Design and Evaluation of a Procedurally Generated Dungeon Game

Arthur R. Pinheiro So[1], **Alinne C. Corrêa Souza**[1], **Lincoln M. Costa**[2], **Rafael G. Mantovani**[3], **Francisco Carlos M. Souz**[1]

[1]Federal University of Technology – Paraná – Dois Vizinhos, PR, Brasil

[2]Federal University of Rio de Janeiro (UFRJ) – Rio de janeiro, RJ – Brasil

[3]Federal University of Technology – Paraná – Apucarana, PR, Brasil

arthurriuiti@alunos.utfpr.edu.br, costa@cos.ufrj.br, rgmantovani@gmail.com,

{alinnesouza, franciscosouza}@utfpr.edu.br

Abstract. *Game development involves creating various content, from artistic tasks to technical activities. Procedural Content Generation (PCG) and meta-heuristics like Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) automate and optimize this process. This paper presents Dungeoncide, a game using GA and PSO for procedural content generation. Dungeoncide features a medieval knight battling monsters in procedural generated dungeons. The primary contributions include the development process and player feedback on the generated content's quality and feasibility. Experiments showed that while generated levels were well-received, they were less challenging and immersive than manual ones. Future work will refine the algorithms to improve balance, aesthetics, and engagement.*

1. Introduction

As players demand for game content rises, the game industry faces the challenge of increasing costs in content production [TOGELIUS et al. 2011]. Procedural Content Generation techniques can help reducing costs providing useful means to create content such as levels, maps, quests, textures and rules with limited or no human intervention [Togelius et al. 2013]. These contents can be pre-generated or created in real-time during gameplay. Games like *Minecraft*, *Terraria*, and *Enter The Gungeon*, for instance, leverage run-time PCG to provide unique experiences with each playthrough. In this context, meta-heuristics, enhance the PCG process by optimizing the generated content.

In general, meta-heuristics like the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) can be employed to find optimal solutions to hard problems. GAs uses the principles of evolution, in which states of the solution are generated by combining two parent states [Russel and Norvig 2013], to search for the best solutions. At the same time, PSO optimizes solutions by simulating swarms of particles, each with its velocity and position relative to other particles in the search space [Kennedy and Eberhart 1995].

In this paper, we introduce a game called *Dungeoncide*, which was developed using GA and PSO for procedural content generation. The algorithms generated the dungeon map, positioned enemies, and controlled enemy behavior. The game draws inspiration from the board game *Zombiecide* [GUILLOTINE GAMES 2012], which is a boardgame set in a zombie apocalypse enviroment where players must gather resources, survive zombies and complete objectives inside a modular board that can be assembled in various ways to introduce new challenges. In *Dungeoncide*, zombies are replaced with monsters, and the protagonist is a medieval knight. The main contributions of this work can be summarized as 1) the development process of the *Dungeoncide* game using PCG to automate the creation of game elements and 2) the evaluation of the generated content's quality and feasibility through feedback from real players.

This paper is structured as follows: Section 2 outlines the studies that served as the foundation for this work. Section 3 details the implementation of the *Dungeoncide* game. Section 4 describes an experiment conducted with real players to evaluate the developed algorithms and analyze the results, and Section 5 analyzes and discusses the results obtained through survey. Finally, Section 6 presents the conclusion of the work and potential future directions.

2. Related Works

Several authors address the topic of PCG in game design. In this section, we explore some of these works and the games created with PCG.

2.1. Maze Generation

The work of Adams and Louis [2017] seeks to produce mazes considered interesting to be played without having the cost of manual production, for this, the author uses GA to evolve the rules of Cellular Automata (CA) to create the mazes automatically.

The generation method uses a CA, the transition rules of these automata are evolved by the GA used. Two approaches called binary and probabilistic for the transition rules were tried, in the binary approach each gene of the GA represents a transition rule for n filled neighborhood cells. In the probabilistic approach, the gene presents a chance for the cell to change state, unlike the binary approach in which the transition is deterministic. Experiments were carried out with three different fitness functions, the first ($F1$) corresponds to the minimum distance to complete the maze, the second ($F2$) evaluates the number of dead ends and the third function corresponds to the sum $F1 + F2$ ($F3$). Algorithms with the probabilistic approach performed better in all fitness functions and were much closer to the maximum possible, the author notes that this is due to the lower sensitivity of the chromosome to the mutation operator.

2.2. Generic Levels Generation

Zafar et al. [2020]'s work is based on the framework created by Khalifa et al. [2016] to generate game levels with maps in grid format, the author's focus is to build this generator generically, focusing on the operation of the algorithm in any grid-style game. The works presented so far focus on the development of a generator and apply the demonstration in a specific game. This work focuses on finding good levels based on generic features present in these games, the aesthetics (referring to the look formed by the architecture) of the level, and the difficulty.

Two algorithms are used to achieve the mentioned objectives, the first algorithm is responsible for initializing the parameters used in the generation of the chromosomes. The second algorithm takes these parameters to construct the levels. The created chromosomes are divided into two different populations: feasible and non-feasible, this division is based on the number of avatars in the level, objects, objectives, and solution size, among others. The population of infeasible chromosomes seeks to reduce the number of invalid chromosomes, while the feasible population seeks to improve the solutions.

The algorithm was tested in several types of games, as the algorithms work generically, and only initial parameters are replaced. Levels were created based on the Zelda games [NINTENDO 1986], Butterflies, Chase, Freeway [DAVID CRANE 1981], and Survive Zombies. The results demonstrate the versatility of the heuristic used, in addition to the work of Khalifa et al. [2016] there was an improvement in the general appearance and difficulty of the generated levels.

3. *Dungeoncide* game

The *Dungeoncide* game was created within the Unity tool [Unity 2021]. The visual elements (sprites) of the game, such as textures used on walls, floors, objects and characters, sound effects, and music, were obtained through the database Open Game Art[1] and itch.io [2] which are of repositories that conglomerate these elements and that can be used for free. It is not necessary to download the game, as it is compiled for the language WebAssembly and hosted using the platform GitHub pages `https://olafmustafar.github.io/dungeoncide-web/`.

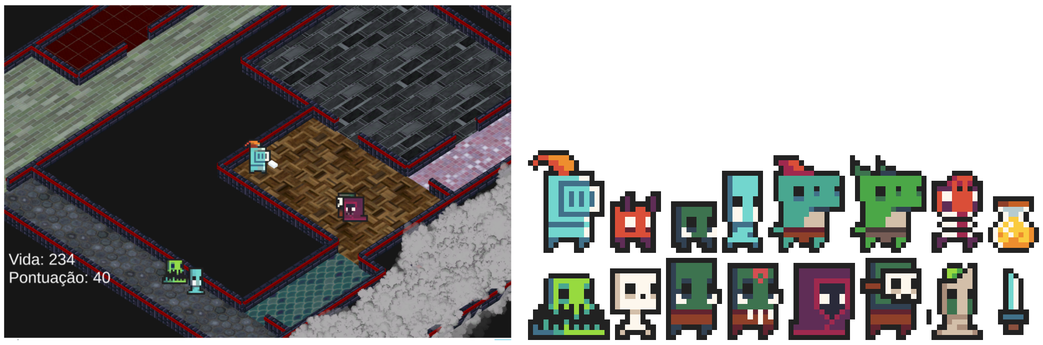
The structure of the game is quite simple: the player can walk using the W, A, S, and D keys or using the arrow keys on the keyboard; the mouse is used for aiming, and the left click shoots the projectiles. There are some elements like music and sound effects and a scoring system to help with player immersion.

As the player walks through the dungeon, he increases the chances of attracting enemies in nearby rooms; enemies will also move toward the player if he is in their line of sight. The player confronts the enemies by shooting projectiles; by defeating enemies, it is possible to acquire a Power Up potion that strengthens the way the player shoots, increasing his bullet amount, spread, fire speed, or size; defeating enemies also increases the player's score. When all enemies are defeated, a message of victory is presented to the player. However, a defeat message will be given if the player runs out of life.

Figure 1(a) shows a snapshot of the game. To enhance the visual aspects of the game, we chose a popular set of sprites known as the Dungeon Tileset for its simplicity and charm. The dungeon enemies consist of various types of monsters, including zombies, undead, demons, and orcs, rather than just zombies. An overview of the sprites used in the game is shown in Figure 1(b).

Figure 2 illustrates the game creation process, in which the steps that involved content generation are marked in blue. We first developed the algorithms for the layout and enemies. Then, we created the game, incorporating the enemy behavior algorithm and handling tasks such as collecting various assets. We used three algorithms to create the game levels. The first algorithm generated the dungeon layout, the second placed the enemies within this layout and defined their attributes and positions, and the third controlled the enemies' behavior during gameplay. The following sections will describe the algorithms used to generate the layout, balancing, and allocation of enemies and their behavior.

[1]https://opengameart.org
[2]https://itch.io



(a) The player, with blue armor in the center, is surrounded by monsters.

(b) Sprites used by *Dungeoncide*

Figure 1. Game Dungeoncode

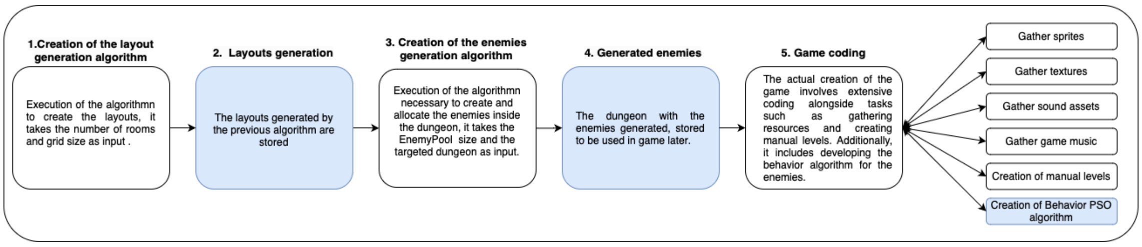

1.Creation of the layout generation algorithm
Execution of the algorithmn to create the layouts, it takes the number of rooms and grid size as input .
2. Layouts generation
The layouts generated by the previous algorithm are stored
3. Creation of the enemies generation algorithm
Execution of the algorithmn necessary to create and allocate the enemies inside the dungeon, it takes the EnemyPool size and the targeted dungeon as input.
4. Generated enemies
The dungeon with the enemies generated, stored to be used in game later.
5. Game coding
The actual creation of the game involves extensive coding alongside tasks such as gathering resources and creating manual levels. Additionally, it includes developing the behavior algorithm for the enemies.
Gather sprites
Gather textures
Gather sound assets
Gather game music
Creation of manual levels
Creation of Behavior PSO algorithm

Figure 2. The process used for *Dungeoncide* creation.

3.1. Dungeon Generation

The GA for layout generation described in Brown et al. [2017] work was initially used to create levels for Hotline Miami, but it can also be adapted for generic dungeon development. The algorithm receives as input the number of rooms (N_{rooms}) to be placed and the dimensions of the grid. Initially, the rooms are positioned randomly and then evolve as the genetic algorithm progresses.

Before the first iteration, the GA randomly creates a list of rooms. It then tries to place them sequentially inside the grid. Initially, some rooms may have invalid locations and fail to be placed. This could be because their position overlaps with an already placed room, or they are in a position that is disconnected from other rooms in the grid. Understanding these reasons for room placement failures helps us grasp the algorithm's decision-making process. When optimizing the dungeon, the GA will attempt to place the most rooms possible and also reduce the number of rooms that are too small. It will try to maintain the dungeons with the largest sizes and will try to prevent the rooms from being too interconnected. Upon completion, the outcome will be a dungeon with maximized fitness, ready for integration into the game. The resulting level, exemplified by the generated dungeon in Figure 3, will be part of the game environment.

3.2. Enemy Generation

After the layout is created, the next step is the position of the enemies within the game, enemies are the essential force that opposes the player from completing the objectives, therefore their positioning and balancing is an essential step to the fun of the game. The enemy generation is done by generating a set of enemies and evolving their placement in the level layout through another GA. The enemies are composed of four attributes: (i)

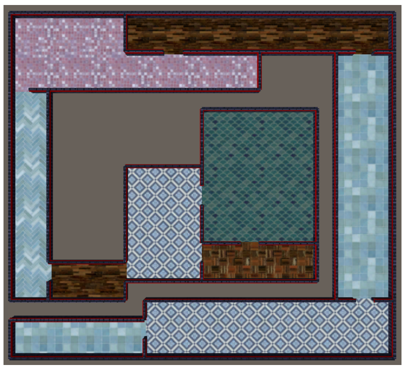


Figure 3. Layout generated by GA.

Health: The amount of damage the enemy has to take until it is defeated; (ii) *Damage*: The amount of it does to the player once it attacks; (iii) *Velocity*: Is how fast it cam move towards the player; and (iv) *Attack Cooldown*: Is how much it waits until it can attack the player again.

As input, the algorithm receives the number of enemies it will try to position, this number is used for the size of a list of enemies the algorithm will randomly create at the start, know as the $EnemyPool$. The GA will then evolve the positions of the enemies inside the dungeon, attempting to place enemies in rooms in a way that the sum of their attributes approaches the room threshold in combination with other enemies in the same room. The GA accumulates a set of enemies in a room, balancing the enemies' attributes. In this way, weak enemies may be paired up with strong enemies in the same room to compensate for their attributes; in the same way, it is possible for a room to have a big number of weak enemies or a low number of strong enemies.

3.3. Enemy Behaviour

A PSO is used to control the behavior of the enemies during the game execution, it works by using the enemies as the particles and the dungeon as the search space, the best solution in this search space is the player's location. In this way, enemies will behave like a swarm to find the player

A few differences exist between a normal PSO algorithm and its implementation within Dungeonside. First, the solution is not static as it normally is. Since the optimized solution is the player location, the PSO must be adapted to consider the moving search space. To this end, two new parameters, $PBestDecau$, and $GBestDecay$, were adopted to gradually reduce the $PBest$ and $GBest$ values over time. This is necessary as the player can get away from the current best solutions, which are the player's previous location. The second change to the original PSO implementation is the introduction of obstacles in the search space, since inside the game, some walls prevent the enemies from moving across. This is adapted, changing the enemy inertial towards the closest path to the player. Third, the inertia weight is prevented from being reduced over time since the search for a solution is constant as the game is running.

With these three adaptations in mind, the PSO tries to optimize two attributes when searching for the solution. The $visionFitness$ is the distance between the particle and the solution (the player location) when there is no obstruction between them. If there is, then its value equates to zero. The second attribute is the $soundFitness$ which is very similar, it consists of the distance to the solution, but when there is a obstacle its value is only diminished instead of negated. The sum of $visionFitness$ and $soundFitness$ composes the final $Fitness$ value. It is expected that with this search heuristic and the three adaptations, this adapted PSO algorithm will offer an innovative and interesting solution in general as an alternative to algorithms used in games.

4. Experimental Study

We conducted an exploratory survey to analyze the players' perceptions about quality and feasibility of generated content of the *Dungeoncide* game. The survey was planned following the process proposed by Kasunic (2005) and Kitchenham e Pfleeger (2008) for effective design of surveys for the software engineering area .

4.1. Research Objectives and Target audience identification

We used the Goal-Question-Metric (GQM) model [Basili and Weiss 1984] to set out the objectives of the experiment that can be summarized as follows:*"Analyse **generated content** for the purpose of **evaluation** with respect to **quality and feasibility** from the point of view of **real players** in the context of **Dungeoncide game**."* For achieving the goal, we seek to investigate the two Research Questions (RQs), presented in Table 1.

Table 1. Research Questions according to Survey Goal

Research Questions	**Description**
RQ_1**:** What is the players' profile?	To answer this RQ, we identify some information about players, such as their age, gender, city, and gaming experience.
RQ_2**:** What are the players' perceptions about the levels generated for the *Dungeoncide* game?	To answer this RQ, we analyze difficulties, immersion, enjoyment, the layout of the level played, balance and distribution of enemies, and behavior of enemies generated through the GA and PSO algorithms and manually.

The target audience of this survey is players of a University. These players are from different origin states. From the target audience defined, the next step consists in selecting a sample. In this survey, we use a random sampling in which individuals of the sampling frame are selected at random [Thompson 2012].

4.2. Survey instrument design and evaluation

Immersion, as defined by Brown et al. (2017) , is characterized by the player's engagement with a digital game, influenced by human, computational, and contextual factors. A questionnaire with 15 questions to measure these factors. The questionnaire is grouped into the following two sections: *(i)* information and experience of players; and *(ii)* players' perception about immersion in the levels played and if they identify the similarity or difference in the levels generated through GA, PSO and manually.

We conducted a pilot study to analyze instrument validity. Six players were chosen based on availability and proximity. They answered questions defined by Hauck et al. (2011). The players' evaluations were positive, with suggestions to (i) reduce the number of questions and (ii) make some questions non-mandatory.

4.3. Data Collection and Analysis

The questionnaire was open for one week, starting on November 10, 2022, and during this time, 51 players responded to the survey. The data collection and analysis was divided into seven steps. In **Step 1**, we created ten levels using algorithms, half manually and half generated. Each set has shared parameters: 10 rooms in a 20x20 grid with a maximum of 20 enemies and an average of two per room. We excluded instances where the algorithm couldn't position all enemies or rooms. In **Step 2**, we present instructions and a welcome message on the game's home screen. On the next screen, we introduce the research context, inform the player about the questionnaire, and ask them to pay attention to the layout of the levels, the enemies, and their behaviors. In **Step 3**, the player begins by answering the first part of the questionnaire to assess their profile. After completing the questions, they are introduced to the game.

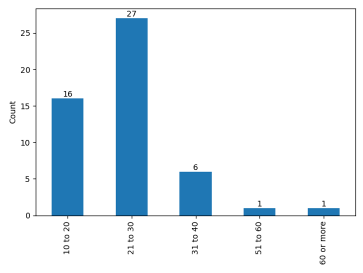
In **Step 4**, the first level shown to the player is randomly selected from the generated or manual level groups. The player must defeat all enemies to proceed. Afterward, the immersion quiz for the level is introduced. This process is repeated for a level from the opposite group. In **Step 5**, we introduce the questionnaire about immersion after each level is played. In **Step 6**, after evaluating the two levels, the player takes a test to identify which level was generated through algorithms. Regardless of the player's answer, we display the correct answer. Finally, we introduce the last part of the questionnaire, focusing on similarity. We store all questionnaire responses in the browser's memory and, upon completion, send them to the Google Sheets API for storage. Finally, in **Step 7**, we carried out two activities beforehand [A. and Pfleeger 2008] to assist the analysis process: (i) validating the data by checking the consistency and completeness of responses, and (ii) partitioning the players' responses into two subgroups: those generated by GA and PSO algorithms and those created manually.

5. Results and Discussion

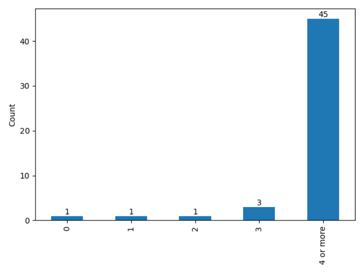
In this section, we present the players' profiles and their perceptions about the levels generated for the *Dungeoncide* game.

5.1. Players' profile (RQ_1)

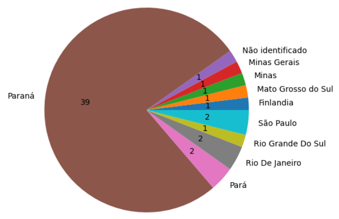
Initially, we aimed to outline the basic profile of the players. Figures 4(a) and 4(b) depict a typical profile of players aged 10 to 20 and 21 to 30. It can be inferred that most players fall within the 18-22 age group, reflecting the demographics of the university groups where the game was promoted. Furthermore, most players have over four years of experience with video games, and the majority are located in Paraná, Brazil. Given the limited diversity in profiles, exploring their relationship with other obtained data is not feasible.



(a) Players age.



(b) Players experience time.

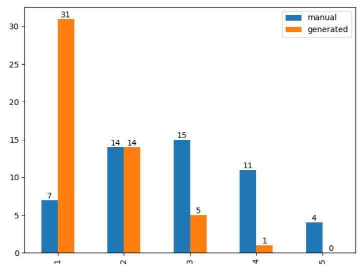


(c) Players origin state.

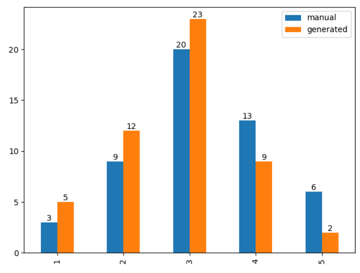
Figure 4. Players profile.

5.2. Players' perceptions about levels generated for the *Dungeoncide* game (RQ_2)

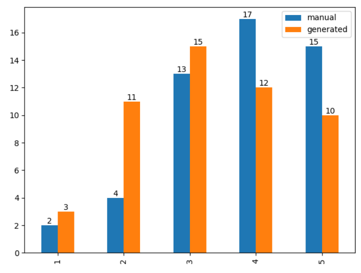
Figure 5 presents the player's perceptions of **difficulty**, **immersion**, and **fun** for levels generated by algorithms and manually. When comparing the difficulty of manual and generated levels, we found that the generated levels tend to be easier (Figure 5(a)). This scenario can be attributed to the limitations of enemy generation and the adapted PSO. The lack of challenge negatively impacts immersion (Figure 5(b)) and fun (Figure 5(c)), resulting in superior performance of manually created levels.



(a) Players difficulty.



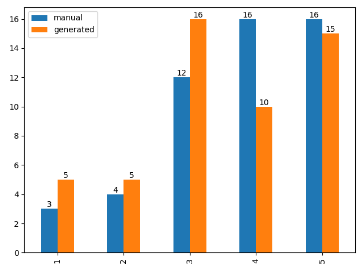
(b) Players immersion.



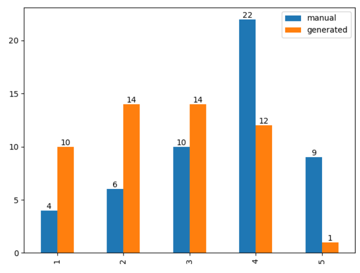
(c) Players fun.

Figure 5. Players' perception of difficulty, immersion, and fun.

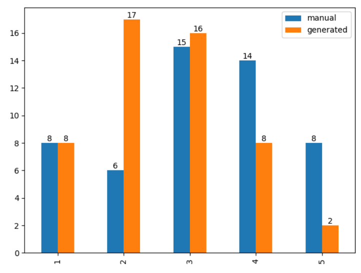
Figure 6 present the **layout**, **behavior**, and **balance scores** further underscore the advantage of manual level creation, consistently yielding higher average ratings. While generated levels receive moderate scores for enemy behavior (Figure 6(b)) and dungeon layout (Figure 6(a)), their scores for enemy balance (Figure 6(c)) are notably low. Regarding difficulty, enemy behavior plays a significant role. While the algorithm used for attribute balancing performed well in expressiveness tests, it fails to consider the impact of each attribute on gameplay. For instance, a "balanced" enemy may possess high damage but minimal movement speed, offering little challenge to the player. The enemies, influenced by the adapted PSO, primarily contribute to the ease of generated levels. Experienced players tend to move more during combat, making it challenging for the PSO to locate the player efficiently, as it must continually search for a solution after the player's position changes.



(a) Layout.



(b) Enemy behavior.

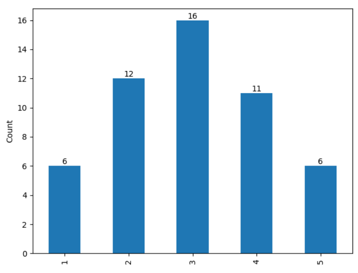


(c) Enemy balance.

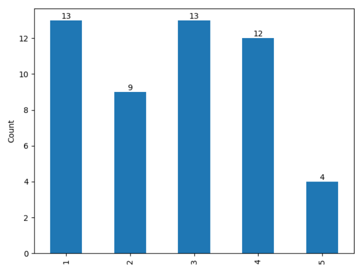
Figure 6. Players' perceptions of layout, enemy behavior, and enemy balance.

Based on player feedback, we observed that shorter corridors in manual levels enhance gameplay by requiring less movement to reach map objectives. This scenario contradicts the fitness function of the level generator, which favors levels with larger graph diameters. Additionally, the symmetry observed in some manual levels positively impacts their aesthetics. These factors explain the higher manual level ratings.

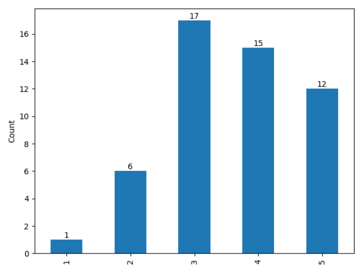
Figure 7 present the structural **similarity**, **generation percentage perception** and **generation perception rate** between the generated and manually created levels. Figure 7(a) suggests moderate similarity. Figure 7(c) shows how much players perceive the level to be generated (a score of 5 indicates full awareness), with perceptions distributed across the spectrum. While some players easily recognize the generated nature of the level, others do not. Lastly, Figure 7(b) illustrates the proportion of the level that players believed was generated. Since we generated the layout, enemies, and their behaviors, we can conclude that most of the level was generated, aligning with the graphical representation.



(a) Similarity between manual and generated levels.



(b) Generation Percentage.



(c) Generation Rate.

Figure 7. Players' perception of similarity, percentage and rate of generation.

Figure 8 shows that half of the players were correct and which levels played are part of the GA-generated group. From this, it is possible to divide the players into two groups and relate them to the data obtained from the similarity and expressiveness questionnaires. As shown in Figure 8, players who more easily perceived the level's generation achieved higher success rates in the level identification test.

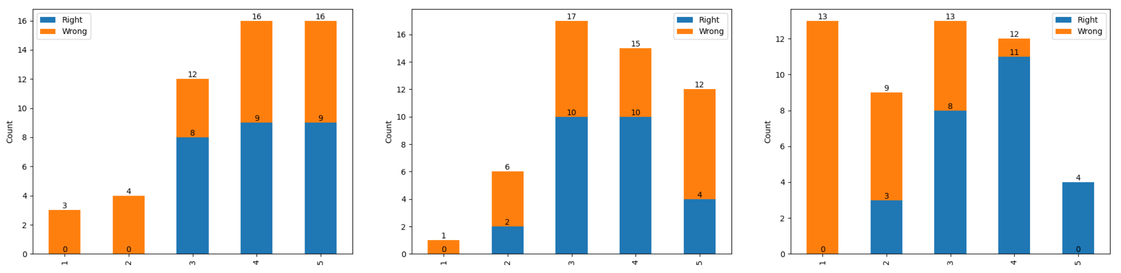
Another observation from these groups is the generation percentage: players who correctly identified the level believed fewer parts of it were generated compared to those who failed the identification test. However, the limited sample size prevents a clear explanation of this relationship. Regarding layout ratings, players who accurately identified the generated level tended to give higher scores to the manual level layout. This suggests that the level in question was likely created manually. Analysis of the manual levels revealed distinctive characteristics that set them apart from generated levels. We gathered critical comparisons between the generated and manual levels from the survey results, focusing primarily on immersion and difficulty differences. Several improvements are needed for the generated levels to match the quality of manually created content. Additionally, feedback from some players outside the scope of the research, although not documented in this work, guided possible improvements to the current algorithms.

5.3. Threats to Validity

A threat to the validity of this work is the small sample size of players, the majority of players who participated in the survey have similar profiles, belonging to the same age group, location and have experience with games, which may prevent the results from being generalized.

6. Conclusion

In this paper, we presented the development and evaluation of the *Dungeoncide* game, which utilizes Procedural Content Generation techniques and metaheuristic algorithms



(a) Layout - Turing Test (b) Generation Perception Rate - Turing Test. (c) Generation Percentage Perception - Turing Test.

Figure 8. Layout, Generation Percent, and generation graphs, related to the identification test results.

to automate the creation of game elements. Our approach involved using genetic algorithms to generate dungeon layouts, balance enemy attributes, and adapt a particle swarm optimization algorithm to control enemy behavior during gameplay.

The results from our experiments, which included a player immersion study, indicated that while the generated levels were generally well-received, they were often perceived as less challenging and immersive than manually created levels. The generated levels tended to be more accessible, likely due to limitations in the enemy generation process and the effectiveness of the PSO algorithm in dynamically adapting to player movements. Despite these challenges, the generated levels showed a moderate similarity to manual levels, and many players could not consistently distinguish between the two. This suggests that our approach has the potential for creating feasible and engaging game content, though improvements are necessary to match the quality and challenge of manually crafted levels.

Future work should focus on refining the fitness functions used in the GA to balance enemy attributes better and improve level layout aesthetics. Additionally, enhancements to the PSO algorithm to more effectively track and respond to player movements could increase the challenge and engagement of generated levels. Player feedback highlighted areas for improvement, such as corridor length and symmetry, which should be considered in further iterations of the algorithms.

References

A., K. B. and Pfleeger, S. L. (2008). Guide to advanced empirical software engineering. chapter Personal opinion surveys., pages 63–92. Springer London, London.

Adams, C. and Louis, S. (2017). Procedural maze level generation with evolutionary cellular automata. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*.

Basili, V. and Weiss, D. (1984). A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, 10(6):728–738.

Brown, J. A., Lutfullin, B., and Oreshin, P. (2017). Procedural content generation of level layouts for hotline miami. In *2017 9th Computer Science and Electronic Engineering (CEEC)*.

DAVID CRANE (1981). Freeway. Jogo digital.

GUILLOTINE GAMES (2012). Zombiecide. Jogo de tabuleiro.

Kasunic, M. (2005). *Designing an effective survey*. Pittsburgh: Carnegie Mellon University.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

Khalifa, A., Perez-Liebana, D., Lucas, S. M., and Togelius, J. (2016). General video game level generation. GECCO '16, New York, NY, USA. Association for Computing Machinery.

NINTENDO (1986). The Legend of Zelda. Jogo digital.

Russel, S. and Norvig, P. (2013). *Inteligência Artificial 3^a.ed.* Elsevier Editora Ltda, Rio de Janeiro, Rio de Janeiro.

Thompson, S. K. (2012). *Sampling*. J. Wiley, 3rd edition.

Togelius, J., Champandard, A., Lanzi, P. L., Mateas, M., Paiva, A., Preuss, M., and Stanley, K. (2013). Procedural content generation: goals, challenges and actionable steps. *Dagstuhl Follow-Ups*, 6:61–75.

TOGELIUS, J., YANNAKAKIS, G. N., STANLEY, K. O., and BROWNE, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186.

Unity (2021). Unity. Ferramenta de criação de jogos digitais.

Zafar, A., Mujtaba, H., and Beg, M. O. (2020). Search-based procedural content generation for gvg-lg. *Applied Soft Computing*, 86.