A Novel Method in Procedural Maze Generation

Akash Ajith
Computer Science and Engineering
Vimal Jyothi Engineering College
Kannur, India
akashajithcheleri123@gmail.com

Sharon Rose Babu
Computer Science and Engineering
Vimal Jyothi Engineering College
Kannur, India
sharonrose2001sep@gmail.com

Sangeeth K
Computer Science and Engineering
Vimal Jyothi Engineering College
Kannur, India
sangeethkaneesh1@gmail.com

Sidharthan A.K
Computer Science and Engineering
Vimal Jyothi Engineering College
Kannur, India
sidharthanak28@gmail.com

Dr. Manoj V. Thomas
Computer Science and Engineering
Vimal Jyothi Engineering College
Kannur, India
manojkurissinkal@vjec.ac.in

***Abstract*—The concept of procedural content generation (PCG) in game development has existed for a long time. It is used in games for the generation of unique content which help in making the game re-playable. Procedural content generation can be used in almost all game design areas. From level generation to creating a storyline for the game, the use of PCG helps in decreasing the overall time required to design an interesting game. The only problem with PCG is that it is hard to implement and optimize. This document consists of an algorithm that works on a type of recursion and the concept of snappable meshes. This is done using prefabs and other features that are available in Unity Engine[6]. All the methods mentioned in this document are done using Unity Engine. Unity Engine is one of many famous game engines that are available online. The algorithm mentioned helps in creating a procedural maze. The game levels are generated dynamically, allowing the player to experience new levels and avoid repetition of the same levels as in traditional games. The algorithm and its implementation in Unity Engine are explained in detail. How the meshes are spawned and placed dynamically to generate a level is also discussed.**

***Index Terms*—Procedural content generation, level design, game development, unity engine**

I. Introduction

The term Procedural Content Generation is used in games where the game contents or data is created algorithmically. In traditional games, levels are created manually by the designer using the different combinations of assets made using one or more of the modeling tools. Whereas in PCG, this is achieved by random generation of content where different algorithms are used to create content for the game[3]. The usage of PCG helps in creating dynamic content that offers a wide range of options, thus increasing the replayability of the game. Though PCG helps in reducing the storage used and time needed for development, implementing this in games comes with its own set of complexities. For implementing PCG in a game, a lot of different complex algorithms have to be implemented so that there would be a minimum number of bugs. Furthermore, once the development is completed, the game needs to be tested many times to ensure its proper functioning. In PCG algorithms that are currently in practice, most of the complications arise during the implementation stage. Therefore there emerges a need for an even more simple algorithm that is easier to understand and implement. Although the methods that are mentioned in this paper don't eradicate the above-mentioned difficulties, it does help in better understanding the concept. This is done by keeping the requirements of the levels generated using PCG to a minimum. By making use of a game engine such as Unity, most of the difficulties encountered during the project build can be avoided. It also helps in better error detection and correction of bugs during the development phase. Many of the optimization features available in the engine can also be utilized to ensure a smooth game that is capable of running on devices that are having lower specifications in terms of device hardware. Games that can run on low-end devices tend to have a larger audience, this is essential for any new product being released in the market.

II. Related Work

Procedural Content Generation is mostly used in creating puzzles for games. Puzzles that are generated using traditional ways are found to be repetitive and with time they slowly drop from being entertaining. Such games fail to keep the players interested in the long run. This greatly affects the overall player experience. PCG can be used as a new way of making unique and interactive game levels. It can also be used for the generation of very complex and large open worlds. Such levels would normally require hours of work for completion but with the help of PCG they can be done faster. Such a level can act as a better and more interesting challenge for the player. The time required for level generation in next-gen games is drastically high for maps/levels that have bigger magnitudes or high detail[5]. PCG can be aimed to reduce the time required to produce large-scale game levels. There are a variety of algorithms that exist for random generation of game levels. One such algorithm is done using the concept of minimum spanning trees[2]. Using such algorithms, the maps

978-1-6654-9291-1/22/$31.00 ©2022 IEEE

can be created instantly and error-free. They also open up ways to create maps that are of higher quality and can provide more control over the generated map.

III. Maze Generation Algorithm

An array can be used in unity where each element in the array is a Prefab. GameObjects can be made into prefabs so that they can be used as a combination of 3d models along with code and other components provided by the engine. The shapes shown in Figure 1 are 3d models created from modeling software. The 3d models shown are all available on the unity assets store[17]. Unity assets store[16] is a platform where 3d models and other game assets can be bought and sold. Blender[7] is an open-source software that can be used for creating 3d models. The program used for the maze generation is made using the algorithm given below.

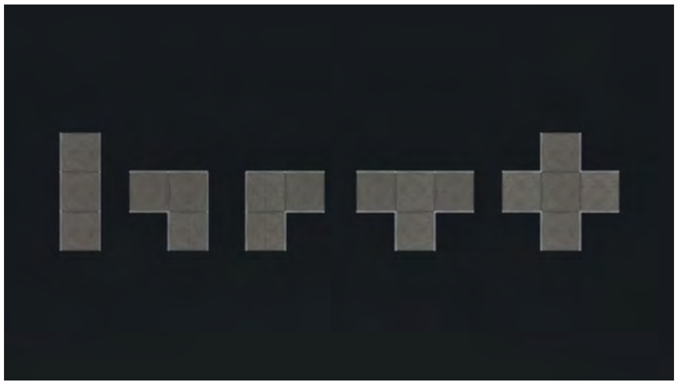
Fig. 1. Elements in the array.

Algorithm 1 Start() function

```
1: Start
2: if (currentPosition within limit) then
3:     if (!deadend) then
4:         Call GenerateNext().
5:     else
6:         Spawn gameobject deadend at current position.
7:         Destroy this gameobject.
8:     end if
9: else
10:    Spawn gameobject deadend at current position.
11:    Destroy this gameobject.
12: end if
13: Stop
```

Algorithm 2 OnTriggerEnter() function

```
1: Start
2: if (Collided object has tag = ”wall”) then
3:     deadend = true.
4: end if
5: Stop
```

Algorithm 3 GenerateNext() function

```
1: Start
2: Set next = Random object from array.
3: Calculate rotation for the ”next”.
4: Rot += prevRot.
5: if (Rot = -90 or 270) then
6:     x = -offsetPosz
7:     y = offsetPosy
8:     z = offsetPosx
9: else if (Rot = -180 or 180) then
10:    x = -offsetPosx
11:    y = offsetPosy
12:    z = -offsetPosz
13: else if (Rot = -270 or 90) then
14:    x = offsetPosz
15:    y = offsetPosy
16:    z = -offsetPosx
17: else
18:    x = offsetPosx
19:    y = offsetPosy
20:    z = offsetPosz
21: end if
22: Assign the calculated position for ”next”.
23: Assign the calculated rotation for ”next”.
24: Call Maze Generation script on ”next” and
25: Set prevRot = current rotation.
26: Stop
```

IV. Implementation

The primary goal is to create a procedural maze where the levels are created efficiently. The focus is to make the algorithm simple enough so that even a beginner can grasp the concept. It should be kept in mind that the overall size of the puzzle that is generated should not exceed a certain limit so that it can be solved without taking too much time. If the generated puzzle is too large, then the player may start to lose interest. There should be a set of rules that would prevent the maze from going over a specified limit. The algorithm should also be able to reduce the workload on the artists and provide a maze in a cost-efficient and productive manner.

A. *Concept of Random Spawning*

Spawning in games is the creation or generation of objects or entities. The object/entity can be the player, Non-Playable Characters(NPC), level contents, or other Mobile Creatures(mobs). Random Spawning is the process of randomly creating such objects within the game. In Unity, this is achieved through making use of prefabs. A Prefab can be used to create or store GameObjects completely with all its components and features. Game Objects are better described as the building blocks of a scene. They are the containers that hold all the components that decide how it looks and what it can perform.

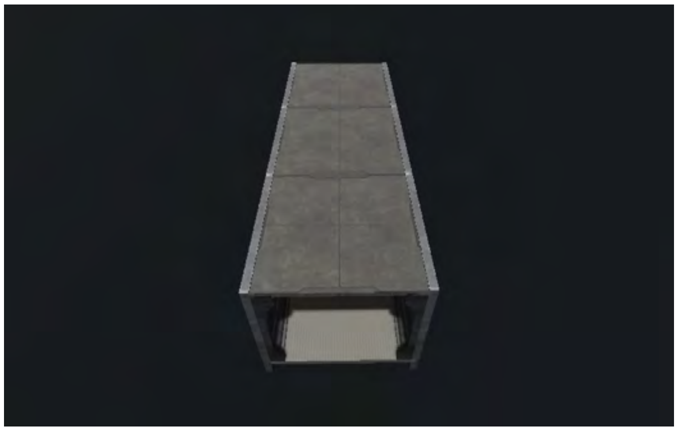

Fig. 2. Straight Path.

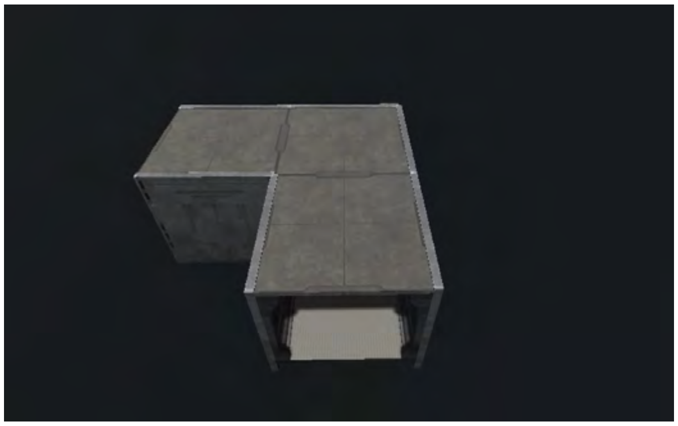

Fig. 3. Left Turn.

B. Concept of Snappable Meshes

The concept that we propose can be implemented by making use of a set of pre-made templates that are made using prefabs. The algorithm makes use of a set of random functions that instantiate an array of GameObjects. These objects would then combine to form the level[1]. The objects would include the shapes shown in Figure 1. The objects used are 3D models that are made into prefabs in Unity, with each object containing the maze generation code. These shapes snap together to make up the basic structure of the game. The 3D shapes used can be seen in detail in Figure 2-6.

C. Building the project

Initially, there would be only a straight line path Figure 2 that contains the Maze Generation program. This program consists of 3 methods.

- Start() method :
  This method is called at the start of the program. It checks if the position of the current object i.e. the 1st object is within the limit. If it is within the limit and if the current object is not a deadend, then the function that generates the next object is called. In all other cases, spawn the

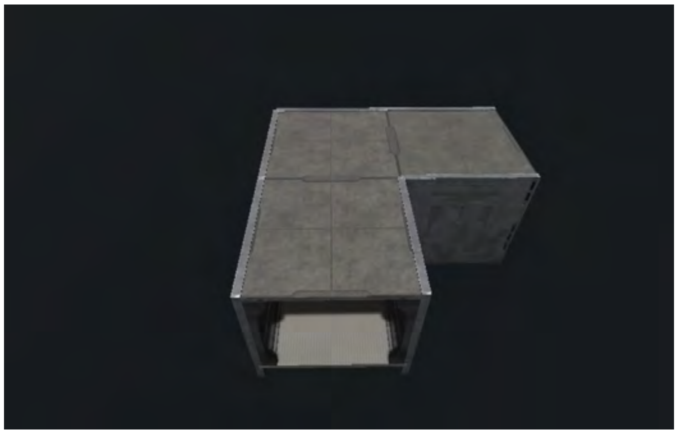

Fig. 4. Right Turn.

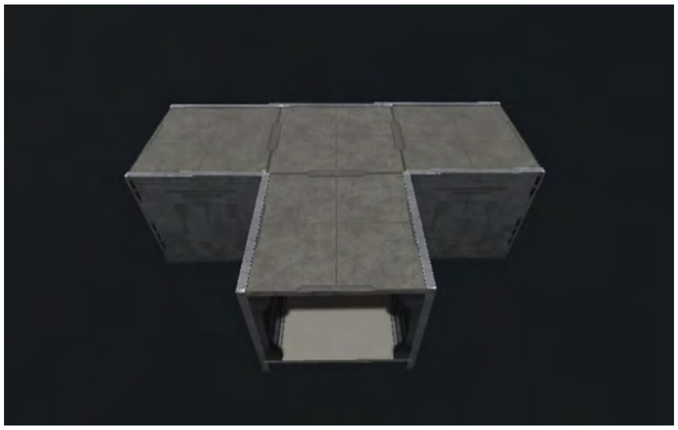

Fig. 5. T-Shape.

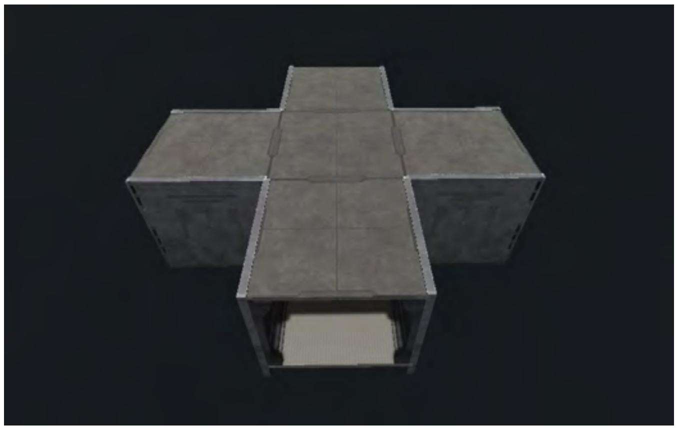

Fig. 6. 4 Way Path.

Fig. 7. The gameobject deadend.

gameobject deadend at the current object position and destroy the current object.

- OnTriggerEnter() method :
  This method gets called when another object enters inside a collider. It is used for destroying the object and spawning a deadend in case of an overlap between the spawned objects. Unity provides a component that can be attached to objects called a collider. This acts as a boundary that surrounds the gameobject. On enabling the Trigger value, this collider acts as a trigger and detects any other colliders that may enter inside this boundary. By default, it also sends a callback to the function OnTriggerEnter() when an object enters. We make use of this feature and checks if the collider of any other object has overlapped with this collider.
- GenerateNext() method :
  This method generates and calculates the transform of the next object. It selects a random value from 0 to the size of the array. Then the element in the array at this index position is taken and instantiated. This object gets randomly spawned into the game scene. The object can be any one of the array elements shown in Figure 1. Each element would also contain its own maze generation program. After instantiating, the function also calculates its position and rotation values. If there is a change in the rotation of the object, the offset values are changed accordingly using the if-else-if blocks. Once the calculations are done, they are assigned to the transform value of the newly formed object.

D. Working of the algorithm using an example

To better understand the algorithm. Let us consider an example. Figure 8 shows a randomly generated maze that was created using the random generation algorithm explained earlier. The starting of the maze is represented by a red block. This indicates the boundary of the 1st straight line object. After its created, the triggers are checked and since no other objects are colliding, the next object is spawned and its prevRot is set to the current rotation which is 0.

The next object created is indicated by the yellow box. This shows the boundary of the 2nd object that was generated. This object now checks its colliders and creates the next object. But in this case, the Rot value of the current object is not 0, which means there will be a change in direction at which the offset is to be applied. Each gameobject has a global variable called Rot. It shows the rotation that the current object will add to the next object. This value is added with the prevRot to find the new rotation value. Since the 2nd object is a left turn, -90 degree is applied to prevRot which equals 0. The next object that is created or the 3rd object's prevRot will be assigned -90.

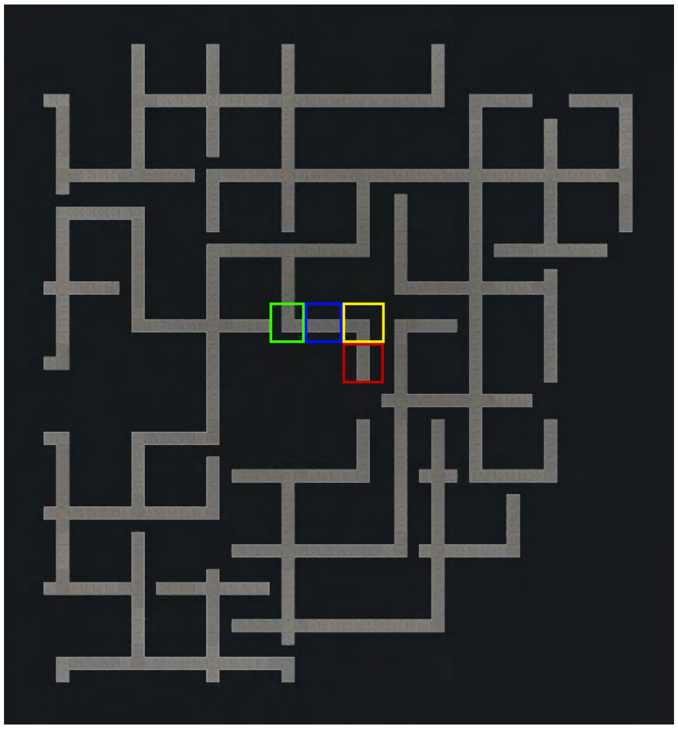

Fig. 8. Working of Random Generation Algorithm.

The blue box is used to indicate the 3rd object. This object now checks for collision and creates 4th object. Since it's a straight path, the old rotation value that is received from the 2nd object is assigned to this and also passed to the prevRot of the 4th object. This continues till the border is reached or an overlap has occurred, ultimately forming the maze shown in Figure 8.

V. Results and Analysis

The final result is the completed maze that is generated within the specified limit and free from any overlaps or errors[18]. Figures 9 and 10 show two such mazes created using the algorithm. Now the player can be spawned anywhere within the maze or at any of the maze endings.

VI. Conclusion

The main advantage of this algorithm is that it is faster and at the same time simple enough for a fresher to understand. It can also be applied in a 3D space by adding gameobjects that have elevation and also considering height in the algorithm.

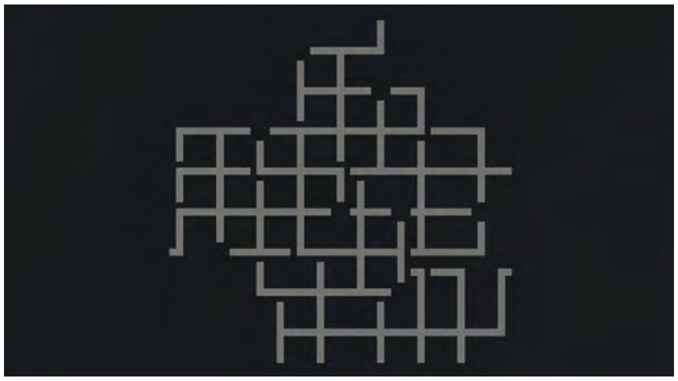
Fig. 9. Example-1 of a Randomly Generated Maze.

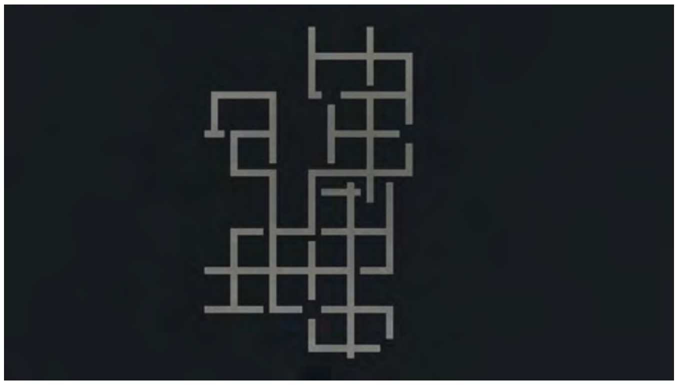
Fig. 10. Example-2 of a Randomly Generated Maze.

Every program has its own set of advantages and disadvantages.

Some of the disadvantages of this algorithm are:

- There will be only one correct path from one deadend to the starting position.
- Proper optimization is needed in Unity which can be harder for a beginner to understand.

From this project, it can be summarized that procedural generation of levels is one of the most advanced fields that could be the leading future of game development. If all the errors and shortcomings in PCG are solved, they could greatly benefit the artists by saving their time and money. The algorithm in this paper could help new aspiring developers to understand the idea of PCG. It could help future artists to understand and further implement and develop fast and efficient programs that would be of use to future developers and designers.

References

[1] Rafael C. E Silva, Nuno Fachada, Diogo De Andrade, Nélio Códices, “Procedural Generation of 3D Maps With Snappable Meshes,” IEEE Access, vol. 10, 2022.

[2] Bartosz von Rymon Lipinski, Simon Seibt, Johannes Roth, Dominik Abé, “Level Graph – Incremental Procedural Generation of Indoor Levels using Minimum Spanning Trees,” 2019 IEEE Conference on Games (CoG), 2019.

[3] Barbara De Kegel, Mads Haahr, “Procedural Puzzle Generation: A Survey,” IEEE Transactions on games, vol. 9, 2020.

[4] Roland van der Linden, Ricardo Lopes, Rafael Bidarra, “Procedural Generation of Dungeons,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, pp. 78–89, 2014.

[5] Remco Huijser, Jeroen Dobbe, Willem F. Bronsvoort, Rafael Bidarra, “Procedural Natural Systems for Game Level Design,” Brazilian Symposium on Games and Digital Entertainment, 2010.

[6] Unity 3D, Unity Technologies. Available [Online]: https://unity.com/

[7] Blender 3D, An open source Modeling software. Available [Online]: https://www.blender.org/

[8] B. Lister, Red Dead Redemption 2, 2018. Available [Online]: https://www.rockstargames.com/reddeadredemption2/.

[9] Bethesda Game Studios, The Elder Scrolls IV: Oblivion: Bethesda Softworks, 2006.

[10] Wikipedia, https://en.wikipedia.org/wiki/.

[11] Github: https://github.com/.

[12] Mojang Studios, Minecraft(3D sandbox game), 2011. Available [Online]: https://www.minecraft.net/en-us/download

[13] Mossmouth, LLC, Spelunky (randomly-generated action adventure), 2008. Available [Online]: https://spelunkyworld.com/

[14] System Era Softworks, Astroneer, 2019. Available [Online]: https://astroneer.space/

[15] Blizzard Entertainment, World of Warcraft: Shadowlands, 2020. Available[Online]: https://worldofwarcraft.com/en-us/shadowlands

[16] Unity Assets Store, Unity Technologies. https://assetstore.unity.com/

[17] Sickhead Games, Sci-Fi Construction Kit (Modular), 2020. Available [Online]: https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-construction-kit-modular-159280

[18] MaZester, 2022. Available [Online]: https://akashajith.itch.io/mazester