International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

http://dx.doi.org/10.12785/ijcds/XXXXXX

Pacing Control in Strategy Game Map Generation using Wave Function Collapse

Muhammad Alifian Aqshol[1], Ardiawan Bagus Harisa*[1], Pulung Nurtantianto Andono[1] and Wen-Kai Tai[2]

[1]*Faculty of Computer Science, Universitas Dian Nuswantoro, Semarang, Indonesia*
[2]*Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan*

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: In our effort to introduce programming foundations to secondary-level students in a more engaging way, we are developing a turn-based tactics game named MECH.AI. In addition of being a strategy game, MECH.AI incorporates elements of serious games, focusing on educating players in basic programming and artificial intelligence concepts. Players can control game objects not only through the graphical user interface but also via the command line, simulating programming practices. However, providing game maps and challenges is crucial and can significantly impact game pacing, thereby affecting the player experience. The creation of battle maps is essential to maintain diversity and excitement in the game, preventing it from becoming monotonous. Producing high-quality content in a short amount of time is challenging, and procedural content generation emerges as an effective solution to this problem. In this study, we aim to apply a system that generates game maps with pacing controllable by the designer using the Wave Function Collapse algorithm. We will examine a pacing aspect that influences player experience, specifically tempo. This system enables us to generate tactical game maps and manage game pacing. The results show that controlling game pacing while in pre-battle map generation is feasible and adjustable to match designer's preferences.

Keywords: Procedural Content Generation, Turn-based Tactics, Strategy Games, Wave Function Collapse, Game Design, Serious Games

1. INTRODUCTION

Besides providing entertainment, serious games serve various purposes, including educational aspects [1], [2] such as enhancing students' motivation [3], [4] and involvement [5] in basic programming fundamentals. Among the popular game sub-genres is turn-based tactics (TBT) game, a subset of strategy games [6]. Turn-based tactics is a sub-genre of strategy games and one of the most profitable genres in the game industry [7]. A turn-based tactics game implements turn-based mechanics and strategic elements. Typically, players control characters and resources to achieve victory by defeating enemies and exploring the environment. The main mechanics in turn-based tactics are not exclusive to this sub-genre, as these mechanics are also found in tactical role-playing games (TRPGs), also called simulation role-playing games (SRPGs) [8], [9], turn-based combat, and turn-based strategy games in general. Due to the presence of tactical aspects in TBT games, this genre is well-suited for teaching basic programming fundamentals.

As part of our progress to teach the basic programming fundamentals to secondary-level students, we created MECH.AI, a turn-based tactics game with a science fiction theme where players act as mecha robot pilots competing to mine precious gems as universal currency of the future. MECH.AI is developed by following the game development life cycle (GDLC), a software development methodology aimed at managing the development process, especially in video games [10], [11].

However, providing maps and challenges that are suitable for players is a crucial task, as it may affect the game pacing and, consequently, the overall player experience. Pacing, an essential aspect of game [12], refers to the level or flow of activity or actions in a video game, such as movement, attacks, or other elements that influence the player's experience. Pacing can be observed separately in threats, movement impetus, and tempo [13]. Understanding the pacing distribution through the game is important, because we can illustrate the level of difficulty and its distribution during gameplay [14]. Moreover, keeping the battle map interesting while also creating non-monotonous challenges is key to maintaining player excitement. Fortunately, procedural content generation (PCG) can be used as an effective solution to address this problem. PCG is a general term for systems that utilize specific design patterns to generate new assets based on those patterns [15].

Creating high-quality content in a short amount of time and automatically generating battle maps that align with each pacing determined by the game designer pose challenges. Manual creation of battle maps according to game pacing by developers can be time-consuming and require significant effort. Therefore, a technique is needed to automatically generate maps quickly and effectively. One procedural method that can address this problem is Wave Function Collapse (WFC). Wave Function Collapse is a search algorithm renowned for its ability to produce consistent output on a large scale while leveraging a small set of constraints [16].

With a greater variety of stage maps, players can employ different strategies at each game pacing and enjoy a more fulfilling gaming experience. The aim of this research is to apply and analyse the WFC technique in automatically generating turn-based tactics game maps according to the game pacing set by game designer. By generating map variants while allowing control over the resulting game pacing, the experience provided to players by the designer will effectively maintain player engagement in learning the basic programming fundamentals incorporated into gameplay. Furthermore, this technique can also be applied in the development of other sub-genres of strategy games, thereby enriching the variety of stage maps in those games as well. However, we do not discuss the impact of the proposed system on serious game purposes.

2. LITERATURE STUDY

In this study, we use the GDLC to develop our turn-based tactics game MECH.AI, and then apply WFC as a technique to procedurally generate battle maps. Additionally, we aim to observe and confirm that the concept of game pacing can be adjusted to control map generation in strategy games, as it has been applied in dungeon-crawler [13] and platformer games [17] before.

A. *Game Development Life Cycle (GDLC)*

GDLC summarizes the entire process of developing a game [10] and can be organized into four phases: analysis, production, testing, and release [18]. Analysis, the first step in GDLC, determines the game type and creates the main concept. This phase results in game concepts and descriptions, including the serious purposes. The second phase, production, is the most essential process related to asset creation, source code creation, and the integration of both. This phase emphasizes on the design and implementation of program and visuals while also ensuring the learning content provided in the game. A prototype is produced as the product of this phase and will be tested on the next phase.

Testing, as the third phase, is performed to assess the functionality and playability of the game. Usually, alpha testing and beta testing are performed. Alpha testing is performed by internal testers, while beta testing is carried out by external testers. The final phase in this study is release, which involves game launching, project documentation finalization, and marketing. This step should explain how the game is launched step-by-step, post-production activities, and other activities needed to expose the game. Today's popular methods of launching include early access or demos, where players can try the game before its final version is released. This approach facilitates quicker player acquisition while also obtaining feedback as early as possible [19].

B. *Wave Function Collapse (WFC)*

In an attempt to automate game content creation, various pseudo-random numbers and other techniques have been widely used in PCG for games. Implementing PCG in game development brings significant benefits, including labour efficiency, greater creative expression for game designers, storage space savings, and potential unlimited replayability [20]. WFC is a procedural generation algorithm that produces images by arranging a set of tiles based on rules about which tiles can be adjacent to each other, as well as how often each tile should appear relative to others [15], [21]. Most research on WFC has focused on testing its ability to generate levels based on image data and expanding the design domain that WFC can accommodate [22], as it is popular for generating grid-based game maps.

1) *Pattern*

The term "pattern" refers to the depiction of repetitive occurrences of game area, characterized by local patterns contained within them. These patterns typically consist of sub-tiles (or images) that are only a few tiles wide and high (for example, an $n \times n$ tile window) [22]. In this context, the input pattern is called seed, while the output is the resulting map generated according to the input.

2) *Constraints Solving*

Constraint solving in WFC ensures that the output image matches the input patterns without breaking any rules. It ensures that the generated image follows specific constraints, such as keeping local patterns intact and avoiding conflicts. The process involves representing the problem as a constraint satisfaction problem (CSP), where variables represent the possible states of each tile (or pixel in image generation) in the output image, and constraints define the relationships between neighbouring tiles. These constraints ensure that the generated image conforms to the input patterns while maintaining coherence and consistency. For instance, adjacency rules are one type of constraint used in this context [22]. During the solving process, the algorithm iteratively selects and collapses tiles based on constraints and input patterns until either a solution is found or it encounters a contradiction, indicating that the constraints cannot be satisfied. Constraint solving algorithms in WFC typically employ backtracking or other search techniques to efficiently explore the solution space and find a valid configuration for the output image.

In this study, we are applying constraints in the form of seed tiles and game pacing. The seed is n×n tiles as a sample input. The game pacing is calculated with the main focus only on the tempo aspect. As [13], [17] stated that the implementation of pacing aspect in the game is might be intuitively tailored according to the designer's preferences.

3) Entropy

In WFC, entropy denotes the degree of uncertainty or disorder within the system, specifically reflecting the unpredictability of patterns at each grid location in the output image. As the algorithm assigns probabilities to various patterns based on input constraints, higher entropy in a cell indicates a greater range of potential patterns, while lower entropy signifies fewer options. During image generation, cells with lower entropy are prioritized for resolution as they contribute to reducing uncertainty and enhancing coherence. By focusing on cells with lower entropy first, the algorithm efficiently explores the solution space, leading to a more coherent and structured final image output [22].

C. Pacing

As we mentioned earlier, understanding game pacing is crucial as it can indicate the distribution of designed difficulties and can be analysed at each segment of the game [13], [14], [23]. Designer can gain deeper insights into the level design and utilize the relative risk values and their distribution across different path clusters as a guide to adjust the level design [24]. This helps in balancing the overall risk or introducing interesting variations in strategic choices for players.

Game pacing can also be measured by comparing the gameplay data of different game groups or parties at each timestep [14]. For instance, the speed of progression through the game environment can be used as an indicator of pacing, and displayed the recorded gameplay data in a heatmap. Additionally, providing "fit" pacing to the players is beneficial because players will be immersed in an experience with the appropriate skill-challenge ratio, thereby placing them in a conducive environment for serious purposes [25], [26].

Pacing is the flow or rate of an activity, while in the context of games, game pacing refers to the pace of events influenced by the complex and cumulative aspects of threats, movement impetus, and tempo for each segment in the game level [13], [17]. In the context of tempo, it refers to the length and timing of player actions. Generally, a higher intensity of actions and a higher density of objects (enemies, obstacles, and treasures) in a segment will result in a higher tempo. In this study, we want to apply the WFC to the map generation and implement the concept of pacing as a constraint to control the resulting player experience set by designer. However, we are only focusing one pacing aspect, namely tempo.

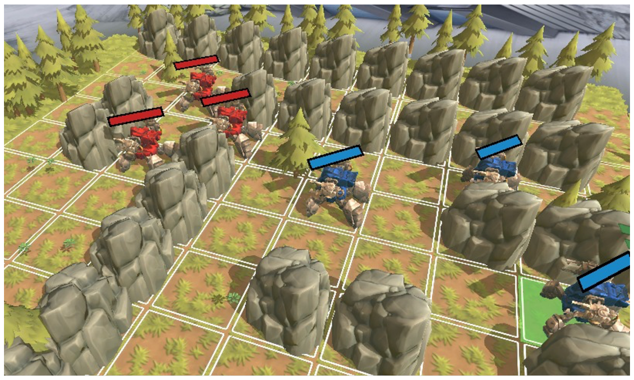
Figure 1. Turn-based tactics game MECH.AI; red team vs blue team.

3. METHODOLOGY

This research aims to implement an efficient and high-quality solution for automatically generating maps for the serious, turn-based tactics (strategy) game MECH.AI while considering the pacing aspect of tempo in game design. The proposed approach involves implementing the WFC algorithm and using map generation data in the pre-battle phase as an experiment. Testing will be conducted to measure the model's ability to create maps that align with the tempo set in the game design by the designer. The test results are expected to yield a procedural model capable of producing efficient and high-quality maps in MECH.AI.

A. MECH.AI

MECH.AI, developed by Game AI Code Lab (GACLab), is a turn-based tactical game designed to enhance students' engagement in learning fundamental programming concepts. In this game, players command robots in strategic battles within a 10 × 10 grid arena filled with obstacles like rocks and trees. The goal is straightforward: eliminate all enemy robots. Players must wisely choose actions: move, attack, or special attack where each has different energy cost. Each team features a unique mix of robots based on their equipment. Players can control robots via a graphical user interface (GUI) or through the command line, assuming the role of pilot engineers directing mecha robots. Our ultimate goal is to provide a feature where players can submit their own AI scripts to control the robots. The game is currently in development following the GDLC and will continue to progress to achieve the best results. Fig. 1 illustrates the gameplay of MECH.AI.

1) Development

As mentioned earlier, MECH.AI is intended to be a serious game where players learn and practice programming through gameplay contents. The ultimate goal of the game is not only to control the robots via GUI or command line but also to allow players to submit AI scripts and simulate battles against opposing parties. This feature provides players with the freedom to experiment and observe the underlying logic, thereby spreading basic programming and

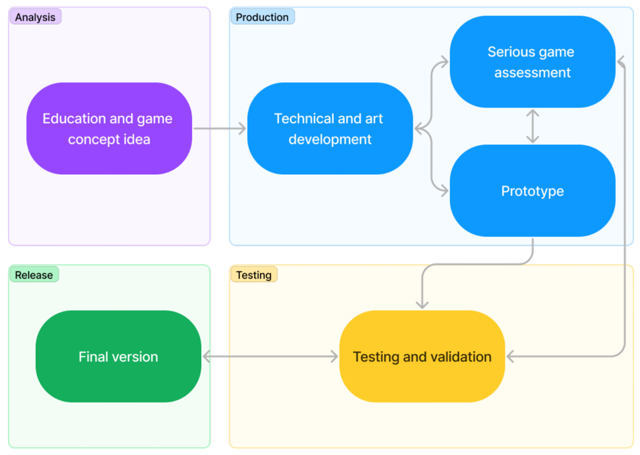

Figure 2. Turn-based tactics game MECH.AI; red team vs blue team.

AI foundations to the public. For instance, $Move(a, x, y)$ is the command to move active robot a to a target tile with coordinates x and y. The analysis is performed as the first phase of the development to map the serious purposes we want to achieve and the game infrastructure. The development cycle is shown in Fig. 2.

The production phase results in a prototype. In this phase, we develop the technical specifications of the game design as well as the serious game aspects. For instance, we designed a tutorial level to familiarize players with the gameplay. Most of the 3D assets are obtained from the Unity Asset Store for rapid and convenient development. The game prototype is then tested in GACLab, focusing on the game concept and core mechanics. The Minimum Viable Product (MVP) of the game is released on GitHub as an open-source project to gather valuable feedback.

In MECH.AI, before the battle starts, players need to deploy their robots on the designated deployment tiles. After all robots are deployed, each player will have a turn to perform actions. Generally, there are four actions a player can take: move, attack, use a skill, or idle (to regain energy).

2) Map Representation

In our game, the map is represented by a combination of tiles in a 10×10 grid. A tile is a single occupiable area where a robot's actions take place. There are two main types of tiles: passable and obstacle. A passable tile is one that players can move through, while an obstacle tile contains a destructible blocking environment. The map is generated using WFC and is designed to be playable according to the game pacing set by the designer. The presence of obstacles requires players to think strategically. For instance, a tree may take one hit to destroy, while a rock may have three hit points. However, in this study, we only experiment with one type of obstacle as a proof of concept, namely rocks. Fig. 3 shows a sample map in MECH.AI.

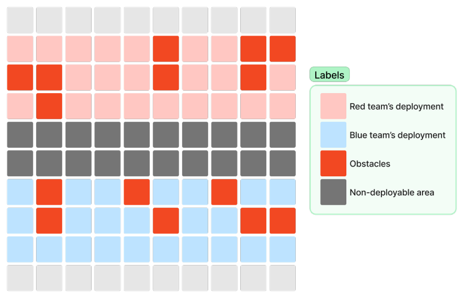

Figure 3. Map representation in MECH.AI.

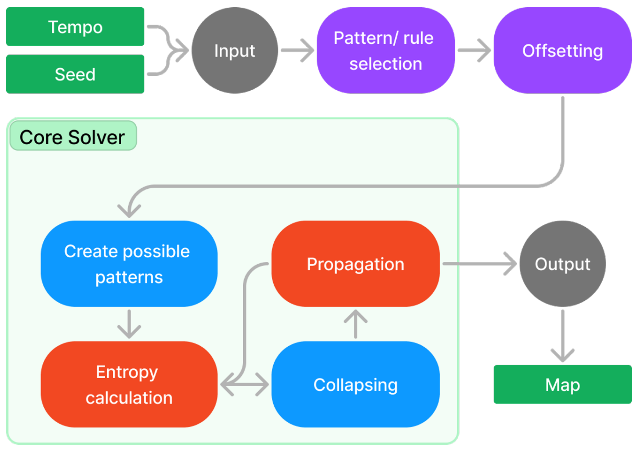

Figure 4. The overview of WFC used in this study.

B. Map Generator

The WFC system generated a map in the pre-battle state. Within the framework of WFC, various processes are executed. An explanation of how the map generation process through WFC works is shown in Fig. 4.

1) Seed Patterns & Adjacency Rules

The patterns on the map are created from the combination of tiles and patterns (input seed). Pattern refers to collections of n×n, each containing visual and gameplay elements designed by the designer. In this study, we set up experiments with pattern sizes ranging from 3×3 to 5×5. The formation of complex patterns from smaller pattern elements requires input that defines the interaction rules between patterns or tiles, presented simply by an input pattern. The seed pattern consists of small parts of the set of tiles used as a reference to establish larger patterns, representing the adjacency rules. Seed patterns have a smaller area than the output map to be generated, so the obstacle ratio in the output map tends to be almost the same as that in the seed pattern. Adjacency rules refer to conditions that govern the relationship between neighbouring elements. In the context of WFC, the function of this rule is to determine the patterns or tiles that may appear in each direction from each element.

2) *Entropy, Weight, Propagation*

Entropy indicates the uncertainty in a system. It is used to assign weights to each tile to generate the most probable neighbouring tiles once one is established, usually represented by a weight value. Entropy is part of the system strategy to determine pattern selection or certain values during the pattern propagation process. The concept of entropy is also utilized to evaluate how chaotic or complex a particular area is, which ultimately can influence the selection of values or patterns during propagation. Propagation in map generation via WFC involves spreading constraints and information throughout the map to influence neighbouring tiles or cells, ensuring consistent enforcement of constraints across the entire map.

3) *Core Solver*

The process begins with the construction of a seed pattern from a sample (input), smaller map. The input is then converted into a 2D matrix containing the indexes of tile types. Fig. 5 shows the core process in WFC, while step 1 in Fig. 5 depicts the tile conversion. The next step is to identify patterns as the basis to determine the relationship between tiles and the complexity space of WFC. Generally, we can detect patterns in the input by using an $n \times n$ tiles filter, please, refer to step 2. In this study, we do not apply rotation to the filter, meaning the pattern is simply obtained from the seed as it is, without any image transformation techniques. For instance, if there is a passable tile surrounded by tree tiles as shown in seed C in Fig. 6, the system cannot detect if the passable tile is on the right corner of the pattern as if mirrored. Moreover, we add offset (step 3), additional tiles on the edge of the seed, ensuring the edge tiles have possible neighbors.

The 10×10 tiles grid of the output map is prepared. Each tile initially cont ains all possible tile types, as shown in step 4. Later, each tile type will be determined, and the weight of each type will be calculated according to frequency and adjacency rules. The starting tile is randomly chosen. Then, each cell will undergo a collapsing process to converge to the final type for the tile by the core solver, where the main WFC process occurs. The solver performs entropy calculation, applies the constraint distribution to all possible tiles, and finally collapses the tiles. The weights of neighboring tiles are calculated by spreading to the surrounding neighbors, deciding which cell will be collapsed based on the calculation; generally, the lowest entropy is chosen. This process is iteratively applied to all cells until all cells collapse. Equation (1) shows the entropy calculation, and step 6 in Fig. 5 depicts the propagation simulation in collapsing cells. Please note that the implementation of entropy calculation may differ according to the specific case. In this study, we define the entropy S as the log of the sum of all weights from a map $Log_2(\sum W)$ reduced by the average of the log of the weight of each pattern.

$$S = Log_2(\sum W) - \frac{\sum_{i=1}^{n} Log_2(\sum W_i)}{\sum W} \tag{1}$$

4) *Segmentation*

In this study, we aim to achieve balanced patterns on the map segments in terms of obstacle distribution. We decide to segment the output map into four segments, each consisting 5×5 tiles. We expect by segmenting the map into four, we patterns will somehow more balance while also not too symmetrical. To achieve this, we break the WFC process into four smaller segments, with each segment offsetting to the neighbouring segment to ensure a seamless connection between them. Refer to step 5 in Fig 5. for the illustration.

5) *Pacing Implementation*

The evaluation step is performed as the final process to assess whether the generated map conforms to the target pacing aspect set by the game designer, namely tempo. In this study, we only applied the pacing aspect of tempo as a proof of concept. The implementation of pacing dynamics in games may vary depending on the designer's definition. However, it should not be limited to tempo; other pacing aspects such as movement impetus and threat can also be applied.

In this study, we design and observe two game design patterns as factors of tempo: deployed obstacles and maze patterns. The deployed obstacles factor focuses on analysing the number of obstacles in the deployment area for both teams. We presume that more obstacles on the deployment tiles will result in more actions taken by the player, such as moving around or attacking obstacles, thereby creating higher tempo. Conversely, a lesser number of obstacles will result in fewer actions, lowering the tempo. In our game design, during the pre-battle phase, players deploy their robots in a designated safe area to prevent the second player from gaining an advantage by having more flexibility in choosing where to deploy their robots. As illustrated in Fig. 3, the possible deployment area is represented by light red and light blue tiles, and the dark grey tiles represent the non-deployable robot areas.

It is notable that the ratio of obstacles in the seed pattern as input will affect the resulting tempo. According to the sample case shown in Fig. 3, we construct (2) to represent the number of obstacles in deployment tiles, denotes as O. The number of tiles in the possible deployment area is represented by t_D and the number of obstacles in the seed pattern is represented by O_s. The ratio of obstacle in the seed pattern is R, while T_I is the intended tempo or target set by designer. Equation (2) implies the number of obstacles generated is determined by the lower value between the number of deployable tiles and the obstacles ratio in relation to the expected pacing. In theory, the system will tend to choose the latter. Finally, T_A is the actual value of tempo currently being calculated, which is derived by the difference between number of obstacles in map and the number of obstacles in seed pattern, divided by the ratio multiplied by the number of tiles in deployment are, as shown in (3). We refer this equation as the first tempo factor (regarding obstacles in deployment area) or tempo A.

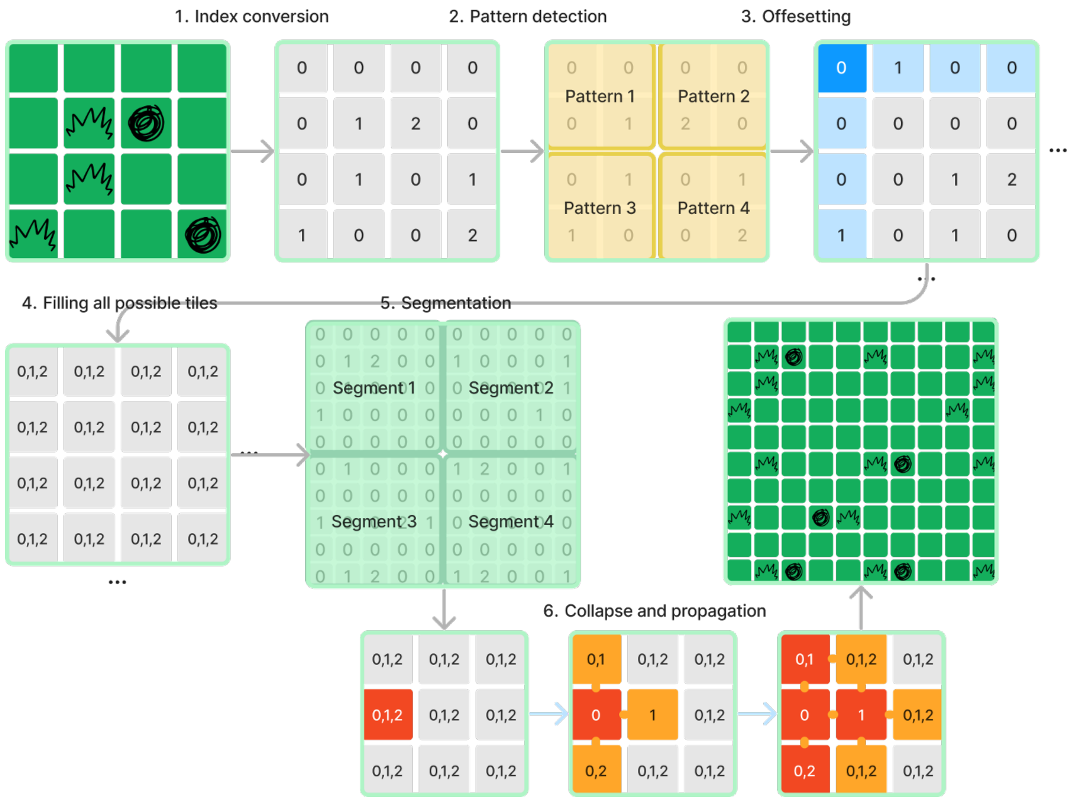

Figure 5. The flow of map generation using WFC in this study.

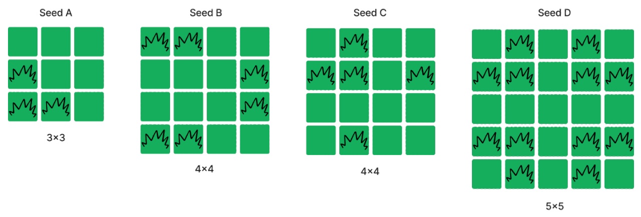

Figure 6. The sample seed patterns used as input for experiments.

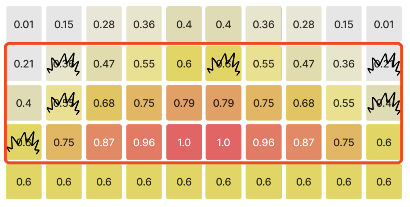

Figure 7. The resulting heatmap of tile weights from top half of a battle map with 10×10 size.

$$O = min(t_D, ((R \times t_D) + O_S) \times T_I)) \tag{2}$$

$$T_A = \frac{O - O_s}{R \times t_D} \tag{3}$$

The second factor affecting the tempo is referred to as maze pattern or tempo B. It focuses on determining the optimal position to place the obstacles by providing obstacles that enforce the player with doing more movement or destroying the obstacle based on the weight of the tiles. Thus, we presume high activities on the robots will raise the tempo, while lower activity levels result in a lower tempo. To identify the weight of each tile, we simulated a pathfinding algorithm using A* and assigned the weight according to the likelihood of them being passed through. We iteratively chose a starting point $s(x_i, y_i)$ on the, let's say, team A's area and iteratively select the target tile $t(x_i, y_i)$ in team B's area until all target tiles in team B were simulated. The loop stop after all possible starting point in team A are simulated. Then redo the simulation with the team B's area as starting points. This ensure more accurate results if the map is not symmetrical for both teams. The results of tiles' weights are shown in a heat map in Fig. 7.

It can be concluded that the tiles with heavier weights tend to converge in the middle of map while the lighter ones are more towards the sides. Please note that the tiles around rows three and four have the same weights because those are intended to be non-deployable areas, meaning the players are not allowed to deploy the robot there. These tiles mean to be the neutral area that bridges the both teams' area. We construct the equation to measure the tempo of the maze pattern or simply tempo B (T_B), as shown in (4). The list of tile weights W with indices i to n from the output map, sorted in ascending order, is represented by L_w where $L_W = [W_i, W_{i+1}, ..., W_n]$. The bottom limit and upper limit for the normalization of the tile weight list is represented by L_n and L_m, respectively. The T_A and T_B will then be averaged according to the weights W_A and W_B set by designer to get the overall tempo T.

$$T_B = \frac{\sum(W_i - L_n)}{L_m - L_n} \qquad (4)$$

4. EXPERIMENTS & RESULTS

In this section, we performed and analysed the two tempo factors in the WFC-generated turn-based tactics game map, in MECH.AI. The first factor T_A focuses on the number of obstacles spread in the deployment area, and the second factor T_B, focuses on the obstacles placement that are positioned to create maze-like structure that increases player movement. There are three main categories and objectives of experiments as shown in Table I. In all three categories, the size of the output map generated is consistent, which is 10×10. The sample seed patterns used as input are shown in Fig. 6. Each experiment was performed in 100 runs.

A. Initial Experiment

The objective of this experiment is to evaluate the fitness of output maps using seeds A, C, and D, with size of 3×3, 4×4, and 5×5, respectively. The target tempo values T_I are set to 0.25, 0.5, and 0.75, where 0.25 represents low tempo, 0.5 represents balanced tempo, and 0.75 represents high tempo. The obstacle ratios R are 0.33 for seed A, 0.31 for seed C, and 0.48 for seed D. In Table II, it can be observed that the fitness of T for seed A is 0.99, for seed B is 1.0, and for seed C is 0.89, which is the lowest for the target tempo ($T_i = 0.25$). From the second target tempo ($T_i = 0.5$), seed D still results in the lowest fitness with 0.71, while seeds A and C have fitness of 1.0 and 0.99, respectively. For the high target tempo ($T_i = 0.75$), seed A produced a fitness of 0.9, seed C produced 0.94 fitness, while seed D only produced a fitness of 0.55.

From Table II, we can imply that seeds A and C are closer to the expected result, and the ratio of the obstacles has a significant impact to the resulting tempo. Although seeds A and C have different size, their obstacles ratio settings produced the better fitness. Moreover, the target tempo value also influencing the output tempo, as seen in seed D, where there are notable differences in fitness from low to high tempo. However, the ultimate solution to this problem lies in the complexity of the seed pattern, as it affected the core process of wave function collapse. Generally, the more complex the seed, the higher the probability of errors occurring and the longer it takes to generate results. Although the generation times indicate that smaller seeds require more calculations, resulting in

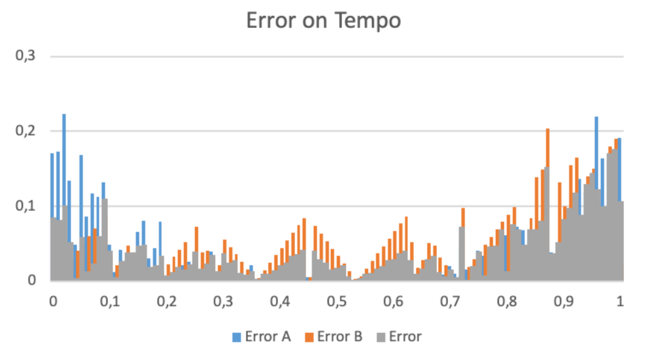

Figure 8. Error distributions across tempo A, B, and overall.

longer generation times, this complexity trade-off is evident in the average generation times: seed A has the highest average generation time t at 9.85 seconds, while seeds C and D require only 0.35 and 0.13 seconds, respectively. This analysis underscores the importance of balancing seed complexity with obstacle ratio to achieve optimal fitness and efficient generation times in procedural map generation for MECH.AI.

B. Single-Seed Experiment

In this experiment, we aim to analyse the resulting tempo with various targets. Similarly, this experiment run in total of 100 iterations using seed B, with 5 inner-iterations within. An initial tempo value of 0.0 was given, with a tempo increase of 0.01 for each iteration. Based on the experiment, the results are quite stable from tempo value above 0.1 and below 0.9. According to Fig. 8, the averaged error on tempo A is 6.35%, and the averaged error on tempo B is 3.48%, making the overall tempo error of 4.83%. It is observed that the highest errors occur in 0 - 0.1 tempo target and 0.9 - 1. This might happen depending on the seed pattern design and the target tempo that in some cases may not converge. For example, if the designer has a seed pattern with high obstacle ratio, then it will be suitable for higher pacing target rather than a low one.

Arguably, the system produced the expected tempo with the low error rate less than 5%. Fig. 9 shows the sample output from various tempo using seed B. According to our observation, the most preferred outputs are produced with approximate $T = 0.2$ to $T = 0.8$, with around $T = 0.5$ being the best. These outputs appear more human-crafted-like level rather than generated solely by a system.

C. Seeds Experiment

The last experiment aimed to observe similar target or expected tempo set by designer using different seeds. As we mentioned earlier, we use seed A, B, C, and D as shown in Fig. 6. In this case, since $T = 0.5$ has produced the most balanced and expected fitness, we used it as target. Each seed was run through 50 simulations independently. According to the output fitness in Table III, all seeds produced the expected tempo with the average

TABLE I. Experiments and Objectives

Experiments	Objectives
Initial	Conducting trials and analysis to ensure that the system operates according to the seed pattern and tempo values that have been set as target.
Single-seed	Performing analysis on changes or decreases in pacing aspect tempo using a seed pattern with the interval 0.1.
Seeds	Conducting analysis on four different seed patterns to achieve similar pacing tempo values.

TABLE II. Fitness on Target Tempo across Seeds

Seeds	0.25 (low)	0.5 (balance)	0.75 (high)	t (s)
A (R = 0.33)	0.99	1	0.9	9.85
C (R = 0.31)	1	0.99	0.94	0.35
D (R = 0.48)	0.89	0.71	0.55	0.13

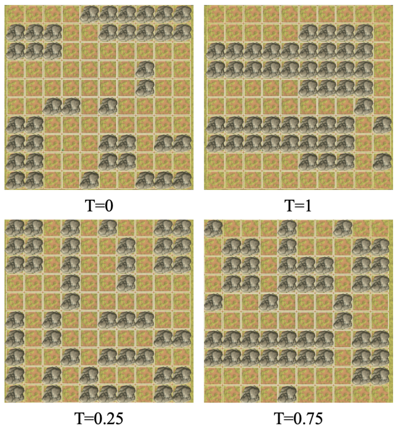

Figure 9. Sample output maps by using various tempo values.

overall tempo $\bar{T}$ = 0.97. Additionally, across seed patterns, $\bar{T}_B$ produced 4% error, and $\bar{T}$ produced 3% error, while the system perfectly converged to fit to $\bar{T}_A$. However, one might notice that the output samples from seeds C and D look less human-crafted (Fig. 10). As we mentioned before, this might be happened because the high obstacle ratio R in the seed. The case in D can be addressed with reducing the obstacle on the seed pattern. Unfortunately, output from seed C also has the less human-craft feel because the obstacles are converged to some area. The simplest way to handle this problem is to change the design of obstacle pattern to be scarcer or more spread out.

D. User Study

We conducted an additional user study to evaluate the results produced by our system. The respondents consisted of 20 students majoring in Computer Science at Universitas

TABLE III. Fitness on All Tempo across Seeds

Seeds	T_A	T_B	T
A (R = 0.33)	1	0.95	0.97
B (R = 0.31)	1	0.93	0.96
C (R = 0.31)	1	0.97	0.98
D (R = 0.48)	1	0.99	0.98
Avg.	1	0.96	0.97

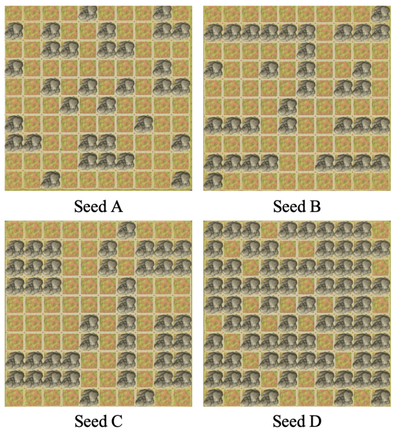

Figure 10. Sample output maps by using various seeds with T = 0.5.

Dian Nuswantoro, Indonesia. They were asked to answer questionnaire containing four questions (Q1-Q3). The purpose of this user study was to observe the respondents' perspectives in experiencing the maps produced by our WFC map generator.

1) *Q1. Which of the following maps has an approximate medium tempo (0.5)?*

The respondents were asked to experienced two maps sequentially and choose one they perceived to have medium tempo (0.5). One of the options presented was a map with the set tempo of 0.5, while the other had a significantly different tempo. This question is aimed to evaluate if the output map produced the expected actual experience based on the tempo set. According to the responses, 17 out of 20 respondents (85%) correctly identified the map with a medium tempo of 0.5.

2) *Q2. Although being asymmetrical, do you think the map is balance for both teams?*

Respondents were asked to experience a map and assess whether it provided balanced area despite being asymmetrical on both sides. The respondents are able to play as both teams. The purpose of this question was to determine if the asymmetrical maps generated by the WFC algorithm could still offer a fair and balanced experience for both teams, which is crucial for the strategic and competitive nature of MECH.AI. Based on the responses, 18 out of 20 (90%) respondents answered with "yes", indicating that they felt the map was balanced for both teams. Please refer to Fig. 11 for the battle map being inspected.

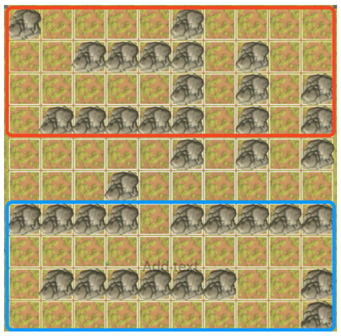

Figure 11. An example of output map. Red line depicts deployment area for team A, and Blue line depicts deployment area for team B.

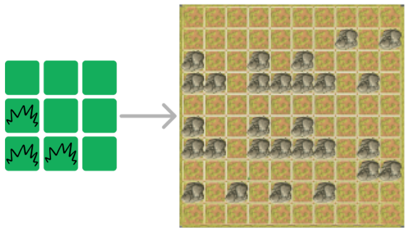

Figure 12. Input: Seed pattern with size 3×3 (left), and Output: battle map with size 10×10 (right).

3) *Q3. Do you think the maps (10×10) represent the pattern in little maps (3×3))?*

Respondents were asked to determine whether the output maps adequately represented the seed patterns (A-D). They were provided with four seed patterns (A-D) and corresponding sample output maps to make their evaluation. The answers are then averaged into 18 out of 20 (90%) respondents agree that the map represent the seed pattern. The purpose of this question was to assess the effectiveness of the WFC algorithm in scaling up from small seed patterns to larger, playable maps while maintaining the visual and structural integrity of the original patterns. This helps to validate the consistency and reliability of the procedural content generation process. Fig. 12 shows an example of seed pattern and output map.

5. Conclusions and Future Work

In the roadmap to create a serious game to enhance the students' initial motivation and knowledge in learning basic programming and AI fundamental, we are developing turn-based tactics game called MECH.AI. The challenge in the form of battle map is generated by embedding the pacing aspect tempo and the Wave Function Collapse (WFC) algorithm into the battle map generator. The pacing aspect tempo functions as the target set by game designer and constraints that drives the core process of WFC. The experiments show that the map generator is able to produce battle map that tend to meet the target. While the analysis and user study are also provided.

While WFC is not new algorithm for generating battle maps, it is an interesting technique to experiment with, especially when embedding the concept of game pacing to produce the expected experience for players as controlled by the designer. In our WFC implementation, we segment the whole output map into four segmentations with the aim to generate fair challenge (obstacle) while the layout is still asymmetrical. Keeping the balance contents over the different areas of the map even if it is not symmetrical. In addition, we implemented the concept of game pacing tailoring in a strategy game, turn-based tactics specifically, continuing and confirming the studies done in controlling the game content generation in dungeon-crawler and platformer games done by [13] and [17]. The general idea of combining game pacing into the procedural game content generation is to create a more abstract or less-technical layer of content generation. In these cases, the game pacing, as it is an important element of one of the game design elements, game narratives.

As our study's limitation, it is important to note that smaller seed pattern requires longer generation time as the size of the filter to do the calculation is small, resulting in more possible cells to be calculated. According to our observation, there is a tendency that the extreme low tempo ($T \geq 0.1$) and high tempo ($T \leq 0.9$) to produce highest errors due to the design of seed patterns. Hence, a deeper study is needed. Although, we argue that the overall error ($\bar{E}_T \leq 5\%$) is still acceptable. Moreover, the problem can be easily addressed by a little tweaking the design of seed patterns and its obstacle ratio. Regardless, the generation process may help designer in providing contents such as for campaign or story modes where the tempo dynamic can be automatically set to fit the progress of the story. Finally, based on our unpresented experiment, we recommend the designer at least place one obstacle on the seed pattern and not fill all the tiles with obstacles ($R > 0, R < 1$), otherwise the result could not be possible to produce fitter outputs.

References

[1] N. K. Ningrum, A. B. Harisa, and L. Umaroh, "Learning through play: Utilizing board games to enhance english vocabulary for early students," *TECHNO CREATIVE*, vol. 1, no. 2, pp. 117–122, 2024.

[2] D. López-Fernández, J. Mayor, J. Pérez, and A. Gordillo, "Learning and motivational impact of using a virtual reality serious video game to learn scrum," *IEEE Transactions on Games*, 2022.

[3] D. Lambić, B. Đorić, and S. Ivakić, "Investigating the effect of the use of code. org on younger elementary school students' attitudes towards programming," *Behaviour & Information Technology*, vol. 40, no. 16, pp. 1784–1795, 2021.

[4] C.-C. Tsai, "The effects of augmented reality to motivation and performance in efl vocabulary learning." *International Journal of Instruction*, vol. 13, no. 4, pp. 987–1000, 2020.

[5] S. Bassanelli, A. Bucchiarone, and F. Gini, "Gamidoc: The importance of designing gamification in a proper way," *IEEE Transactions on Games*, 2024.

[6] K. Unlu *et al.*, "Determinism versus stochasticity in the action econ-

omy of turn-based tactics games," Master's thesis, Aalto University, 2023.

[7] Steam, "Turn-based tactics," https://store.steampowered.com/tags/en/Turn-Based+Tactics, 2024, accessed: 2024-05-15.

[8] S.-G. Nam, C.-H. Hsueh, and K. Ikeda, "Generation of game stages with quality and diversity by reinforcement learning in turn-based rpg," *IEEE Transactions on Games*, vol. 14, no. 3, pp. 488–501, 2021.

[9] J. Gao, "The computational complexity of fire emblem series and similar tactical role-playing games," *arXiv preprint arXiv:1909.07816*, 2019.

[10] R. Ramadan and Y. Widyani, "Game development life cycle guidelines," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 2013, pp. 95–100.

[11] S. Aleem, L. F. Capretz, and F. Ahmed, "Game development software engineering process life cycle: a systematic review," *Journal of Software Engineering Research and Development*, vol. 4, pp. 1–30, 2016.

[12] D. Petko, R. Schmid, and A. Cantieni, "Pacing in serious games: Exploring the effects of presentation speed on cognitive load, engagement and learning gains," *Simulation & Gaming*, vol. 51, no. 2, pp. 258–279, 2020.

[13] A. B. Harisa and W.-K. Tai, "Pacing-based procedural dungeon level generation: Alternating level creation to meet designer's expectations," *International Journal of Computing and Digital Systems*, vol. 12, no. 1, pp. 401–416, 2022.

[14] M. Ashton and C. Verbrugge, "Measuring cooperative gameplay pacing in world of warcraft," in *Proceedings of the 6th International Conference on Foundations of Digital Games*, 2011, pp. 77–83.

[15] H. Kim, S. Lee, H. Lee, T. Hahn, and S. Kang, "Automatic generation of game content using a graph-based wave function collapse algorithm," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–4.

[16] A. Sandhu, Z. Chen, and J. McCoy, "Enhancing wave function collapse with design-level constraints," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–9.

[17] A. B. Harisa, S. Nugroho, L. Umaroh, and Y. P. Astuti, "Threat construction for dynamic enemy status in a platformer game using classical genetic algorithm," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 2023.

[18] R. Roedavan, B. Pudjoatmodjo, Y. Siradj, S. Salam, and B. Hardianti, "Serious game development model based on the game-based learning foundation." *Journal of ICT Research & Applications*, vol. 15, no. 3, 2021.

[19] D. Lin, C.-P. Bezemer, and A. E. Hassan, "An empirical study of early access games on the steam platform," *Empirical Software Engineering*, vol. 23, pp. 771–799, 2018.

[20] Y.-C. Chen, S.-R. Li, and I.-C. Liu, "Attractiveness of mobile games—a case study of tile-matching games," in *2018 IEEE International Conference on Applied System Invention (ICASI)*. IEEE, 2018, pp. 642–645.

[21] D. Cheng, H. Han, and G. Fei, "Automatic generation of game levels based on controllable wave function collapse algorithm," in *Entertainment Computing–ICEC 2020: 19th IFIP TC 14 International Conference, ICEC 2020, Xi'an, China, November 10–13, 2020, Proceedings 19*. Springer, 2020, pp. 37–50.

[22] I. Karth and A. M. Smith, "Wavefunctioncollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–10.

[23] N. Baumann, C. Lürig, and S. Engeser, "Flow and enjoyment beyond skill-demand balance: The role of game pacing curves and personality," *Motivation and Emotion*, vol. 40, pp. 507–519, 2016.

[24] J. Tremblay, P. A. Torres, and C. Verbrugge, "Measuring risk in stealth games." in *FDG*. Citeseer, 2014.

[25] G. B. Moneta, "On the conceptualization and measurement of flow," in *Advances in flow research*. Springer, 2021, pp. 31–69.

[26] N. Baumann, "Autotelic personality," in *Advances in flow research*. Springer, 2021, pp. 231–261.

Muhammad Alifian Aqshol is an alumnus of Universitas Dian Nuswantoro, majoring in Computer Science. He just earned his B.Sc in 2024. He is a member of Game AI Code lab (GACLab) and currently working as full stack engineer.

Ardiawan Bagus Harisa is a lecturer at Universitas Dian Nuswantoro, majoring in Computer Science, specifically game AI and PCG in game. He earned his M.Sc in Computer Science from National Taiwan University of Science and Technology at 2018. He is currently developing and supervising Game AI Code lab (GACLab).

Pulung Nurtantio Andono is a professor at Study Program of Information Engineering, Faculty of Computer Science at Universitas Dian Nuswantoro. He received his B.Sc in 2006 from Universitas Trisakti, M.Sc in 2009 from Universitas Dian Nuswantoro, and P.hD in 2014 at Institut Teknologi Sepuluh November.

Wen-Kai Tai is a professor of Department of Computer Science and Information Engineering at National Taiwan University of Science and Technology. He has been awarded his PhD from National Chiao Tung University. He is expert in the domain of computer graphics, procedural content generation, and also game computing and development technique.