desired criteria. Achieving this alignment necessitates a considerable amount of time for experimenting with and refining various fitness function variants.

## ACKNOWLEDGEMENTS

We extend our sincere gratitude to the Surabaya Institute of Integrated Science and Technology for their support and facilitation of this research.

## AUTHOR CONTRIBUTIONS

**Lukky Hariyanto:** Application Development, Article Writing, Image and Data Provision.

**Hendrawan Armanto:** Article Writing, Article Copyediting, Finishing.

## COPYRIGHT



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

## REFERENCES

- [1] J. Schell, *The Art of Game Design: A Book of Lenses, Third Edition*. CRC Press, 2019.
- [2] E. Adams, *Fundamentals of Game Design*. Pearson Education, 2010.
- [3] M. Moore, *Basics of Game Design*. CRC Press, 2016.
- [4] N. A. Barriga, *A Short Introduction to Procedural Content Generation Algorithms for Videogames*. 2018.
- [5] V. Kraner, I. Fister jr, and L. Brezočnik, "Procedural Content Generation of Custom Tower Defense Game Using Genetic Algorithms," 2021, pp. 493–503.
- [6] S. Putra and W. Istiono, "Implementation Simple Additive Weighting in Procedural Content Generation Strategy Game," *vol.* 4, pp. 9–18, 2022.
- [7] E. Frank and N. Olsson, "Procedural city generation using Perlin noise." 2017.
- [8] N. Sainio, "TERRAIN GENERATION ALGORITHMS," 2023.
- [9] A. Lambora, K. Gupta, and K. Chopra, "Genetic Algorithm- A Literature Review," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 380–384. doi: 10.1109/COMITCon.2019.8862255.
- [10] L. Haldurai, T. Madhubala, and R. Rajalakshmi, "A study on genetic algorithm and its applications," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 10, pp. 139–143, 2016.
- [11] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimed. Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, 2021, doi: 10.1007/s11042-020-10139-6.
- [12] H. Armanto, H. A. Rosyid, M. Muladi, and G. Gunawan, "Evolutionary Algorithm in Game – A Systematic Review," *Kinet. Game Technol. Inf. Syst. Comput. Network. Comput. Electron. Control*, May 2023, doi: 10.22219/kinetik.v8i2.1714.
- [13] W. Alfonsus, A. Hendrawan, and T. J. Gunawan, "Focused Web Crawler Using Genetic Algorithms and Symbiotic Organism Search," *革新的コンピューティング・情報・制御に関する速報*, vol. 15, no. 12, p. 1345, 2021.
- [14] H. Armanto, K. Setiabudi, and C. Pickerling, "Komparasi Algoritma WOA, MFO dan Genetic pada Optimasi Evolutionary Neural Network dalam Menyelesaikan Permainan 2048," *J. Inov. Teknol. dan Edukasi Tek.*, vol. 1, no. 9, pp. 676–684, 2021.
- [15] H. Armanto, R. D. Putra, and C. Pickerling, "MVPA and GA Comparison for State Space Optimization at Classic Tetris Game Agent Problem," *Inf. J. Ilm. Bid. Teknol. Inf. dan Komun.*, vol. 7, no. 1, pp. 73–80, 2022.
- [16] S. Prayudani, A. Hizriadi, E. B. Nababan, and S. Suwilo, "Analysis effect of tournament selection on genetic algorithm performance in traveling salesman problem (TSP)," in *Journal of Physics: Conference Series*, 2020, vol. 1566, no. 1, p. 12131.
- [17] S. L. Yadav and A. Sohal, "Comparative study of different selection techniques in genetic algorithm," *Int. J. Eng. Sci. Math.*, vol. 6, no. 3, pp. 174–180, 2017.
- [18] J. Y. Setiawan, D. E. Herwindiati, and T. Sutrisno, "Algoritma Genetika Dengan Roulette Wheel Selection dan Arithmetic Crossover Untuk Pengelompokan," *J. Ilmu Komput. dan Sist. Inf.*, vol. 7, no. 1, pp. 58–64, 2019.
- [19] J. L. Pachau, A. Roy, and A. Kumar Saha, "An overview of crossover techniques in genetic algorithm," *Model. Simul. Optim. Proc. CoMSO 2020*, pp. 581–598, 2021.
- [20] P. Kora and P. Yadlapalli, "Crossover operators in genetic algorithms: A review," *Int. J. Comput. Appl.*, vol. 162, no. 10, 2017.
- [21] A. Malik, "A study of genetic algorithm and crossover techniques," *Int. J. Comput. Sci. Mob. Comput.*, vol. 8, no. 3, pp. 335–344, 2019.
- [22] L. Manzoni, L. Mariot, and E. Tuba, "Balanced crossover operators in genetic algorithms," *Swarm Evol. Comput.*, vol. 54, p. 100646, 2020.
- [23] B. H. Abed-alguni, "Island-based cuckoo search with highly disruptive polynomial mutation," *Int. J. Artif. Intell.*, vol. 17, no. 1, pp. 57–82, 2019.

- [24] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach," *Information*, vol. 10, no. 12, p. 390, 2019.
- [25] G. Guariso and M. Sangiorgio, "Improving the performance of multiobjective genetic algorithms: An elitism-based approach," *Information*, vol. 11, no. 12, p. 587, 2020.
- [26] H. Du, Z. Wang, W. E. I. Zhan, and J. Guo, "Elitism and distance strategy for selection of evolutionary algorithms," *IEEE Access*, vol. 6, pp. 44531–44541, 2018.