A Procedural Content Generation Algorithm for First Person Shooter and Tower Defense Games

Manuel Bacallado[1], Jesús Miguel Torres[1] and Rafael Arnay.[1]

[1] *Universidad de La Laguna, Escuela Superior de Ingeniería y Tecnología, San Cristóbal de La Laguna, Spain*

Abstract

The purpose of this work is to present the implementation of a rule-based procedural generation algorithm used for the creation of a game that mixes two genres: First Person Shooter and Tower Defense. The algorithm creates 3D levels, where there will be added enemy towers to destroy, base tower to defend, generation of enemies that will have different mechanics, being one of them, the destruction or conversion of the base tower and more elements. Each time the player starts a new game, a new map is generated with the elements located in different positions adding roguelikes features.

Keywords
Procedural Content Generation, Games, First Person Shooter, Tower Defense, Roguelike

1. Introduction

The procedural content generation (PCG) [1] has its origins in the 1980s with the title Rogue [2], a dungeon video game which has inspired and created a whole genre: the roguelike [3]. In this type of video games, the map that makes up the levels is created autonomously through pre-established rules, making each new game different from the previous one, since the elements that compose it take different shapes and positions. First Person Shooter (FPS) games is a sub-genre of action games, which is generally a first-person immersive experience where the player must eliminate waves of enemies to complete levels and advance in the narrative that the game proposes itself. Finally, Tower Defense (TD) games is a sub-genre of strategy games, whose main objective is to defend the base territory from enemy waves. These genres share elements, such as manual generation of maps, waves of enemies and key items or skills. The procedural content generation makes it difficult for the player to create strategies in relation to the surrounding environment due to the randomness of the generated map [4]. This can be counteracted in the game design part, adding object and skills that solve this problem. Nowadays, there are games that combine both FPS and TD genres, such as Sanctum [5] or GROSS [6], but they do it on manual generation, instead of procedural generation, which is what is shown in this article.

In this work, we make a PCG system for the creation of FPS and TD game with roguelike features. The rest of the document is structured as follows: Section 2 discusses how the PCG system is built and the components it integrates. In Section 3, we describe how the gameplay is implemented and tools used. In Section 4, we make some conclusions and discuss future work.

2. PCG DESIGN AND IMPLEMENTATION

In this section, we present our PCG architecture, and we describe our use case of the algorithm implementation in a roguelike FPS - TD prototype.

Proceedings Acronym: Proceedings Name, Month 30–09, 2023, San Cristóbal de La Laguna, Spain
mbacalll@ull.edu.es (Bacallado, M.); jmtorres@ull.edu.es (Torres, J.M.); rarnayde@ull.edu.es (Arnay, R.)
0009-0003-1260-8637 (Bacallado, M.); 0000-0003-4391-0170 (Torres, J.M.); 0000-0002-9074-0488 (Arnay, R.)

2.1 The PGC system

The architecture of the PCG system is composed for three modules that share data with each other: Map Generation, Tower Spawn Position Generation and The Enemy Generation. The game contains only one level generation, which occurs procedurally each game. With this system, and through the internal modules, it is possible to create new level shapes for the game. Although the enemies and objects are the same, the presentation of the set makes the player always feel that he is in a different game.

2.1.1 Map Generation

The map generator system is a procedurally generated algorithm based on the Random Walk algorithm [7], where starting from an initial position (x=0, y=0, z=0), new paths are created using the next cardinal positions (north, south, east, west) shown in Figure 1, adding the new position randomly generated with the previous one and storing the generated route. The algorithm is normally built on 2 dimensions and for this project it has been adapted to 3 dimensions, modifying the attributes so that the coordinates are now x and z, instead of x and y, and changing the 2D sprites for 3D models.

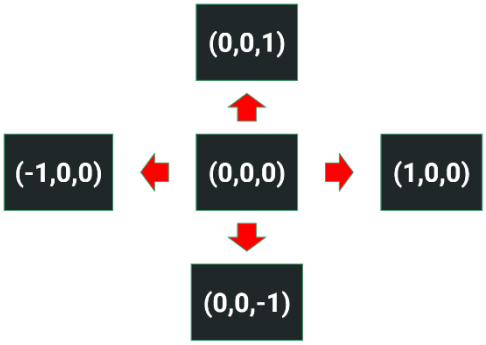

Figure 1: Origin position and cardinal positions

The algorithm has two parts: the first one to generate the path where the floor tiles are positioned, and the next one to create the walls tiles and give the level the appearance of a dungeon. In this case, the tiles are 3D models to represent the art of the game. The rules on which the algorithm is executed are as follows: the number of iterations, length of walk in each iteration, and whether each iteration starts at a random position as shown in Figure 2. The attributes are stored in a user-modifiable structure.

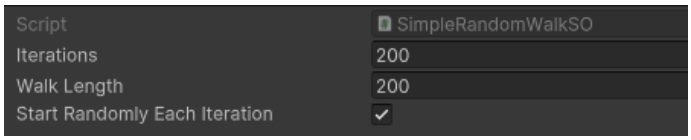

Figure 2: Random Walk rules

Once the floor path has been created, a method based on binary values is used to know the location of the wall tile to be inserted, since the walls have different representations depending on the position in which they are located. This method is used twice: once for cardinal walls, and once for diagonal walls. In both cases, we iterate over the list of floor coordinates obtained earlier and depending on the wall type, we add the position of the floor to a cardinal or diagonal coordinate. To create the binary number, we check if the resulting coordinate exists in the list of floor coordinates. If so, then a 1 is added to a string variable. Otherwise, 0 is added. Once we have the string representing the number, it is converted to integer and compared with the list of specified values of each address to obtain the corresponding model.

Once we have both the directions of the floor and the walls, we use a visualization system to place the corresponding model in each coordinate, and finally have the generate map. Figure 3 shows the complete result.

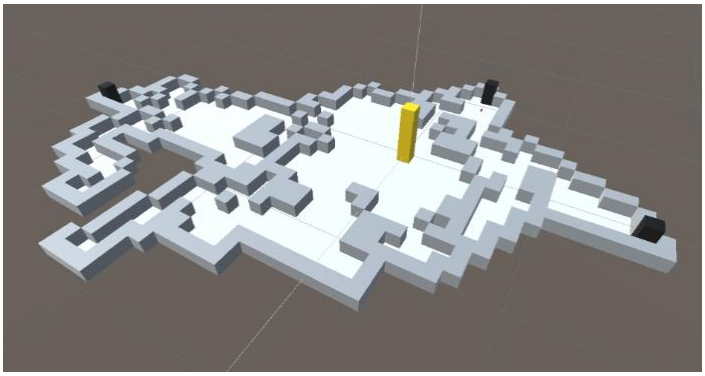

Figure 3: Generated map

2.1.2 Tower Spawn Position Generation

After the map has been generated, enemy towers and the tower we must defend are positioned. In this case, for the player's tower, it is always positioned at the coordinate (0,0,0), although the visual representation is different, since the map always is different. Enemy towers are placed randomly at the extremes of the map. These extreme positions are positions where both x and z coordinates have minimum and maximum values. In our prototype there are only three enemy towers, so once instantiated on the map as shown in Figure 3, the remaining extreme position is used to place the player.

2.1.3 The Enemy Generation

The enemy generation gets a random position on the map in which there is no key element such as walls or towers to generate an enemy in that position. There is a generator integrated in each enemy tower, and each generator have a different group of enemies and the interval in seconds of generation. When a tower is destroyed, the generator is destroyed with it. In our prototype, we have five types of enemies as shown in Figure 4. Of these enemies, two attack the tower directly (EnemyConverter and EnemyFollowTower), while the rest goes after the player (EnemyGun and EnemyFollowPlayer) or remains blocking the map (EnemyRadar).

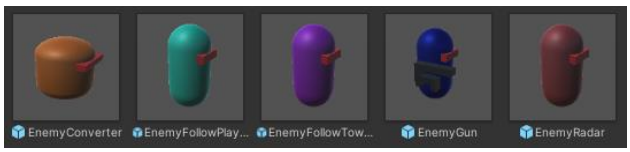

Figure 4: Group of enemies

3. Game Implementation

The proposed PCG system was developed for the game with the code name: Project Survivor. In the game, the player's mission is destroy the enemy towers before the enemies convert or destroy the player's tower or kill the player. If the tower is converted, all enemies target the player and increase their speed to reach and destroy it, greatly increasing the difficulty of the game. In case of destroying it, the game is simply over and a new one would have to be started from scratch. In this pre-alpha version of the prototype, the enemies have a basic artificial intelligence performing the behaviors described in section 2.1.3 and the player has only one weapon available.

The game was developed with the Unity game engine [8]. For the art, and being a prototype, basic engine models have been used to represent the map, towers, enemies, weapons and bullets. Figure 5 shows one gameplay screen with all the elements in the scene.

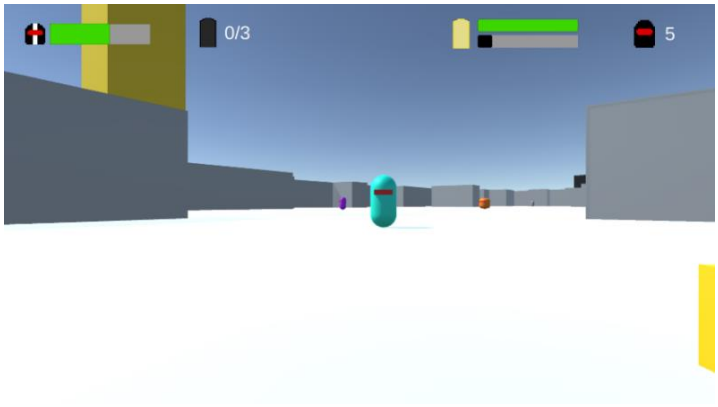

Figure 5: Gameplay screen

After testing the game when this pre-alpha version has been completed, it has been found that it is necessary to adjust the generation times of enemies and add obstacles to slow down the player's progress, and on several occasions, it has been possible to reach the enemy towers with easily. Apart from adding new weapons for the player and objects of healing and destruction.

4. Conclusion

In this paper, we propose an algorithm capable of generating the game and level structure for a game that mixes the FPS and TD genres. During the development of the game, enemies have been designed to challenge the player, such as enemies that directly target to the base tower. In future works, new skills, items, enemies, obstacles, and different challenges for the player will be implemented. Also, add extra complexity to the map by making it smaller every so often, generating the redesign procedurally but respecting the current state of the elements on the map. In addition, playtesting sessions with external users will be carried out to collect as much data and feedback as possible and make the appropriate changes.

References

[1] J. Togelius, J. Whitehead, and R. Bidarra. "Procedural Content Generation in Games (Guest Editorial),". IEEE Transactions on Computational Intelligence and AI in Games, 3 (3) pp. 169–171, Sep. 2011. doi: 10.1109/TCIAIG.2011.2166554.
[2] "Sanctum", 2011. https://coffestainstudios.com/games/sanctum/ (accessed Oct. 15, 2023)
[3] "GROSS", 2023. https://hangryowl.games/gross-press-kit (accessed Oct. 15, 2023)
[4] D. T. Mattheus, Rogue – Video Game, Ventana Press., Inc., 2012.
[5] V. Cerny and F. Dechterenko, Rogue-Like Games as a PlayGround for Artificial Intelligence - Evolutionary Approach, in: Proceedings of the Internacional Conference on Entertainment Computing, Sep. 2015. doi: 10.1007/978-3-319-24589-8_20
[6] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead, "PCG-based game design: enabling new play experiences through procedural content generation," in Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, New York, NY, USA, Jun. 2011, pp. 1–4. doi: 10.1145/2000919.2000926.
[7] F. Xia, J. Liu, H. Nie, Y. Fu, Liangtian and X. Kong. "Random Walks: A Review of Algorithms and Applications". IEEE Transactions On Emerging Topics in Computational Intelligence, 4 (2): 95-107, April. 2020. doi: 10.1109/TETCI.2019.2952908
[8] "Unity", 2023. https://unity.com (accessed Sep. 27, 2023).