International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 12, No.1 (Jul-2022)

https://dx.doi.org/10.12785/ijcds/120132

Pacing-based Procedural Dungeon Level Generation: Alternating Level Creation to Meet Designer's Expectations

Ardiawan Bagus Harisa[1] **and Wen-Kai Tai**[2]

[1]*Department 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 6 Mar.2021, Revised 1 Apr. 2022, Accepted 25 May. 2022, Published 20 July 2022

Abstract: Dungeon Crawler games are getting a greater prominent rise in attention in the present day. Therefore, managing the pace in dungeon games is necessary. In this paper, we generate dungeon levels with the game pacing as intended by the game designer by integrating the concept of the Mission and Space framework and Game Design Patterns to generate dungeon game levels with the intended pacing. The Mission and Space framework handles the generation of tasks and in-game geometry spaces. We also present Pacing Patterns as patterns of the pacing on a game level to drive the level generations. We use a genetic algorithm to generate the potential level candidates. Those processes are encapsulated in a mixed-initiative tool, Pacing-based Dungeon Generator, to generate meaningful dungeon levels for players based on game designer preferences of game pacing. The proposed approach can minimize both time and expenses used to create game levels and effectively provide a more formal approach for game designers. Moreover, it keeps the players' progression and experience as what the designer expected.

Keywords: Procedural Level Generation, Game Design Pattern, Game Pacing, Pacing Patterns, Genetic Algorithm

1. Introduction

Dungeon crawler games are regaining more attention in last recent years. Steam [1] is a gaming platform made by Valve where players can buy, gather information, and discuss digital games. Based on data from the Steam platform alone, more than 350 games have been categorized as action, rogue-like games (also reflected as dungeon crawler games) since 2014. Although some said that the market of the genre is a "*niche market*", more than 270 games are in "*mostly positive*" and "*positive*" feedback from the player, meanings players or buyers prefer over 75% of the distributed title on the platform. Respectively, there are 21, 33, and 26 games that fall into the genre of action rogue-like from 2017 to 2019. In 2020 and 2021, around 63 and 104 games fall into the same genre. It also means the popularity of the genre is still increasing since. For additional information, since 2017, the game containing the title with the keyword "*dungeon*" is also increasing. There are more than 4000 games in total searched with the keyword even though the mechanics of the games may not represent the dungeon crawler games.

Some games have successfully implemented the procedural generation of dungeon maps, including *Diablo*, *Spelunky*, and *Enter the Gungeon*. The generated level may increase the variations and replayability of the game. Especially in dungeon-crawler games, providing accurate game pacing to the player is one crucial task that should engage the player better in the game [2], [3]. Unfortunately, only a few articles discuss game pacing or pacing in games to fulfil players' excitement. Thus, the definitions of game pacing vary among the designers, making the game pacing poorly defined and comprehended. Moreover, if a system shows the pacing information on the game level, the designer will have a better understanding to make that level more interesting for the player. The designer provides a meaningful dungeon level for the player based on the designer's preferences of pacing patterns by using such a system. The concept of design patterns [4] and procedural level generation [5] has been used in the game design domain, and we can use such concepts to provide meaningful dungeon levels to players [6]. In this study, we use pacing patterns to implement game design patterns.

We proposed an alternative approach that minimizes the time and expenses used to create a game level and a formal approach for the game designer to tailor the game pacing procedurally. Moreover, the system can keep the players' progression and experience by providing various level candidates that follow the game designer's expectations.

2. Related Work

This related-work study aims to convince us that by using design patterns in the procedural level generation,

E-mail address: ardiawanbagus@dsn.dinus.ac.id, wktai@mail.ntust.edu.tw

we can provide meaningful game pacing for better player engagement.

A. Procedural Level Generation

Joris Dormans [7] proposed an approach to construct an action-adventure game's level by distinguishing the creation of a level into two structures: missions and spaces. The mission structure is a series of tasks of the player, while the space structure is the geometrical layout of the game spaces. Dormans claimed that using the mission and space framework with the application of shape grammar is an efficient way to generate high varieties of quality levels for action-adventure games, although the resulting levels are tightly dependent on the quality of the grammar used in the rules to drive the generation process.

B. Game Pacing

Davies [8] analyzed what constitutes level pacing and identified several key aspects of game pace: movement impetus, threat, tension, and tempo. Movement impetus is the will or desire of a player to move forwards through a level. The threat is the notion of danger given by the game entities, while tension is the perceived danger that occurs from the belief of an unknown danger. Hence, achieving the perceived danger (tension) is hard. The easy way to distinguish between threat and tension is that tension is created from the fear of the unknown, and threat must be known in some ways. In this paper, we merge the tension into the threat. The last aspect of game pacing is tempo, which mainly describes the level of intensity and timing of actions from the player. Game pacing can be driven by the designer or the player's preferences, parameterized by the game's dynamics and aesthetics [3].

Metrics to evaluate the procedural level design are proposed in [9]. Some of the metrics that were proposed are related to the threat level, rhythm, completion time, and awareness. The metrics of threat, rhythm, and awareness that have been proposed conform with the aspects of pacing by Davies [8]. The rhythm is similar to the tempo, and the awareness is arguably related to the movement impetus that corresponds to what was previously described in [8]. These metrics can be evaluated quantitatively. However, different metrics implementation might be needed for two different games [10]. Therefore, the measurement of each metric is still abstracted.

Among the definitions of game pacing in [3], [8], [11], we summarize game pacing as the rate flow of activity in a video game, such as movement, attacks, or any others that affect the player's experiences and it can be separately observed on the several pacing aspects, such as threat, movement impetus, and tempo.

C. Mixed-Initiative Tool

Mixed-initiative tool for game level is a tool that helps the designer to create a game level interactively with the input parameters set by the designer. Launchpad [12], a level generator for 2D platformer games, is built on a rhythm-based game model. The system needs embedded verbs as commands to generate levels.

Ludoscope [6] uses transformational grammar to generate the content. It allows the designers to use some types of grammar and set up commands/recipes to generate content tailored to particular types of games (i.e., dungeon and platformer games). However, the system is still in an experimental state.

Another tool by Wang [13], called Dungeon Generator, is a tool for helping level designers to create a complete level of *Rogue-like* [14] or *Darksouls-like* game starting from the generation of missions, spaces, and the game objects using some fitness functions on genetic algorithm [15] to drive the evolution of the level. Those fitness functions mainly focus on the threat of the enemies and trap behaviour, called the co-play pattern.

Gu [16] tried to automatically optimize the gameplay of a slot game based on the expectation of the game designer. He proposed a graphical curve tool for the designer to observe and control the gameplay experiences. The curve control is simple and useful, and it might be integrated with any level generator as an interactive UI to generate a game level with the intended pacing.

Table I shows the difference between related works and our proposed system. We focus on generating dungeon level using aspects of game pacing defined in this paper. The concept of pacing aspects and their factors can be implemented in the different game genres since it is not tightly related to the specific game mechanics to create flexibility in implementation by the game designer.

Table II shows the methods and properties from references using genetic algorithms in their procedural level generation taken from [2]. The designer can easily set the target pacing in our work by using graphs for a more intuitive user interface. Moreover, the designer can modify genetic-related parameters and local and global constraints to drive the generation process. Based on [2], [17] there are only a few systems are applied to 3D dungeons. We have implemented the proposed method in 3D dungeon-crawler games. Hopefully, our system can open new potential in the research on 3D level creation.

3. Methodology

Our system, the Pacing-based Dungeon Generator, is a generator that produces dungeon levels as the designer intended by setting up the pacing patterns and constraints.

A. Level Representation

We describe the representation of a game level in detail, including the elements of a game level, how we build the design patterns from those reusable components, and finally, the metrics that facilitate us to create pacing patterns.

TABLE I. The summary of the differences between our work and the references.

Ref.	**Description**	**Game genre**
[6]	Ludoscope uses transformational grammar and lets the designer setup recipe of rule to generate level. In experimental state.	2D dungeon
[7]	Use shape grammar to separate level generation into two processes: mission and space.	2D dungeon
[12]	Launchpad, a rhythm-based model to create platformer games. Need embedded verbs in the system.	2D platformer
[13]	Dungeon Generator, generates level guaranteed to have co-play pattern (a game design pattern) using genetic algorithm that focus on measuring fitness of the threat level.	3D dungeon, but not limited to
Our work	Pacing-based Dungeon Generator (PBDG), generates variants of dungeon levels using intended game pacing and constraints set by the designer. The concept of pacing across genre may differs, thus provide flexibility on the implementation for designer.	3D dungeon, but not limited to

TABLE II. Methods and properties across references.

Ref.	**Approach**	**Control**	**Output**	**Results**
[18]	Space tree mutation	Game story, fitness, player model	2D game world	Combination of premade locations
[19]	Tree mutation	Fitness, genetic parameter	2D dungeon	Tightly packed rooms connected by hallways
[20], [21]	Combining 4 genetic representation and 5 fitness functions	Fitness, genetic parameter	2D maze	Larger between combinations
[22]	Mixed-initiative: map sketch and constrained	User sketching and fitness	2D dungeon	Large, even beyond dungeon
Our work	As plugin/ tool: pacing patterns and constrained	Target pacing (threat, impetus, tempo), genetic parameter, local and global constraints	3D dungeon level, but not limited to	List of level candidates contains rooms and object within

1) *Game Level Structure*

Our dungeon level consists of many rooms. Each room is built from combinations of tiles. Every tile has a specific tile type that determines the information for the room to where it belongs. A level is constructed from interconnected rooms which do not necessarily have the same dimension and layout, as illustrated in Figure 1. The rooms on a level are varied, from 4 to 25 rooms. There are four different room sizes: Mini has 3x3 tiles, Small has 4x4 tiles, Medium has 6x6 tiles, and 9x9 tiles for Large. The number of exit connections is between 1 and 3, meaning that every room must have at least one exit connection, although we may delete unused exit connections if the connection has no neighbour room to connect to. We pre-assume the player paths, enemy paths, and paths to treasures in each room by computing them using a pathfinding algorithm. Figure 1 shows an example of a level containing eight rooms (left) with its pacing patterns (right) visualized by the threat, impetus, and tempo curves. In the first room, the player's start position is indicated by the green tile, while the red tile indicates the exit connection. In contrast, the hexagonal-like and red object in the first room indicates the weak and melee enemy. The brown rectangle and green-white coloured object in the seventh room represent the obstacle and the boss enemy, while the treasure is visualized with a small object that has a gold colour.

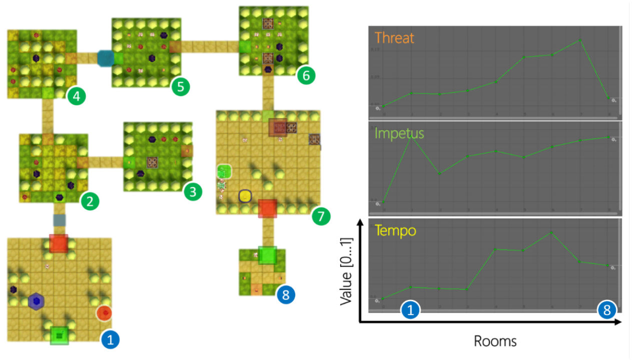


Figure 1. An example of a level and its pacing patterns.

Tile is the smallest level space on which a game entity can occupy. Each tile has a type, grid position, isEntrance (a boolean value to indicate if the tile is an entrance connection), isExit (a boolean value to indicate if the tile is an exit connection) and an enemy. There are six types of tile: None is a mark that marks a grid position, and Empty is an empty-passable tile that the player and the agents can traverse onto. Wall is the non-passable tile the entity can occupy, Treasure is the passable tile where a treasure is placed on, Enemy is a passable tile where an enemy will chase the player, and Obstacle is a passable tile where an obstacle is placed on top of it.

2) *Pacing Patterns: Aspects, Factors, and Metrics*

Game pacing is the rate of events affected by the values of aspects of threat, movement impetus, and tempo for each segment of the game that will affect the player's experiences. The game pacing pattern is a pattern related to the pace of the game. We limit our pacing pattern as a pattern of game pacing that occurs in the rooms-like shape, and present those patterns as a linear sequence of room pacing, as shown in Figure 1. The metrics we use are borrowed from studies in [8] and [9]. Please note that our formulas are intuitively designed since the different game genres may require different implementations of the pacing aspects, and pacing formulas in previous studies have not been proposed.

a) Threat

Threat relates to factors: damage, movement, and distance between sources of the threat (enemy and obstacle). We define the fitness function, f_{Threat} (calculated by using Equation 16), to reward the threat by the weighted sum of fitness f_{DmgE}, f_{DmgObs}, f_{DistE}, $f_{DistObs}$, and f_{MoveE} as shown in Equation 1 to 5 respectively. Where f_{DmgE} rewards total damage from all enemies, f_{DmgObs} rewards total damage from all obstacles, f_{DistE} rewards distance effectiveness from all enemies, $f_{DistObs}$ rewards distance effectiveness from all obstacles, and f_{MoveE} rewards movement effectiveness from all enemies. $DmgE_i$ is the damage of enemy i, AfE_i is the attack frequency of enemy i, ArE_i is the attack range of enemy i, DE_i is the distance of enemy i to the player path, SpE_i is the speed of enemy i, $DObs_i$ is the distance of obstacle i to the player path, $DmgObs_i$ is the damage of obstacle i, HP_{Player} is the health point of player, and Sp_{Player} is the speed of the player. N_E and N_{Obs} are the numbers of enemies and the number of obstacles, respectively.

$$f_{DmgE} = \sum_{i=1}^{n_E} \frac{DmgE_i \times AfE_i}{HP_{Player}} \tag{1}$$

$$f_{DmgObs} = \sum_{i=1}^{n_{Obs}} \frac{DmgObs_i}{HP_{Player}} \tag{2}$$

$$f_{DistE} = \sum_{i=1}^{n_E} \frac{ArE_i}{|DE_i - ArE_i|} \tag{3}$$

$$f_{DistObs} = \sum_{i=1}^{n_{Obs}} \frac{1}{DObs_i} \tag{4}$$

$$f_{MoveE} = \sum_{i=1}^{n_E} \frac{SpE_i}{Sp_{player}} \tag{5}$$

b) Impetus

Impetus relates to factors: velocity of the player, linearity and the object sparseness in a room. We define the fitness function $f_{Impetus}$ (calculated by using Equation 16) to reward the impetus by the weighted sum of fitness f_V, f_{Li}, f_{ASpr}, and f_{TrSpr} as shown in Equation 6 to 9. Where f_V rewards velocity or quickness of player to clear the room (face the enemies and the obstacle, and move through the player path to the exit), f_{Li} rewards linearity of the room, f_{ASpr} rewards area sparseness, f_{TrSpr} rewards treasure sparseness. Sp_{Player} is the speed of the player, n_E is the number of enemies, n_{Obs} is the number of obstacles, n_{UT} is the number of unique tiles in player paths, n_C is the number of connections, n_{PT} is the number of passable tiles, DTr_i is the distance of treasure i to the player path, and n_{Tr} is the number of treasure.

$$f_V = \frac{Sp_{Player}}{n_E + n_{Obs} + n_{UT}} \tag{6}$$

$$f_{Li} = \frac{1}{n_C} \tag{7}$$

$$f_{ASpr} = \frac{n_{PT} - (n_E + n_{Obs} + n_{UT})}{n_{PT}} \tag{8}$$

$$f_{TrSpr} = \sum_{i=1}^{n_{Tr}} DTr_i \tag{9}$$

c) Tempo

Tempo relates to the factor of the intensity and the density of the objects in a room. Intensity is the measurement of power attributed to the number of objects in a player path to hinder the player. We also consider the density of objects, either how close they are to another same type of object or how close their points are to another point. We use the term points to define the tile where the player path and objects path collide. We define the fitness function f_{Tempo} (calculated by using Equation 16), to reward the tempo by the weighted sum of fitness f_I, f_{DAE}, f_{DAObs}, f_{DATr}, and f_{DBO} as shown in Equation 10 to 14, respectively. Where f_I rewards the intensity of the objects (enemy, obstacle, and treasure), f_{DAE} rewards the average distance of enemies to player paths, f_{DAObs} rewards the average distance of obstacles to player paths, f_{DATr} rewards the average distance of treasures to player paths, and f_{DBO} rewards the closeness of the objects (density). n_{PP} is the number of player paths (literally equal to exit connections), n_O is the number of objects (enemy, treasure, obstacle), RDO_{ij} is the ratio of density between objects and the number of objects in a tile. The distance of object used to calculate the object density includes the distance of the object i to object $i+1$ in path j, start tile to the first object (object i) in path j, or the last object (object n_O) in path j to the exit connection in path j.

$$f_I = \frac{n_E + n_{Obs} + n_{Tr}}{n_{UT}} \tag{10}$$

$$f_{DAE} = \frac{1}{n_E} \sum_{i=1}^{n_E} \frac{ArE_i}{|DE_i - ArE_i|} \tag{11}$$

$$f_{DAObs} = \frac{1}{n_{Obs}} \sum_{i=1}^{n_{Obs}} \frac{1}{Dobs_i} \tag{12}$$

$$f_{DATr} = \frac{1}{n_{Tr}} \sum_{i=1}^{n_{Tr}} DTr_i \tag{13}$$

$$f_{DBO} = \frac{n_{PP} + n_O}{\sum_{n_{PP}}^{j=0} \sum_{n_O}^{i=0} RDO_{ij}} \tag{14}$$

Rewards of all fitness are normalized using Equation 15, where $f_{factor_{i_{min}}}$ and $f_{factor_{i_{max}}}$ are taken from the global constraints set by the game designers to make each reward between 0 and 1 in light of their own preferences. $f_{factor_{i_{min}}}$ and $f_{factor_{i_{max}}}$ are determined through simulation of computing each combination of parameter values from the related equations. Where f_{factor_i} is the un-normalized value of factor i, $f_{factor_{i_{min}}}$ is the minimum value allowed for factor i, and $f_{factor_{i_{max}}}$ is the maximum value allowed for factor i. While $f_{\widehat{factor_{i_j}}}$ is the normalized value of factor j in pacing aspect i, $\omega_{factor_{i_j}}$ is the weight of function of factor j in pacing aspect i, $f_{Aspects_i}$ is the pacing aspect i (threat, impetus, or tempo), and n_{factor_i} is the number of factors for pacing aspect i.

$$f_{\widehat{factor_i}} = \frac{f_{factor_i} - f_{factor_{i_{min}}}}{f_{factor_{i_{max}}} - f_{factor_{i_{min}}}} \tag{15}$$

$$f_{Aspects_i} = \sum_{i=1}^{n_{factor_i}} \frac{\omega_{factor_{i_j}} \cdot f_{\widehat{factor_{i_j}}}}{\omega_{factor_{i_j}}} \tag{16}$$

B. Pacing-based Dungeon Generator (PBDG)

We implement our system as a Unity game engine adds-on using C# programming language in Unity 2017.0.1 version. Our system takes a dungeon level as an input, either manually or procedurally designed by a generator. First, the designer may observe the pacing patterns of all rooms. The pacing patterns are visualized by the graphs of threat, impetus, and tempo curves, respectively. Secondly, the designer can set the intended pacing patterns and the constraints to generate new rooms with new pacing patterns as candidate levels using the genetic algorithm. The designer can then observe and choose one of the level candidates. Finally, the candidate level the designer prefers most will be used to replace the original level. Figure 2 shows the framework of our proposed dungeon generator.

1) Constraints

Constraints are the schemes or rules to drive the process of level generation by giving the limitation to the system. There are two types of constraints: global constraints and local constraints. The local constraints are used to limit the generation of a room, e.g., the maximum number of enemies allowed in each room. The use of global constraints occurs only once in the whole level, while the local constraints might occur in each room.

2) Level Candidates

Level candidates are several levels that are extremely close to the designer's target of pacing patterns. For example, two level candidates might have different placement of game objects while keeping the pacing patterns extremely similar. Thus, it will benefit the game by providing the variant of game levels.

C. Genetic Algorithm

We use a straightforward genetic algorithm (GA) as our initial works; hence, the performance comparisons from different techniques are not yet presented. This type of algorithm can naturally find optimal solutions with high variations of products [23]. Although, it is also very dependent on the meaningful fitness function to separate good and bad level candidates and stuck in a local optima. We address this issue by giving the designer full control to configure the objects generated in a level, also called local and global constraints in this paper. Our system's genetic algorithm is taking a level, which is a list of rooms, the pacing patterns target, and the constraints as inputs; see Figure 2. The system will make a chromosome from a sample room. We use the term chromosome to represent each individual/ solution of the problem. Each chromosome contains a list of genes, where each gene has an ID (identity), position, and type. The ID is used as an identity, and the position is used to store the position information of a sample tile that is taken from a sample room. The type of the gene is taken from the type of tile in the sample room. There are six types of genes, the same as tile types.

The genetic algorithm will run for each room. The number of genetic runs in each room corresponds to the intended number of level candidates. The first process is the initialization which is making a chromosome from a sample room. The first chromosome is cloned as many as the size of the population and then randomly mutated. The next process is selection, which is choosing the best chromosome from the population and then mating it with the other chromosomes (all chromosomes in the population) by a crossover in the further process to get better successor chromosomes. Each chromosome is evaluated using Equation 17, where $f_{\widehat{Fitness}}$ is the final fitness score, ω_i is the weight of pacing aspect i, and $f_{Fitness_i}$ is the fitness value of pacing aspect i. $f_{Fitness_i}$ is calculated using Equation 18, where T_i is the target value of pacing aspect i, and P_i is the current value of pacing aspect i. The selected chromosome is the first-ranked chromosome chosen using rank-based selection [24].

$$f_{\widehat{Fitness}} = \sum_i \frac{\omega_i \times f_{Fitness_i}}{\omega_i},\ i \in \{Threat, Impetus, Tempo\} \tag{17}$$

$$f_{Fitness_i} = 1 - |T_i - P_i| \tag{18}$$

For each chromosome in the population, the evolution is performed with a certain probability chance. Our evolution process is divided into two phases: mutation and crossover. Given a random value for each chromosome, if the value is less than the mutation rate value, the mutation is then performed. In mutation, each gene has a 5% chance of changing its type into another, and/or a 5% chance of swapping with the neighbour. Wall and null types of genes are not allowed to change to another type or swap with their neighbours. Given a random value for each chromosome, if the value is less than the crossover rate value, the crossover is then performed. Each chromosome has a 50% chance to

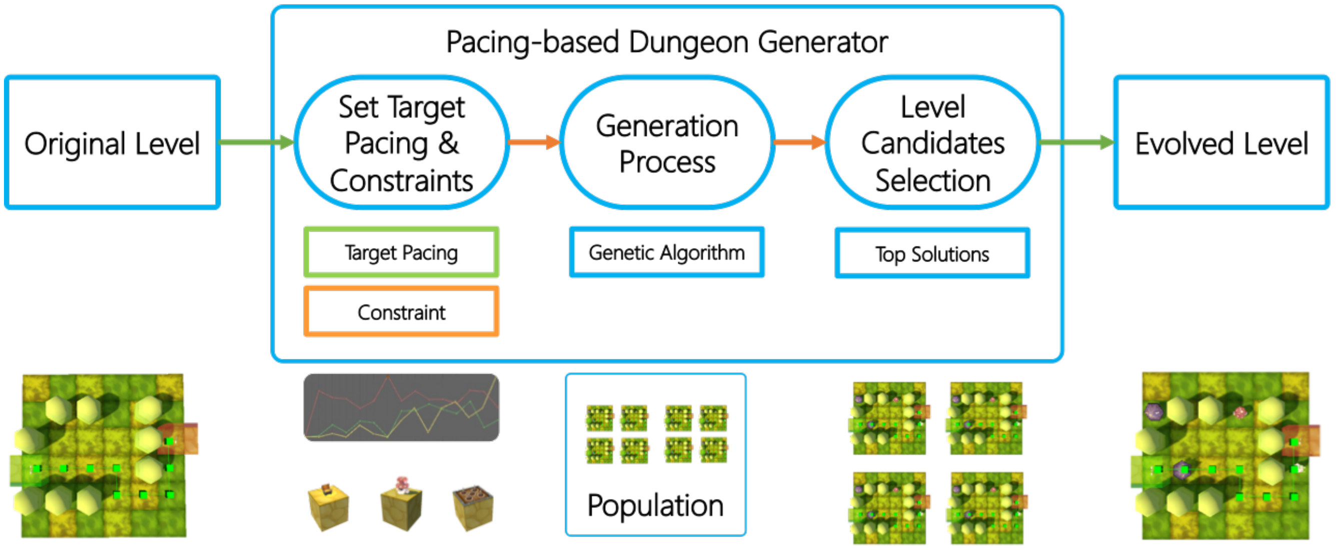


Figure 2. An overview of proposed Pacing-based Dungeon Generator.

copy the first or the rest half of the genes from the selected chromosome from the previous process, and another 50% chance to copy it vertically or horizontally since we use a square room in this study (same width and length).

After the chromosomes evolved, we eliminated 50% of the less-fit chromosomes and replaced them with the re-evolved top 50% fit chromosomes. There is a penalty scheme to reduce the score of a non-fit chromosome (the chromosome which cannot produce the room with intended constraints). In this study, we reduce the current fitness score of a chromosome by 0.1 for each unfulfilled target of pacing aspect and constraint. Then, we get the population for the next generation by reiterating the second phase, which is the selection, and continue with the evolution and evaluation phases again. Our genetic process will stop when the process has already reached the number of generations, or the fitness score is higher than the expected value.

4. Experimental Results and Analysis

The analysis of experimental results and user study is shown in this section. The first part shows the level's precision to match the designer's pacing targets. The factors that influence the results are also presented. Next, the actual player experience in playing the generated levels in the experiment is shown. Finally, we observe and analyze the effect of pacing patterns towards the actual player experiences.

A. Experiments

All of the experiments and user studies are performed in Windows 10 Pro 64-bit OS with Intel Core i5-6400 Processor, 27.4GHz, and 11GB of available RAM. The 3D game assets are bought at Unity Asset Store made by Suriyun. In the experiments to generate the levels, we pre-produce the level spaces using the concept of mission and space from [6]. Therefore, we are focusing more on the producing the expected pacing patterns and the object within levels.

We perform level experiments labelled with A to F, refer to Table III. The purpose of experiments A, B, and C is to observe whether the system can provide various pacing patterns using a level space. Upon the experimental result, we obtain the level average error rates of experiments A, B, and C as 1.16%, 6.78%, and 0.79%, respectively. We calculate E_i, the error rate of pacing aspect i in Equation 19, where T_i is the expected/ target value of pacing aspect i, while R_i is the experimental result value on pacing aspect i.

$$E_i = \left| \frac{R_i - T_i}{T_i} \right| \times 100 \tag{19}$$

In experiments D, E, and F, we use different level spaces but similar pacing pattern targets to show if different level spaces can produce similar pacing patterns. The impetus of the second, third, and fourth rooms in experiment D are not close to the target. That is because the small room size and the number of exit connections of these rooms make our system hard to converge to the target. The average error rate in D is 18.55%, although the threat and tempo are close to the target. The average error of this experiment is 16.23%. Experiment F has the best result with a 7.11% of error rate. In experiments D, E, and F, our system shows the best result in the patterns of tempo and threat. It happens because the impetus is tightly affected by the characteristics of the room space, i.e., room size, number of exit connections, and number of tiles on the player path, which would not change at all in the generation process.

Using the same settings of experiment B, experiments B1, B2, and B3 aim to check whether our system can

TABLE III. Rooms' errors comparison across the levels.

Experiment Label	Pacing Aspect	Error according to room (%)								
		Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Average
A	Threat	0	0	0	0	0.855	0	26.829	0	3.46
	Impetus	0	0	0	0	0	0	0	0	0
	Tempo	0	0	0	0	0.254	0	0	0	0.032
	Average	0	0	0	0	0.37	0	8.943	0	**1.164**
B	Threat	15.385	17.375	0	12.613	4	0	1.412	70.455	15.155
	Impetus	0	0	0	0	0	0	1.803	0	0.225
	Tempo	9.766	0.625	5.672	0	2.024	0	16.110	5.676	4.984
	Average	8.383	5.999	1.891	4.204	2.008	0	6.442	25.377	**6.778**
C	Threat	0	3.03	0	2.034	0	12.121	0	0	2.148
	Impetus	0	0	0	0	0	0	0	0	0
	Tempo	0	0.092	0	1.702	0	0	0.146	0	0.242
	Average	0	1.041	0	1.245	0	4.04	0.049	0	**0.797**
D	Threat	89.423	11.525	20	28.959	8.969	1.836	5.182	62.5	28.549
	Impetus	3.344	135.75	35.349	0.152	0.6	0.35	17.615	0.91	24.259
	Tempo	4.136	1.757	7.812	0.087	2.513	0.682	0.904	0.904	2.851
	Average	32.301	49.678	21.054	9.733	4.027	0.956	21.438	21.438	**18.553***
E	Threat	15.385	54.908	1.531	1.357	0	0.847	0.238	131.250	25.690
	Impetus	24.396	2.254	10.565	22.369	6.994	0.413	19.413	1.386	10.974
	Tempo	6.408	2.434	60.244	0.004	10.847	0.089	10.573	5.774	12.047
	Average	15.396	19.865	24.113	7.910	5.947	0.450	10.075	46.137	**16.237***
F	Threat	13.462	0	0.714	0	0.448	2.401	1.31	131.25	18.698
	Impetus	0	0	0	4.009	1.178	0.474	1.353	4.237	1.406
	Tempo	0	0	1.46	0.105	6.328	0.195	0.36	1.479	1.241
	Average	4.487	0	0.725	1.371	2.652	1.023	1.008	45.655	**7.115**
B1	Threat	0	31.081	68.919	12.613	15.2	0	38.983	32.576	24.921
	Impetus	0	0	19.456	0	0	0	1.803	0	2.657
	Tempo	0	0.156	3.267	0.829	0.99	0	1.0	0.280	0.815
	Average	0	10.412	30.547	4.480	5.397	0	13.929	10.952	**9.465**
B2	Threat	0	33.333	0	0.601	2.4	43.182	3.39	127.273	26.272
	Impetus	0	0	0	0	0	0.879	0	0	0.11
	Tempo	0	0.092	0	4.903	1.003	7.12	2.614	0.003	1.967
	Average	0	11.142	0	1.835	1.134	17.06	2.001	42.425	**9.450**
B3	Threat	0	36.293	70.27	4.204	0.8	0	18.644	0	16.276
	Impetus	0	0	15.063	0.425	0.938	0	1.785	0	2.276
	Tempo	0	0.625	1.543	0.218	9.978	0	0.188	2.311	1.858
	Average	0	12.306	28.959	1.616	3.905	0	6.872	0.77	**6.804**
B4	Threat	0	0.644	1.351	1.802	0.8	0	3.672	31.818	5.011
	Impetus	0	0	9.728	4.978	0	0	1.803	0.271	2.098
	Tempo	0	0.313	0.817	1.116	2.024	0	16.37	2.479	2.89
	Average	0	0.319	3.965	2.632	0.941	0	7.282	11.523	**3.333**

produce various level alternatives (level candidates) given a level space and pacing patterns. Experiments B1, B2, and B3 have 9.46%, 9.44%, and 6.80% errors. The error rate of tempo averaged from experiments B1, B2, and B3 is 1.54%, the error rate of impetus is 1.68%, and the average error rate of threat is 22.5%. Due to the extremely low threat value, the system is difficult to converge on that aspect. Reducing the function weight of impetus and tempo, as described in experiment B4, results in a lower error rate of threat which is 5.01%, and the error rate of impetus and tempo, 2.09% and 2.88%, respectively, making the average error of B4 3.32%. This error rate is the lowest among B1, B2, and B3. The highest averaged pacing aspect error is in experiments D (18.55%) and E (16.23%) when our system hard to give a better result due to the effect of room spaces towards impetus where it cannot change at the moment. The lowest error is the result of experiment C, with only 0.79% of error.

Obviously, the size of the rooms affects the generation process. Currently, our system is intended to help the designer generate intended pacing patterns for the player before the gameplay. We are not focusing on the room space modifications.

B. User Studies

In order to analyse the impact of pacing aspects towards actual player experience, we conduct user studies and then analyse the data obtained through the questionnaire and actual recorded gameplay data. We create a simple cute-themed dungeon game that relies on player actions such as killing the monsters, finding the treasures, and surviving until the finish point. The subjects of these user studies are eight computer science students in GameLab, NTUST (National Taiwan University of Science and Technology). These user studies are performed on the same machine we use for experiments, and the players control the game using a wired X-Box One Controller.

In the user study, we let the players play seven different levels, which are the results of our previous level experiments. First, we use levels A, B, and C to observe if the levels can produce different actual gameplay experiences as the result of different pacing patterns implemented in the same level space. Next, using levels D, E, and F, we observe the player experience as we implement similar pacing patterns to different level spaces. Last, we compare the gameplay data from levels B and B4 to understand the effect of playing the level alternatives (level candidate) while using similar pacing patterns and level spaces. There are three questions for each room in a level that the players need to answer. Q1 asks whether the room is dangerous (because of the enemies and obstacles), Q2 asks if the room makes the player can or want to keep moving (regarding less danger and more treasures found), and Q3 asks if the players do many actions in the room (including move, jump, attack, and ravage actions).

We show the summary of the effect of pacing differences, pacing similarities, and level candidates' variants on the actual gameplay experiences, in Table IV and Table V. The symbol '$\surd$' on the tables indicates the room strongly matches the expectation (the target pacing or intended gameplay experience), the symbol '$\times$' indicates the room is unmatched to the expectation, while '$\surd$*' indicates the room slightly matches the expectation. Therefore, in the pacing differences section, '$\surd$' means the rooms between the levels are dissimilar, while the '$\times$' means the rooms between the levels are similar. Likewise, in the pacing similarities section, '$\surd$' means the rooms between the levels are similar, while the '$\times$' means the rooms between the levels are dissimilar.

From the recorded gameplay data, it is difficult to observe the effect of using different pacing patterns on a level space (levels A, B and C), especially for the factors such as the number of total actions by player, gameplay time, and total damage. In contrast, it is easier to observe through the number of objects (treasure, obstacle, and enemy) and the number of specific actions. There are 5 out of 8 rooms that show the expected patterns resulting from pacing similarity observation (levels D, E, and F). Moreover, 6 out of 8 show similarities in the gameplay experience for the level candidate's variants (levels B and B4). Table IV shows the summary of players' recorded actions. In summary, experiments of levels A, B, and C show the closest result to our expectation, as well as B and B4.

As inferred from the questionnaire data, the pacing difference's effect is easier to be observed, where the pacing patterns of levels A, B, and C result in dissimilar players' responses. For the observation of pacing similarity using different level spaces, 4 to 5 out of 8 rooms show the similarities. Furthermore, the players' responses regarding the similarities of level candidates' variants show that 7 out of 8 show similar players' feedback. See Table V for the summary of players' feedback. We provide the detailed observation on the pacing differences, pacing similarities, and variants of level candidates on the following subsections.

1) Pacing Differences

Figure 3 shows the comparison of the original and resulting pacing patterns from levels A to B4. Based on the recorded gameplay data in Figure 4(a), levels A, B, and C, the most different playing times are shown in rooms 2, 4, 5, 6, and 8, while in rooms 1, 3, 7 are not showing significant difference. For the number of total actions by the players, level C is so different compared to levels A and B. Rooms 2, 5, 6, 7, and 8 are quite different as expected, but rooms 1, 3, and 4 show insignificant dissimilarities regarding the number of actions. Figure 6(a) confirms that level C is so different to level A and B for the specific actions by the players. However, as shown in Figure 5(a), the number of treasures, obstacles, and enemies in those levels are much different. Note that the more dissimilar the pacing patterns, the more diverse the actual gameplay experiences. Similar to the recorded gameplay data, the players' responses shown in Figure 7(a) confirm that the same level spaces can produce different pacing patterns as the designer intended.

2) Pacing Similarities

According to Figure 4(b), the number of actions in levels D, E, and F are quite different in rooms 2, 4 and 7 due to pacing values differences shown in Figure 3. Rooms 1 and 3 have slight differences, while the rest of the rooms show quite similar patterns. As shown in Figure 6(b), the jump and ravage actions in those levels depict the similarities in most of the room. However, move and attack actions in some rooms from level E are considerably different from the other levels, especially for rooms 2, 4, and 7 due to the high error rate for room 2 in level D. Therefore the resulting damage is not as expected.

The clear dissimilarities of the time to complete the objective in the rooms are shown in room 4 due to the room's space differences. However, no significant difference is observed at levels D and F. We observe that room 7 from all levels produces the highest damage among the rooms on the same level, meaning the patterns of the room that give the high damage to the player occur between these levels,

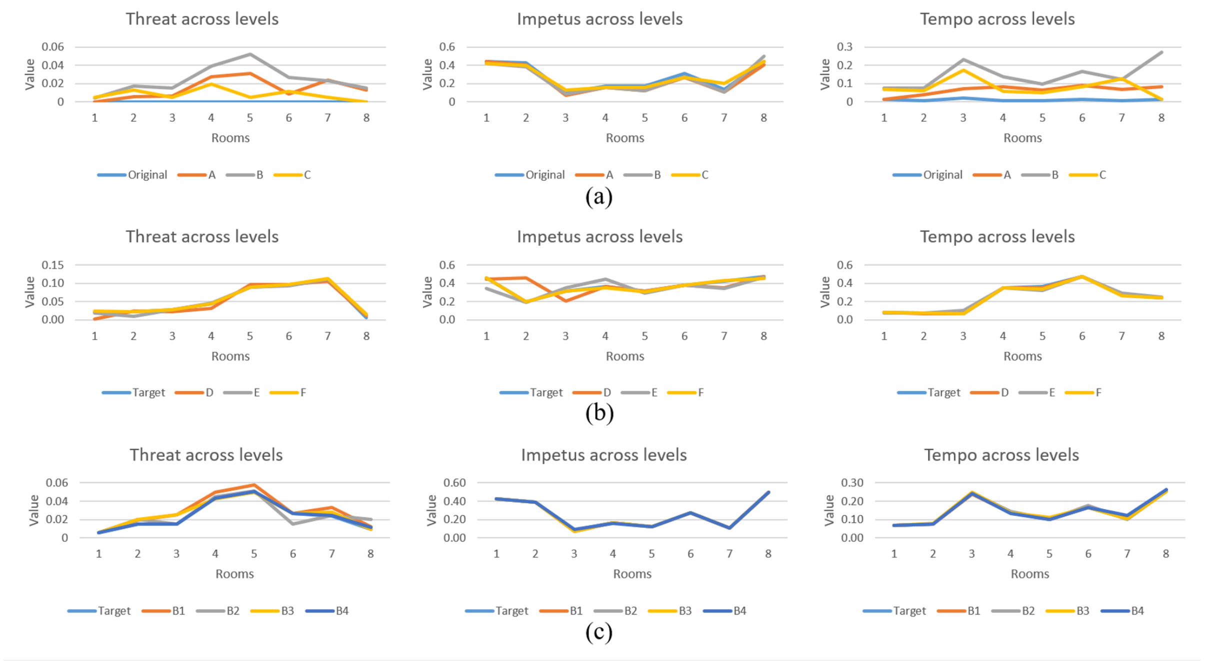


Figure 3. The comparison of the original and resulting pacing patterns from level experiments (a) A, B, and C, (b) D, E, and F, and (c) B1, B2, B3, and B4 plotted per pacing aspect.

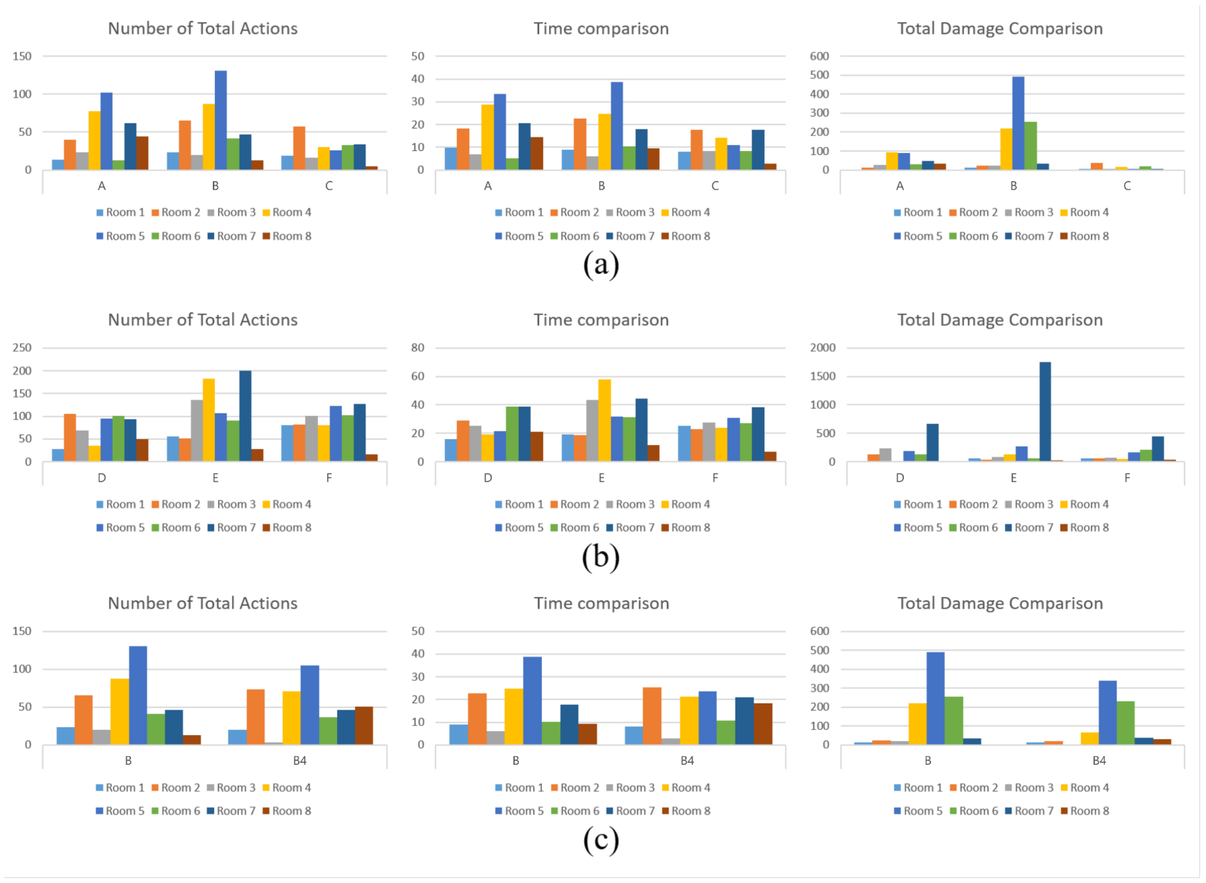


Figure 4. The comparison of total actions, total time spent, and total damage on (a) the experiments A, B, and C, (b) the experiments D, E, and F, (c) the experiments B and B4.

TABLE IV. The recorded gameplay experience expectation regarding to number of total actions, time spent, and total damage on pacing difference, pacing simiarities, and level candidates' variants. Note that the number of objects (treasure, obstacle, and enemy) and the specific type of actions are not presented.

Gameplay Variable	Expectation on Pacing Differences (level A, B, C)								
	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Total
Total Actions	×	√	×	√	√	√	×	√	5
Gameplay Time	×	√	×	×	√	√	√	√	5
Total Damage	√	√	√	√	√	√	√	√	8
Total	1	3	1	2	3	3	2	3	
Gameplay Variable	**Expectation on Pacing Similarities (level D, E, F)**								
	Room 1	**Room 2**	**Room 3**	**Room 4**	**Room 5**	**Room 6**	**Room 7**	**Room 8**	**Total**
Total Actions	√*	×	√*	×	√	√	×	√	5
Gameplay Time	√	√*	√	×	√	√	√	√	7
Total Damage	×	√	×	×	√	√	√	√	5
Total	2	2	2	0	3	3	2	3	
Gameplay Variable	**Expectation on Level Candidates' Variants (level B, B4)**								
	Room 1	**Room 2**	**Room 3**	**Room 4**	**Room 5**	**Room 6**	**Room 7**	**Room 8**	**Total**
Total Actions	√	√	√*	√	√	√	√	×	7
Gameplay Time	√	√	√	√	×	√	√	×	6
Total Damage	√	√	√	√	√	√	√	×	7
Total	3	3	3	3	2	2	3	0	

TABLE V. The players' feedback expectation regarding to number of total actions, time spent, and total damage on pacing difference, pacing simiarities, and level candidates' variants. Note that the number of objects (treasure, obstacle, and enemy) and the specific type of actions are not presented.

Gameplay Variable	Expectation on Pacing Differences (level A, B, C)								
	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Total
Q1: Room is dangerous	√	√	√	√	√	√	√	√	8
Q2: Want to keep moving	√	√	√	√	√	√	√	√	8
Q3: Do many actions	√	√	√	√	√	√	√	√	8
Total	3	3	3	3	3	3	3	3	
Gameplay Variable	**Expectation on Pacing Similarities (level D, E, F)**								
	Room 1	**Room 2**	**Room 3**	**Room 4**	**Room 5**	**Room 6**	**Room 7**	**Room 8**	**Total**
Q1: Room is dangerous	×	×	√	×	√	√	√*	√	5
Q2: Want to keep moving	×	×	√	×	√	√	√*	√	5
Q3: Do many actions	√	√	√	√	√	√	√	√	8
Total	1	1	3	1	3	3	3	3	
Gameplay Variable	**Expectation on Level Candidates' Variants (level B, B4)**								
	Room 1	**Room 2**	**Room 3**	**Room 4**	**Room 5**	**Room 6**	**Room 7**	**Room 8**	**Total**
Q1: Room is dangerous	√	√	√	×	√	√	√	√	7
Q2: Want to keep moving	√	√	×	×	√	√	√	√	6
Q3: Do many actions	√	√	√	×	√	√	√	√	7
Total	3	3	2	0	3	3	3	3	

although their level spaces (the number of connections, tiles in player's path, and room size) are different. All rooms present similar amounts of obstacles. Rooms 1, 2, 4 and 7 show different numbers of treasures as the values of impetus between these levels are not really close to each other, possibly due to the weight of impetus is too small. Rooms 1, 3, and 4 do not show quite similar patterns. Although the pacing values are similar, the magnitude of gameplay data might still be different (as shown in the total damage comparison in Figure 4). The reasons affecting those results are the player preferences while playing the game, the object's position, the error rate of the patterns, and the closeness of the pacing patterns. Besides, we can only offer the expected gameplay but we can never give the perfectly matched-to-intended gameplay experiences.

Rooms 3, 5, 6, and 8 show quite similar patterns of recorded gameplay experience, while rooms 1, 2, and 4 present dissimilar results to the expected target due to the error rate of threat and impetus for Q1 and Q2, as shown in Figure 8. The Q3 show the most similar result since Q3 is much related to the tempo patterns where the error rates of tempo among the levels are low. It implies that the different level spaces might implement the similar pacing patterns set up by the designer, although the result is still highly affected by some geometrical factors such as room

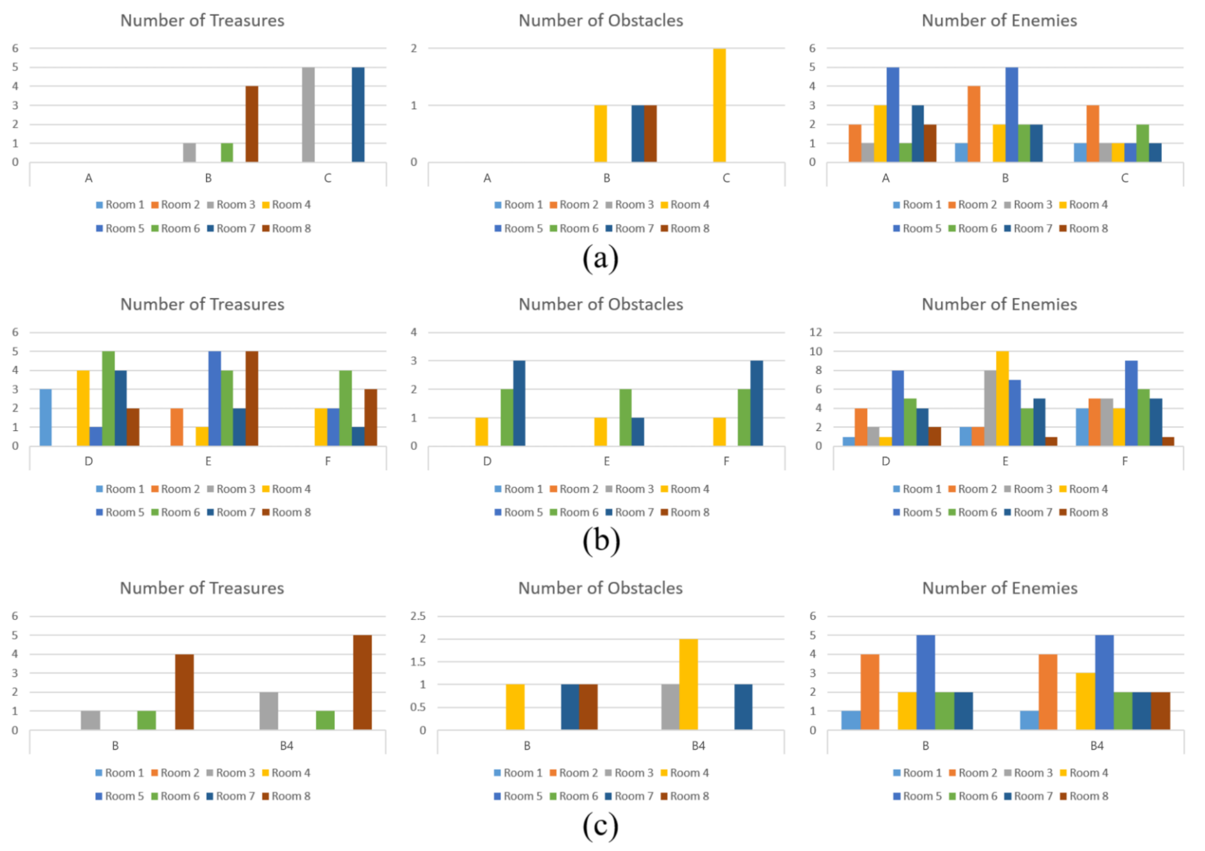


Figure 5. The comparison of the number of treasures, the number of obstacles, and the number of enemies on (a) the experiments A, B, and C, (b) the experiments D, E, and F, (c) the experiments B and B4.

size. For example, the rooms with the same order (number) from different levels which the players play might result in different numbers of movement actions due to the room size difference, although the pacing patterns for levels D, E, and F are considerably similar.

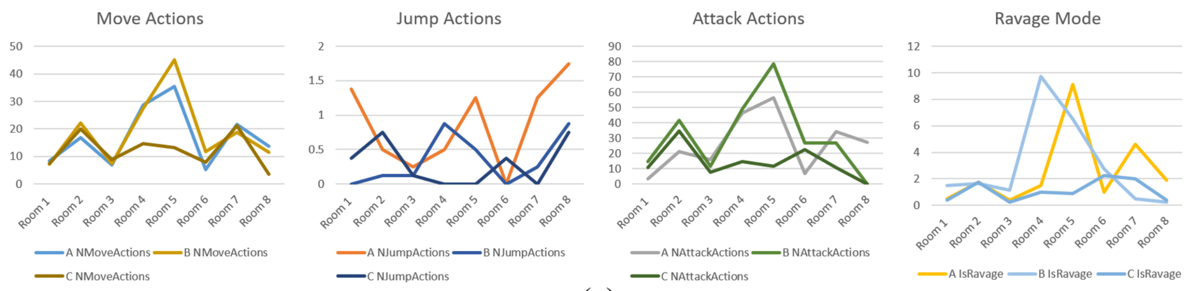
3) *Variants of Level Candidates*

Levels B and B4 show extremely similar pacing patterns since they are built on the exact inputs. Therefore, between those two levels, the total actions, the gameplay time, the total damage is taken by the player, and the number of objects that the players face are quite similar, as shown in Figure 4(c), Figure 5(c), and Figure 6(c). There is a difference between the number of objects on those levels because, in our experiments, we set the allowed number of obstacles must be between 0 to 2. Therefore, the system might produce a different number of obstacles while still satisfying the constraints. Other than that, the differences in the number of total actions, the gameplay time, and the total damage occur because every player might have their own strategy to defeat the exact same enemy in the same position and at considerably the same time. Hence, the result will likely never be perfectly the same.

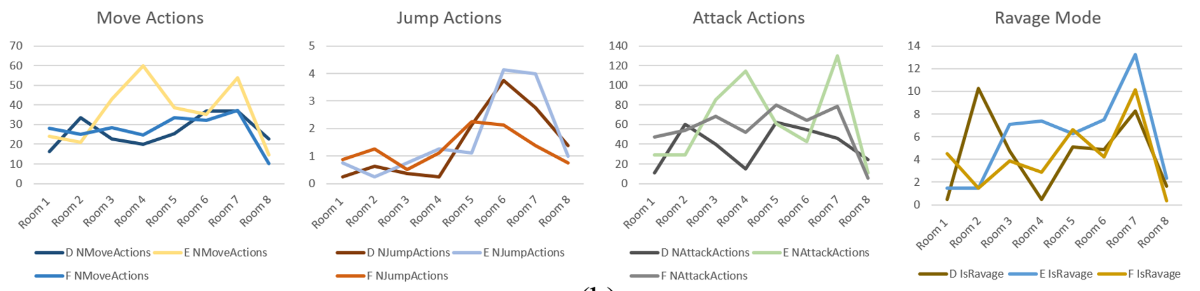
For the answer to Q1, the only different result is shown in room 4, where all of the players agree that room 4 in level B is dangerous while room 4 in level B4 is not really dangerous. However, Figure 9(b) shows that two players agree that the room is not dangerous, while the other players choose to agree and be neutral (due to their unsure feeling). For Q2, most players disagree that rooms 3 and 4 in level B do not make them keep moving, whereas all players agree that room 3 in level B4 keeps them moving, and half of the players want to keep moving in room 4. For Q3, room 4 shows the different responses of players. All players agree if they did many actions in room 4, whereas only two players disagree that they did many actions in room 4 of level B4. It implies that providing the same pacing patterns in the same level spaces might offer similar gameplay experiences. However, there is still a little drawback, as shown in room 4. Every time the player plays a level, they always change the way they play because they get bored. Therefore, it is a hard task to get exact similar actual gameplay data in every generated level and play session.

5. Conclusions, Limitations, and Future Work

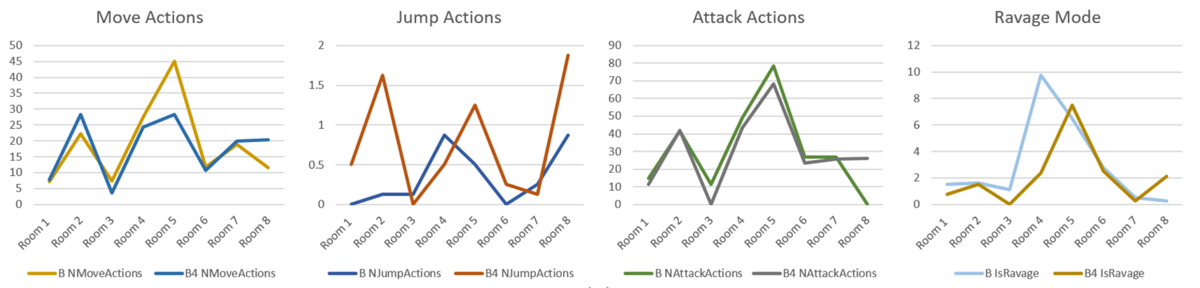
We present our summary, limitations in our system, and the potential further work as a continuation of this study.



(a)



(b)



(c)

Figure 6. The comparison of move, jump, attack, and ravage actions on (a) the experiments A, B, and C, (b) the experiments D, E, and F, (c) the experiments B and B4.

1) *Conclusions*

In this paper, we present game design patterns to control the generation of action-adventure game levels based on the game pacing to provide diverse variants of game levels while keeping the level meaningful to the players. We proposed a system that helps and guarantees the game designer to produce meaningful game pacing of the level according to the designer's intent. Our system, Pacing-based Dungeon Generator, is used to create and analyse a level's pacing patterns and give suggestions to the designer by generating level candidates based on the pacing metrics of threat, impetus, and tempo. Considerably, the system can produce the expected level that the designer wanted. Furthermore, the generated levels can follow the target pacing and the constraints set by the designer. Therefore, the game designer saves lots of development time to create a game level with the expected pacing values since the designer may observe the game pacing of a level in real-time. The pacing metrics to calculate the pacing values and the constraints are also presented.

We propose the game pacing as the rate of events that are affected by complex, cumulative values of the threat, impetus, and tempo for each segment of the game, which will bring the players to better engagements. The designer may set the expected pacing patterns, constraints, and parameters to limit the generation of level candidates. There are two types of constraint: a global constraint used to normalise the pacing aspect values and a local constraint used to limit the generation.

We perform level experiments to show if the system can produce room or level with the expected pacing values and constraints set by the designer. Our global constraints configurations made the system react sensitively to the small changes in pacing values. However, it could be addressed by either changing the global constraint parameter values or reducing the weights of the threat function. In the level experiments, the lowest average error rate is at level C (0.79%), and the highest average error rate is at level D (18.55%), in which the system seems troubled to provide the

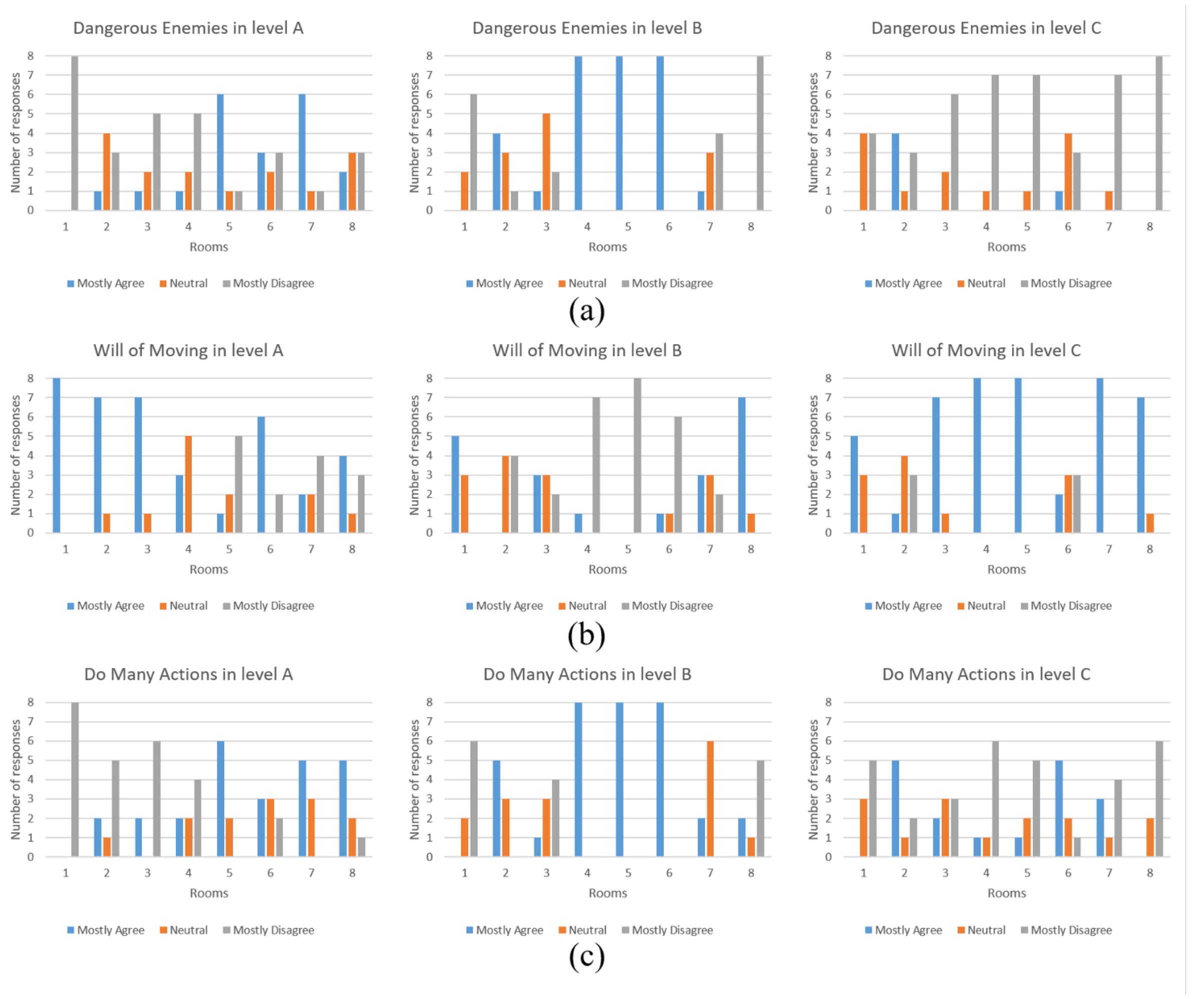


Figure 7. Players' feedback comparisons across level A, B, and C regarding (a) the dangerous enemy (Q1), (b) the will of player to move (Q2), and (c) the quantity of actions by players (Q3).

best result due to the incapability of our current technique to allow a change in room size and the number of connections, resulting in the high error of impetus. Generally, the room with more passable tiles will likely produce the intended pacing. However, if the designer is provided with such information, the designer may get a better result.

The closeness of the result to our expectation is tightly affected by the level's space (e.g., room size, number of exit connections, certain types of tiles), while the sensitivity of the pacing aspect curves is highly affected by the value range in the global and local constraints. We also notice that if two rooms with the same average error rates have different sources of error; one has a high error rate on only one pacing aspect, and the other has similar small error rates among the pacing aspects, the second room has a high tendency to provide a better result as compared to the first room. The resulting room with the error rates shared among all the pacing aspects is better than the room with a high error rate on one of the pacing aspects.

Our system provides game levels with the intended pacing patterns and follows the constraints given by the game designer with their own preferences as the results are measured and presented. In addition, we also let the designers use their own configuration to calculate the value of each pacing aspect. It will give the freedom to the game designer to control the process since we know each game designer might have a different preference.

Finally, in this paper, we focus on the 3D level of a dungeon crawler game, but our concept provided with more flexibility; hence such technical implementation may differ across different game genres. For example, although the concept of game pacing strongly appears in an action game like a dungeon crawler, the concept itself can also be found in every other genre.

2) *Limitations*

Our study offers the initial idea for further studies since there are still only a limited number of research that discuss

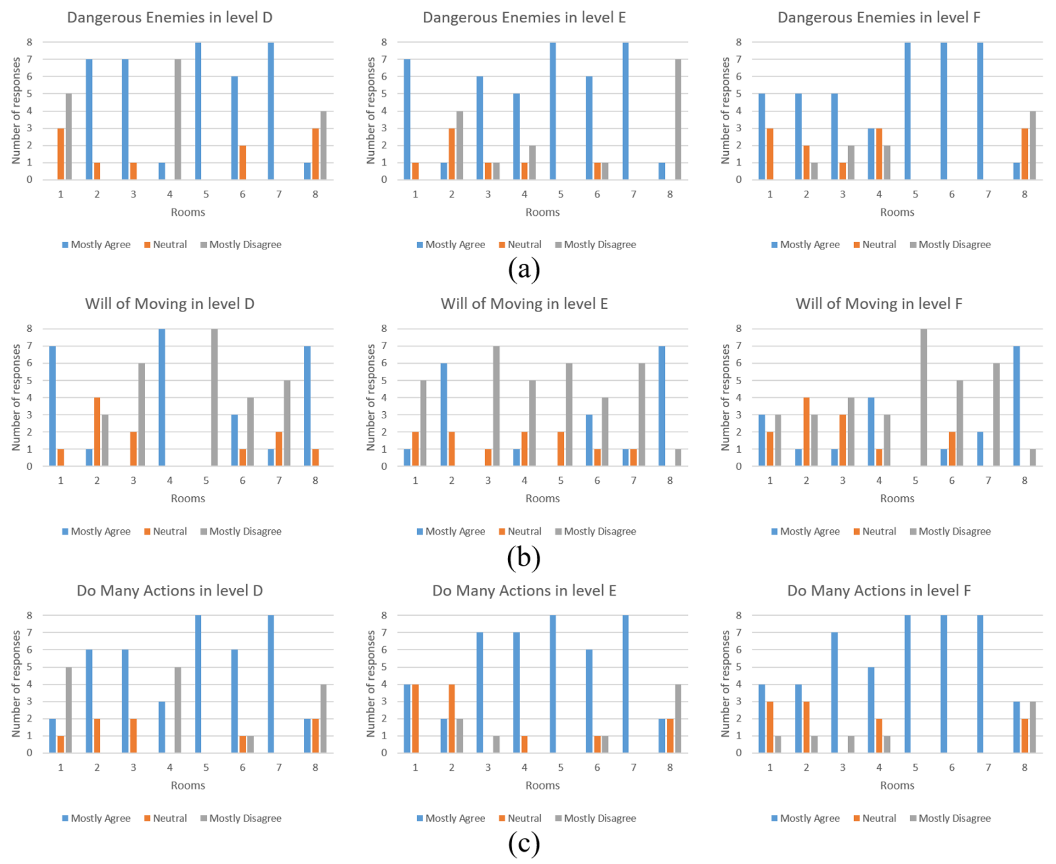


Figure 8. Players' feedback comparisons across level D, E, and F regarding (a) the dangerous enemy (Q1), (b) the will of player to move (Q2), and (c) the quantity of actions by players (Q3).

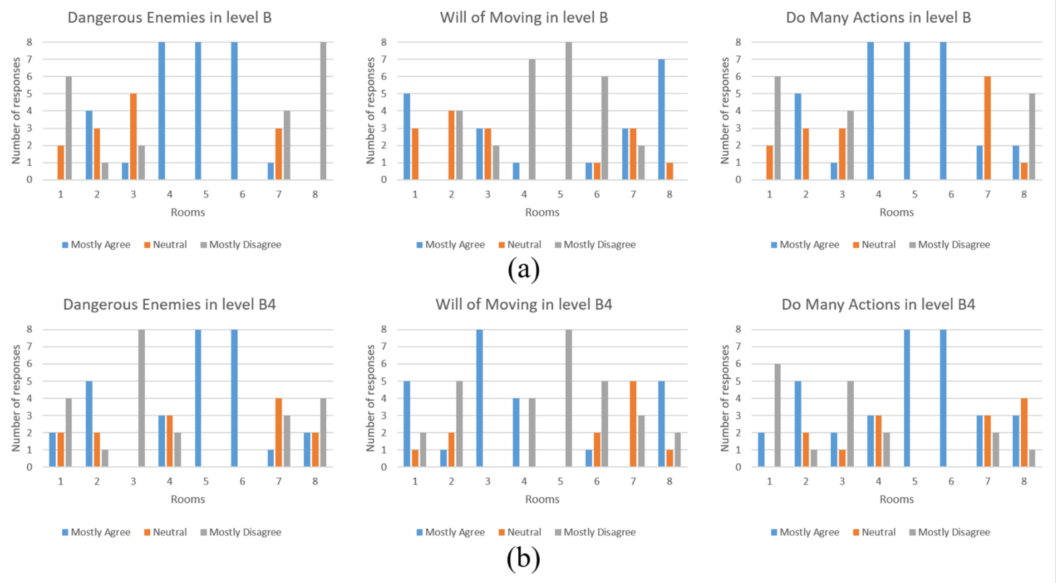


Figure 9. Players' feedback comparisons regarding the dangerous enemy (Q1), the will of player to move (Q2), and the quantity of actions by players (Q3) between level B and B4, as shown in (a) and (b) respectively.

game pacing. The pacing patterns are visualised linearly as the players play the level as a sequence of room spaces to do the actions. In some games which allow non-linear gameplay (room with multiple connections or will be passed several times), the pacing pattern alternatives are provided in parallel order. Meaning the player may finish multiple quests simultaneously or choose which quest is needed to be finished first in any order. We avoid this issue by applying the lock and key mechanism to make the rooms be played linearly from the entrance to the finish point.

3) *Future Work*

This study can be used as a basis for some potential future works with the following two steps as our recommendation. First, we need to address a game level that supports parallel pacing (branched pacing patterns). Secondly, we can implement the pacing patterns on the smaller scale of the game segment, e.g., each point in the pacing pattern represents each tile in the player path. Theoretically, it will give a more accurate experience for the players. We can use artificial intelligence to get which configurations produce the best result, where the AI agent or human players might qualitatively and quantitatively measure the result.

Acknowledgment

I would like to express my gratitude to my advisors, my fellow scholars in NTUST GameLab and Dinus IntSys Gamelab, and everyone involved in this research.

References

[1] Valve Corporation, “Steam,” 2022. [Online]. Available: http://www.store.steampowered.com/

[2] R. Van Der Linden, R. Lopes, and R. Bidarra, “Procedural generation of dungeons,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, 2013.

[3] J. Wesołowski, “Beyond pacing: Games aren’t hollywood,” 2009. [Online]. Available: https://www.gamedeveloper.com/design/beyond-pacing-games-aren-t-hollywood

[4] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg, “Mixed-initiative procedural generation of dungeons using game design patterns,” in *2017 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2017, pp. 25–32.

[5] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.

[6] J. Dormans and S. Leijnen, “Combinatorial and exploratory creativity in procedural content generation,” in *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games*. [Sl]: Society for the Advancement of the Science of Digital Games, 2013, pp. 1–4.

[7] J. Dormans, “Adventures in level design: generating missions and spaces for action adventure games,” in *Proceedings of the 2010 workshop on procedural content generation in games*, 2010, pp. 1–8.

[8] M. Davies, “Examining game pace: How single-player levels tick,” 2009. [Online]. Available: https://www.gamedeveloper.com/pc/feature-examining-game-pace----how-single-player-levels-tick-

[9] A. Canossa and G. Smith, “Towards a procedural evaluation technique: Metrics for level design,” in *The 10th International Conference on the Foundations of Digital Games*. sn, 2015, p. 8.

[10] G. Smith and J. Whitehead, “Analyzing the expressive range of a level generator,” in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 2010, pp. 1–7.

[11] B. Patatas, “The dungeon crawler recipe,” 2013. [Online]. Available: https://www.gamedeveloper.com/design/the-dungeon-crawler-recipe

[12] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha, “Launchpad: A rhythm-based level generator for 2-d platformers,” *IEEE Transactions on computational intelligence and AI in games*, vol. 3, no. 1, pp. 1–16, 2010.

[13] Z.-H. Wang, “Game design goal oriented approach for procedural content generation,” Master’s thesis, National Taiwan University of Science and Technology, 2017.

[14] G. R. Wichman, “A brief history of ’rogue’,” 1997. [Online]. Available: http://www.wichman.org/roguehistory.html

[15] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[16] D.-Y. Gu, “A study on automatically optimizing gameplay based on the expectation of game designer,” Master’s thesis, National Taiwan University of Science and Technology, 2016.

[17] B. M. Viana and S. R. dos Santos, “A survey of procedural dungeon generation,” in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2019, pp. 29–38.

[18] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, “Toward supporting stories with procedurally generated game worlds,” in *2011 IEEE Conference on Computational Intelligence and Games (CIG’11)*. IEEE, 2011, pp. 297–304.

[19] V. Valtchanov and J. A. Brown, “Evolving dungeon crawler levels with relative placement,” in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, 2012, pp. 27–35.

[20] D. Ashlock, C. Lee, and C. McGuinness, “Search-based procedural generation of maze-like levels,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 260–273, 2011.

[21] C. McGuinness and D. Ashlock, “Decomposing the level generation problem with tiles,” in *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, 2011, pp. 849–856.

[22] A. Liapis, G. N. Yannakakis, and J. Togelius, “Sentient sketchbook: computer-assisted game level authoring,” 2013.

[23] A. Gellel and P. Sweetser, “A hybrid approach to procedural generation of roguelike video game levels,” in *International Conference on the Foundations of Digital Games*, 2020, pp. 1–10.

[24] L. D. Whitley *et al.*, *The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.* Citeseer, 1989.

Ardiawan Bagus Harisa is a lecturer at Universitas Dian Nuswantoro, majoring in Computer Science, specifically artificial intelligence and procedural content generation in game development. He earned his M.Sc in Computer Science from National Taiwan University of Science and Technology at 2018. He is also a creative multimedia consultant and creative director at PT. Nagara Karya Karsa.

Wen-Kai Tai is a professor at Department of Computer Science and Information Engineering and a director of GAME Lab at National Taiwan University of Science and Technology. He is an expert in computer graphics, procedural content generation, and development technique. He received his M.Sc and Ph.D in Computer Science from National Chiao Tung University in 1989 and 1995.