Student Name : Ho Kim Chuan
Matric No : 17068452/1
Title : Report Writing - Generating Content- Maze Generation

Introduction

Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithm. It uses a random or pseudo-random process that results in an unpredictable range of possible gam play spaces. Its definition includes using a dynamic as opposed to precomputed light maps, and procedural generation includes using dynamic as opposed to precomputed light maps, and procedurally generated textures, which while procedural in scope, do not affect game play in a meaningful way.

In this assignment, student is given a task to generate a 3D Maze content using different method with at least one PCG method in Unity. There are several content PCG content given by James Buck in his blogpost for example recursive backtracking, Prim's Kruskal's, Eller's Adlous - Broder or Wilson's algorithm, Recursive division, Hunt-and-kill and more.

The instruction given was to evaluate the generators covered in the blogpost and compare the performance of the algorithm used. Due to time constraint, I have chosen one non-PCG algorithm one PCG algorithm to generate the maze content in the Unity Program so that I can see the advantage of PCG in creating games.

I have submitted two programs separately, where the 'Atari-Creation/HO_KIM_CHUAN/Assignment_3_MazeGenerator/Maze Runner' program generating a maze using **Recursive Backtracker Algorithm** while the 'Atari-Creation/HO_KIM_CHUAN/Assignment_3_MazeGenerator/KillandHuntMaze' using the **Hunt-and-Kill Algorithm.** The result and evaluation for these two algorithms is explained in the section of 'Tested Algorithm' below.

Tested Algorithm

Both algorithms are used to generate a randomized 3D maze of 20x20 in size.

The first algorithm used was a non PCG method which is the blogger all-time favorites, the Recursive Backtracker.
The pseudocode for the algorithm:

1. Choose a starting point in the field
2. Randomly choose a wall at that point and carve a passage through to the adjacent cell, but only if the adjacent cell has not been visited yet. This becomes the new current cell.
3. If all the adjacent cells have been visited, back up to the last cell that has uncarved walls and repeat.
4. The algorithm ends when the process has backed all the way up to the starting point.

The Second Algorithm was a PCG method which called Hunt-and-Kill.
The overview of the algorithm goes like this:

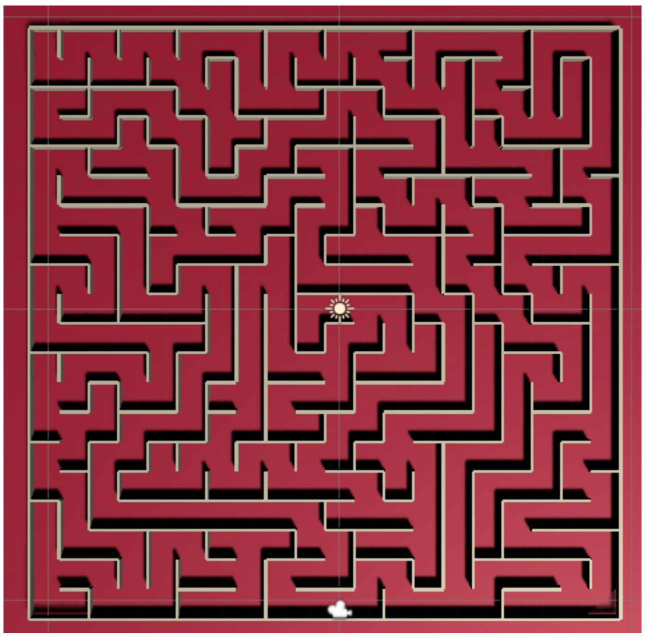
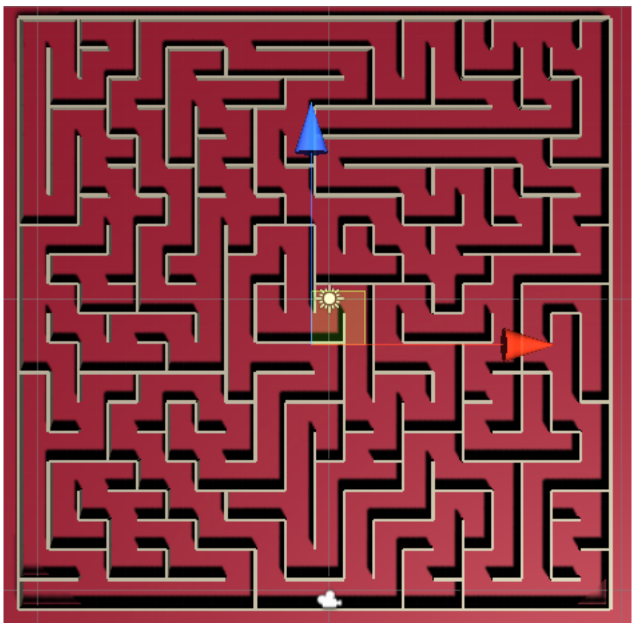
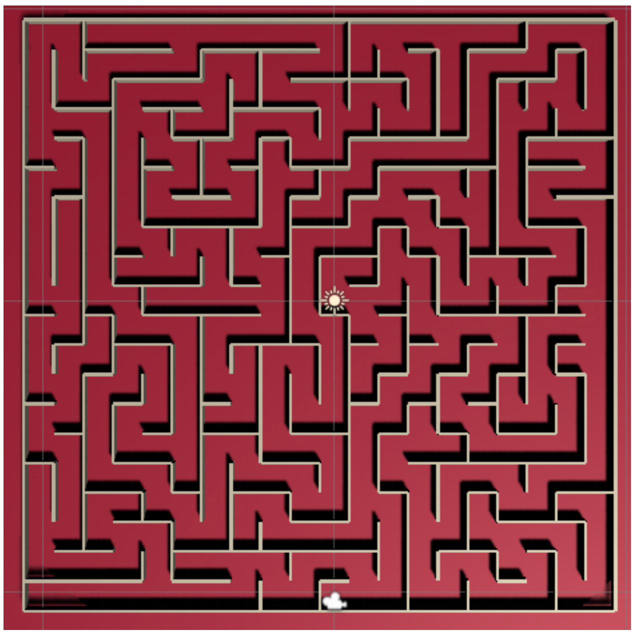
1. Choose a random starting location.

2. Perform a random walk, carving passages to unvisited neighbors, until the current cell has no unvisited neighbors.
3. Enter ‘Hunt’ mode, where the algorithm scans the grid looking for an unvisited cell that is adjacent to a visited cell. If found, carve a passage between the two and let the formerly unvisited cell be the new starting location.
4. Repeat steps 2 and 3 until the hunt mode scans the entire grid and finds no unvisited cells.

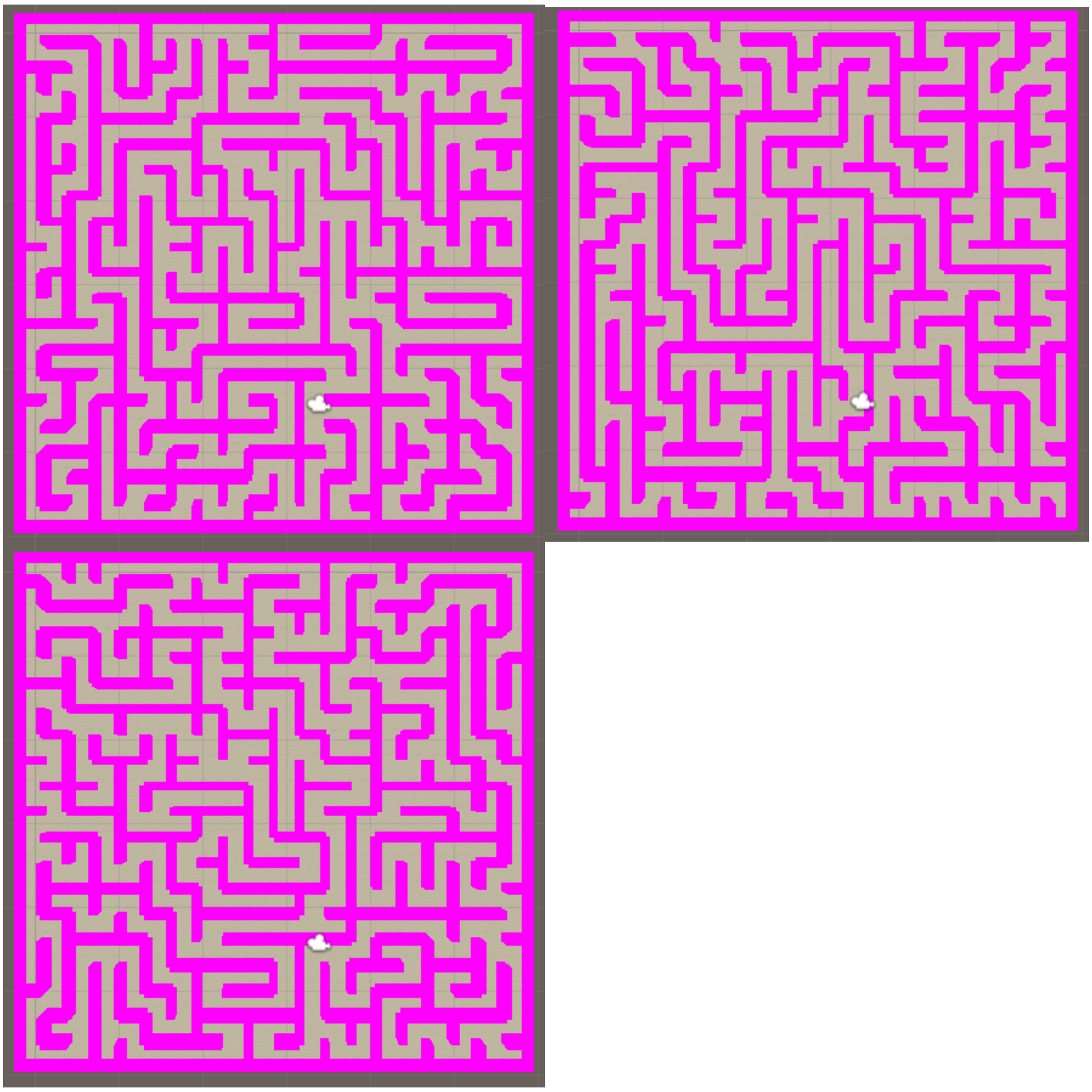
Result

The starting point of a maze is set at the **first tile of upper left corner, and the ending point of the maze is set at the bottom right corner**.

The example results for Recursive Backtracking generated 20 x 20 3D Maze.

The example results for Hung-And-Kill algorithm in generating 20 x 20 3D Maze.



Discussion

The Hunt – and – Kill is quite similar to the recursive backtrackings, the difference is how it handle the dead ends. This is what makes difference for a Hunt and Kill algorithm as when it hits a dead end, the recursive backtracker will choose to take a step back and start another looping or paving, but this algorithm will not step back, instead, it will scan the maze for an uncut cell and choose the point as starting point and continue the process again. It continues until all cell have been cut. As this algorithm doesn't use recursion, it can avoid the potential stack overflows that plague the Recursive Backtracking algorithm for large mazes. It can also minimize the dead ends created.

However, the generated mazed shows that the complexity of a recursive backtracker seems to have more complexity than the Hunt-and-kill. This shows that PCG does not really helps in terms of increasing the complexity of the mazed generated but it does help in saving memory and time complexity in generating the mazed content. Recursive backtracker required large memory in generating maze and the time requirement is definitely greater.

Conclusion

Both Hunt-and-kill and Reverse backtracking did a good job in designing a randomized mazed content and they are renowned in making saving times for content generation. However, Procedural content generation proved to increase in gameplay variety like generating different kind of mazed using a short time and also it is able to create larger content with lesser memory usage compared to non-PCG related algorithm.