# Game 8

Cellular Automata Maze

## Cellular Automata Maze ~ Introduction (1)

- **Procedural Map Generation**, is a technique used in many different games, to enable the generation of different maps, over various runs of the game.
- In essence, Procedural Map Generation Algorithms provide endless unique maps, given little effort from programmers.
- This type of algorithm also provides a variety of benefits to the player, such as being able to repeatedly experiencing the game whilst, simultaneously encountering different environments from one instances to another.
- A type of Procedural Map Generation is **Cellular Automata Maze**. This type of algorithm works similarly to Conway's Game of Life [1].

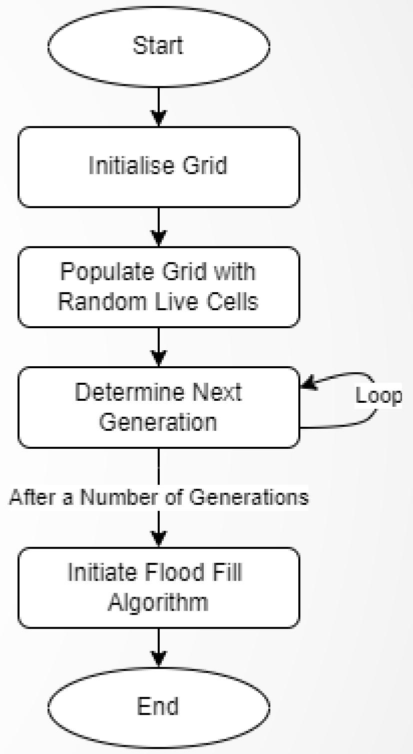
## Cellular Automata Maze ~ Introduction (2)

- As the name suggests, this algorithm utilises a cellular automaton, and a Grid of Cells, to  determine whether cells will survive from one generation to another [1].
-  There are different types of cellular automaton, which determine a surviving Cell. The most common types used are **Mazectric** and **Mice**.
- In **Mazectric** a cell survives from one generation to the next, if it has from 1 to 5 living neighbours.
- In **Mice** a cell survives from one generation to the next, if it has from 1 to 4 living neighbours.
- The implementation carried out will focus on utilising the **Mice** Cellular Automaton [1].

## Cellular Automata Maze ~ AI Explanation (1)

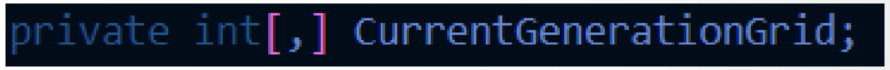
- Implementation of the Cellular Automata Maze can be partitioned in the following steps:

1. **Initialising/Creation of Grid**
2. **Populating Grid with some Random Live Cells**
3. **Determining the Next Generation of Living Cells**
4. **Initiating the Flood Fill Algorithm to determine whether Maze is completable**


Start
Initialise Grid
Populate Grid with Random Live Cells
Determine Next Generation
Loop
After a Number of Generations
Initiate Flood Fill Algorithm
End

# Cellular Automata Maze ~ AI Explanation (2)

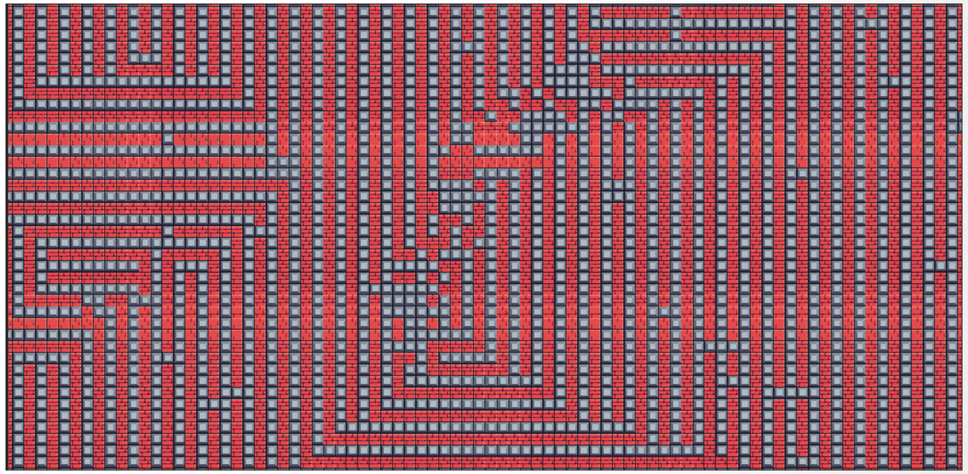
1. **Initialising/Creation of Grid:**



```
private int[,] CurrentGenerationGrid;
```

**Declaration of 2D Array of Cells**

- What is the Grid ?
- The Grid determines the area over which the Maze will be generated. This algorithm is known to be explosive in nature, meaning that a random starting pattern of living Cells will eventually explode in all directions, thus it is imperative that the area which the Maze is applied, would have a predefined size [1].
- In this implementation a 2d array of integers is being used to represent the different Cells, whereby every element in the array represents the state of a cell [2]. The Cells are then mapped to a Tile on the tile map. The living Cells will determine the walls of the Maze and have a red colour.

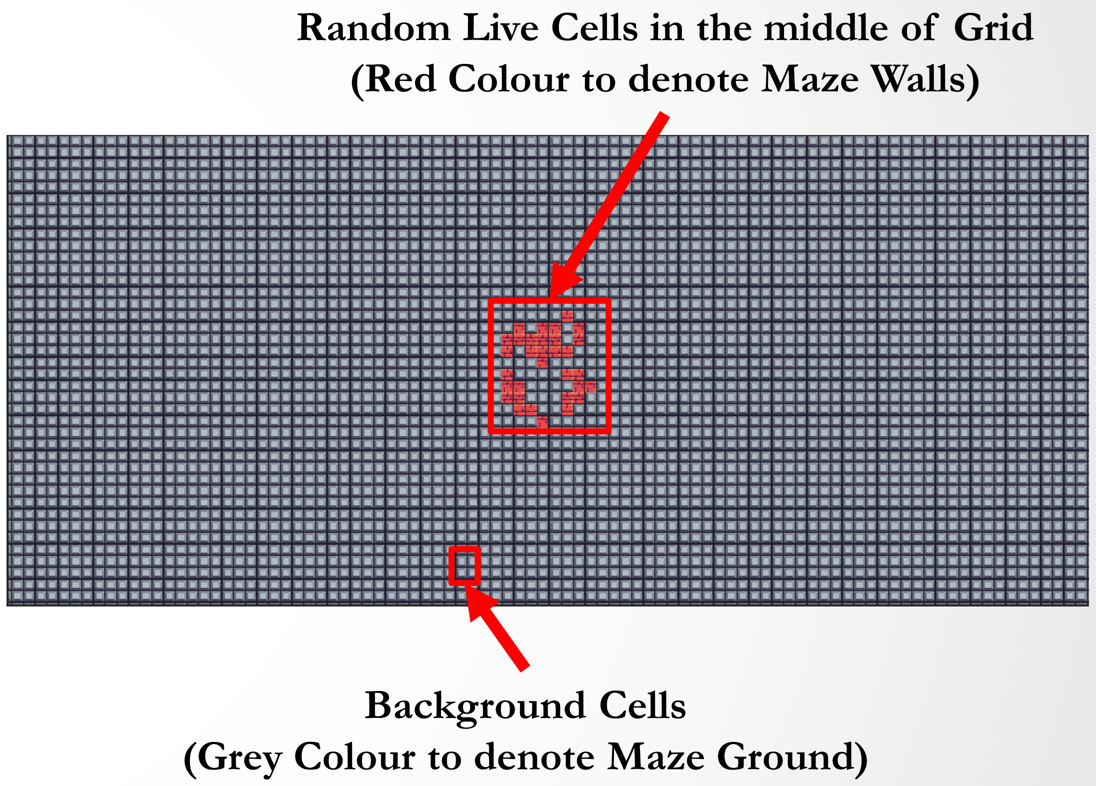


**Visualisation of Cells**

# Cellular Automata Maze ~ AI Explanation (3)

2. **Populating Grid with some Random Live Cells:**

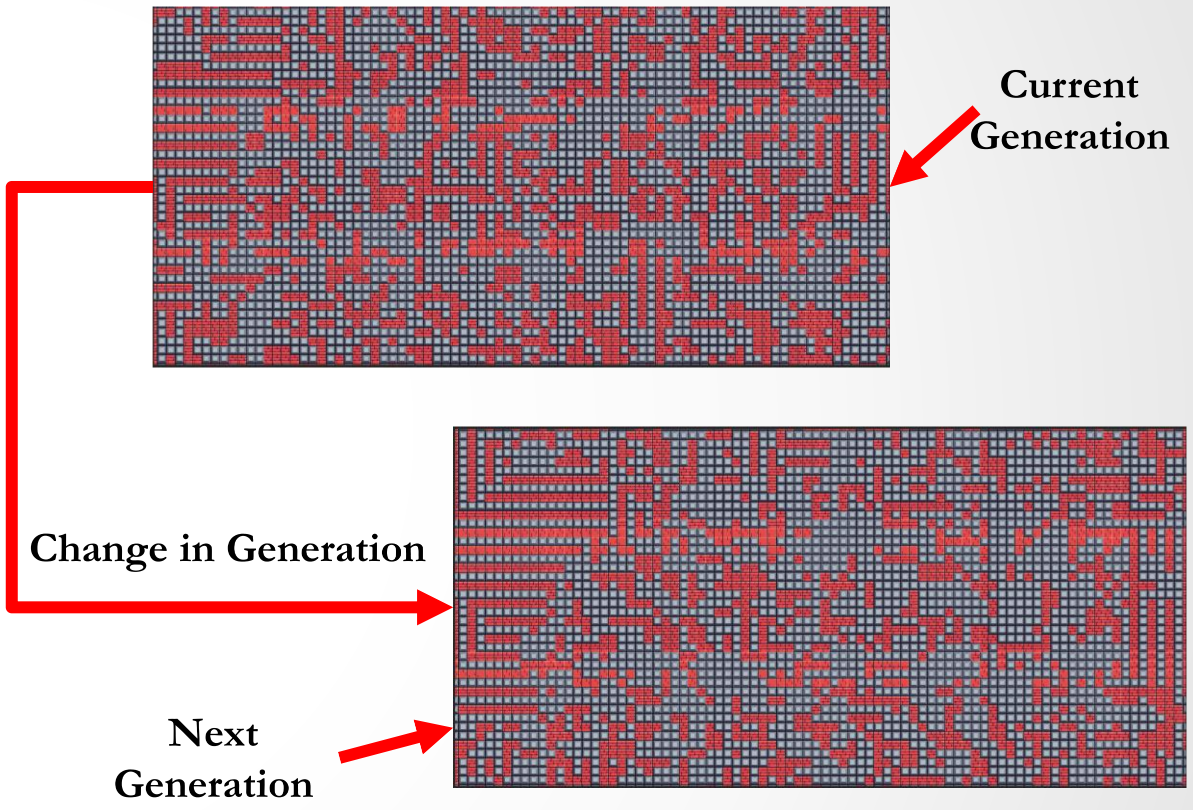
- How is the Maze Generation Initiated?
- After the Grid has been created, and all cells would be initialised to dead cells, some cells would need to be set to live cells, in order to initiate the algorithm to explode in all directions, and for the Maze to evolve into a complex one [3].
- It is also important that the process of populating the Grid with some Live Cells from the middle position of the Grid, is done randomly to always generate a different Maze each time.


Random Live Cells in the middle of Grid
(Red Colour to denote Maze Walls)
Background Cells
(Grey Colour to denote Maze Ground)

# Cellular Automata Maze ~ AI Explanation (4)

3. **Determining the Next Generation of Living Cells:**

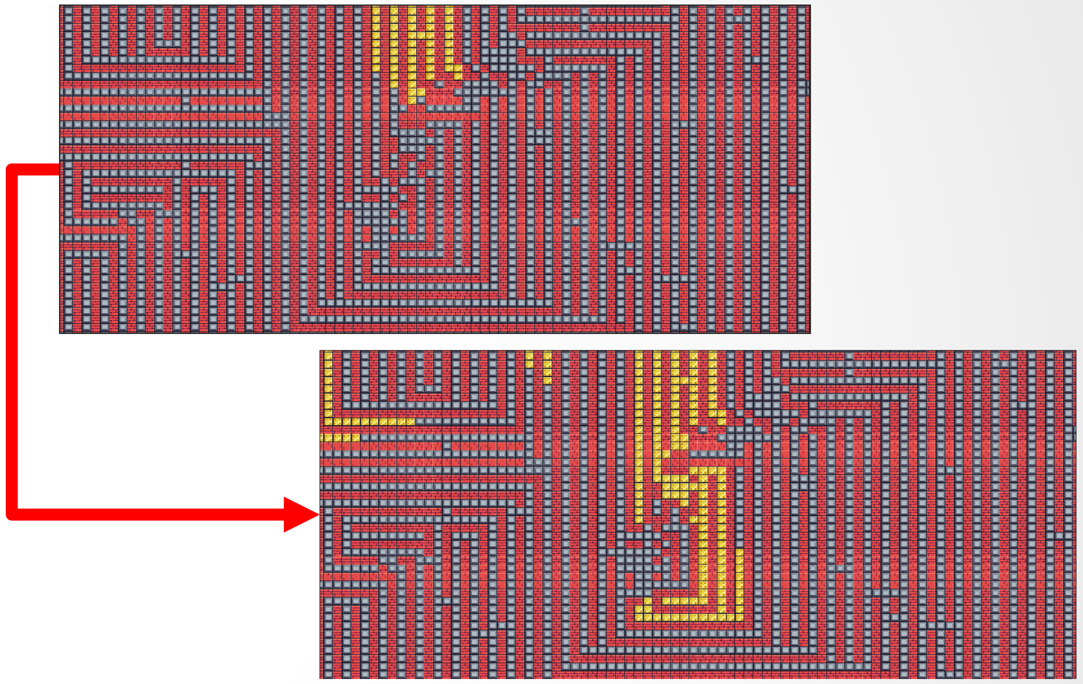
- How is the Maze formed?
- After the Grid has been populated with some Random Live Cells, the Maze could finally be generated, by applying the following rules [2]:
  - Utilise another Grid to hold the next Generation of Cells.
  - Check whether the Cell will live in the Next Generation.
  - If the Cell will live in the Next Generation, then set the respective Cell in the next Generation Grid to living.
  - Finally overwrite the Current Generation Grid with the Next Generation Grid.
- This method will loop for several generations.



# Cellular Automata Maze ~ AI Explanation (5)

4. **Initiating the Flood Fill Algorithm:**

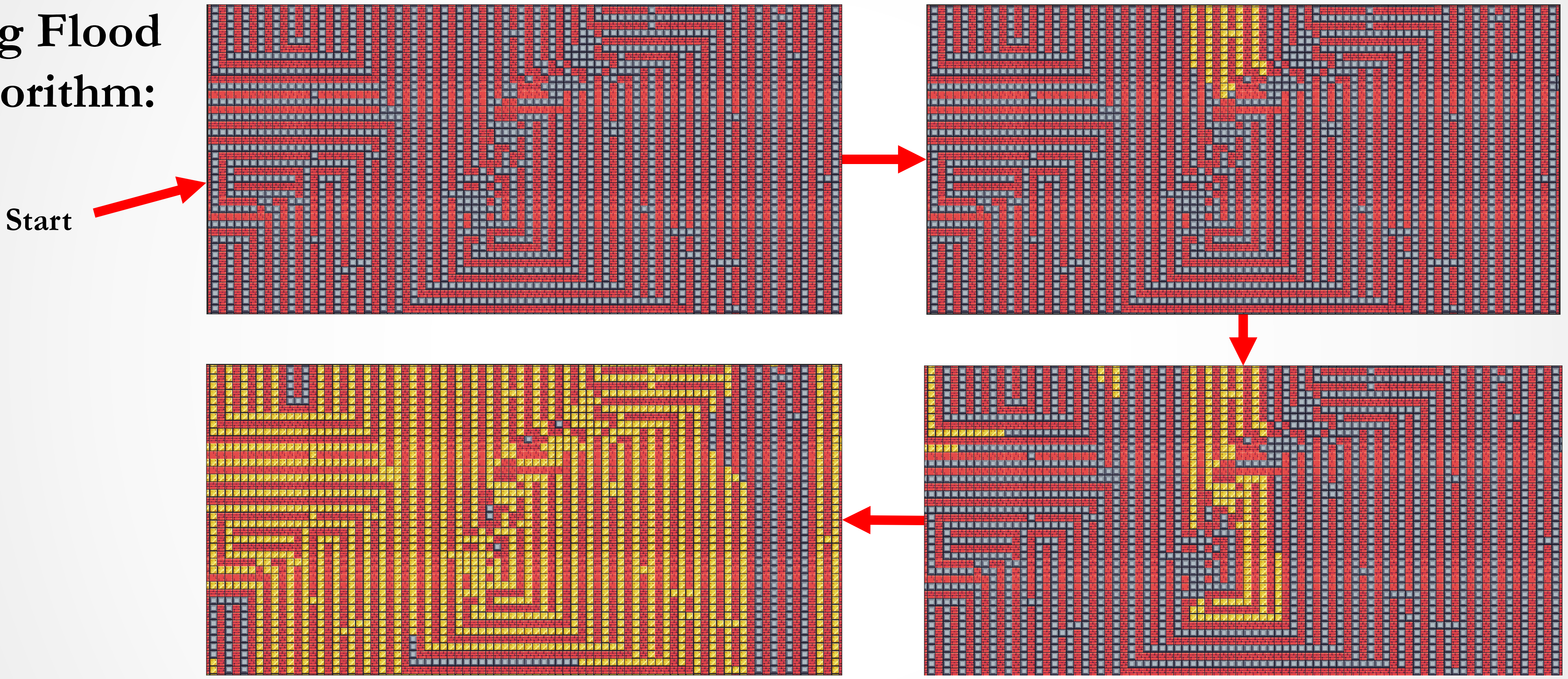
- How to determine whether Maze is completable?
- After the Maze has been generated, the Flood Fill Algorithm needs to carried out, in order to determine whether the maze is completable, and to identify unreachable areas. The algorithm works in the following way [2]:
  - Start the algorithm from the edges of the Grid.
  - Check whether the Cell is not a wall or whether Cell has not already been visited by the algorithm.
  - If not, then return, else set the colour of the current Cell to Gold.
  - Recursively call the Flood Fill Algorithm on the Cell's primary neighbours (North, South, East and West).
- This method can be implemented via either recursion or using a queue and would stop once there are no more Cells to explore.



**Running Flood Fill Algorithm (Gold Colour to denote reachable path)**

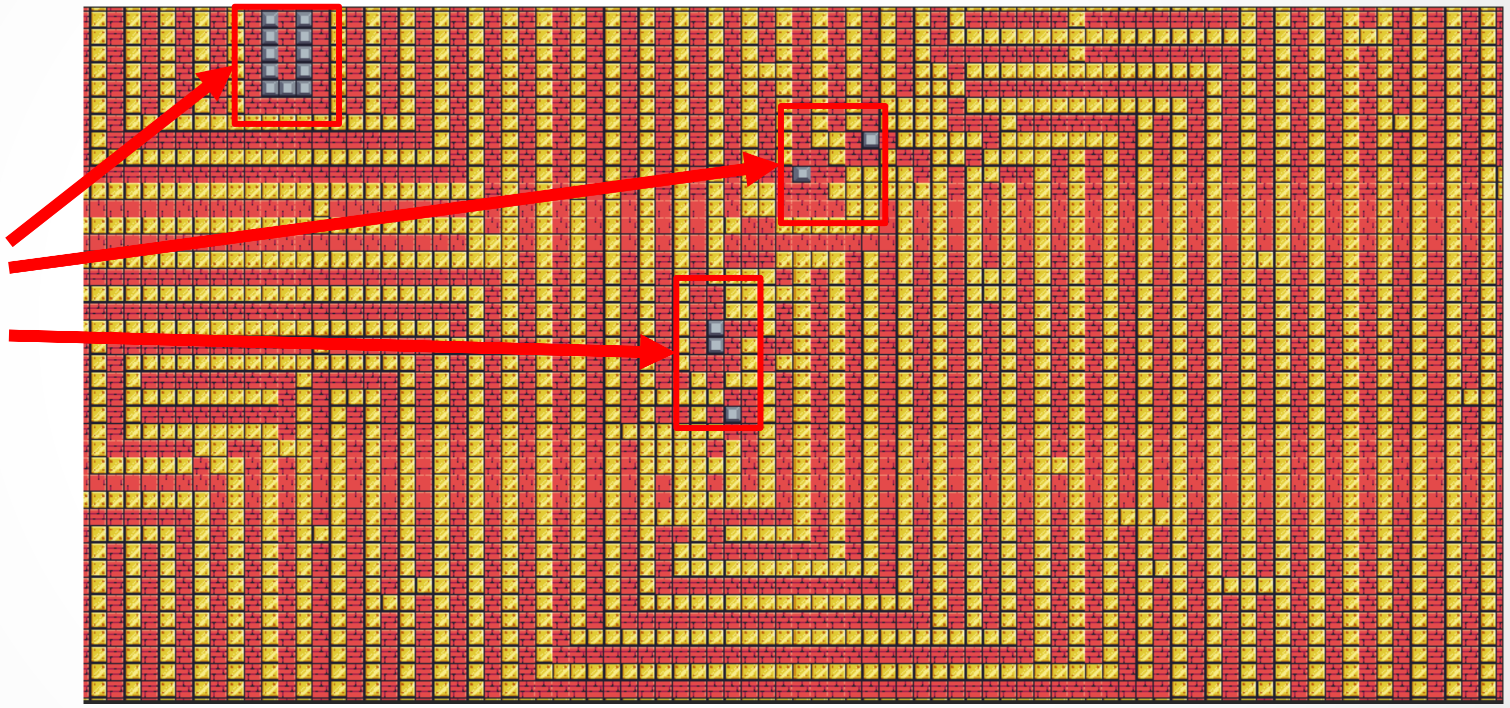
# Cellular Automata Maze ~ AI Explanation (6)

**Running Flood Fill Algorithm:**



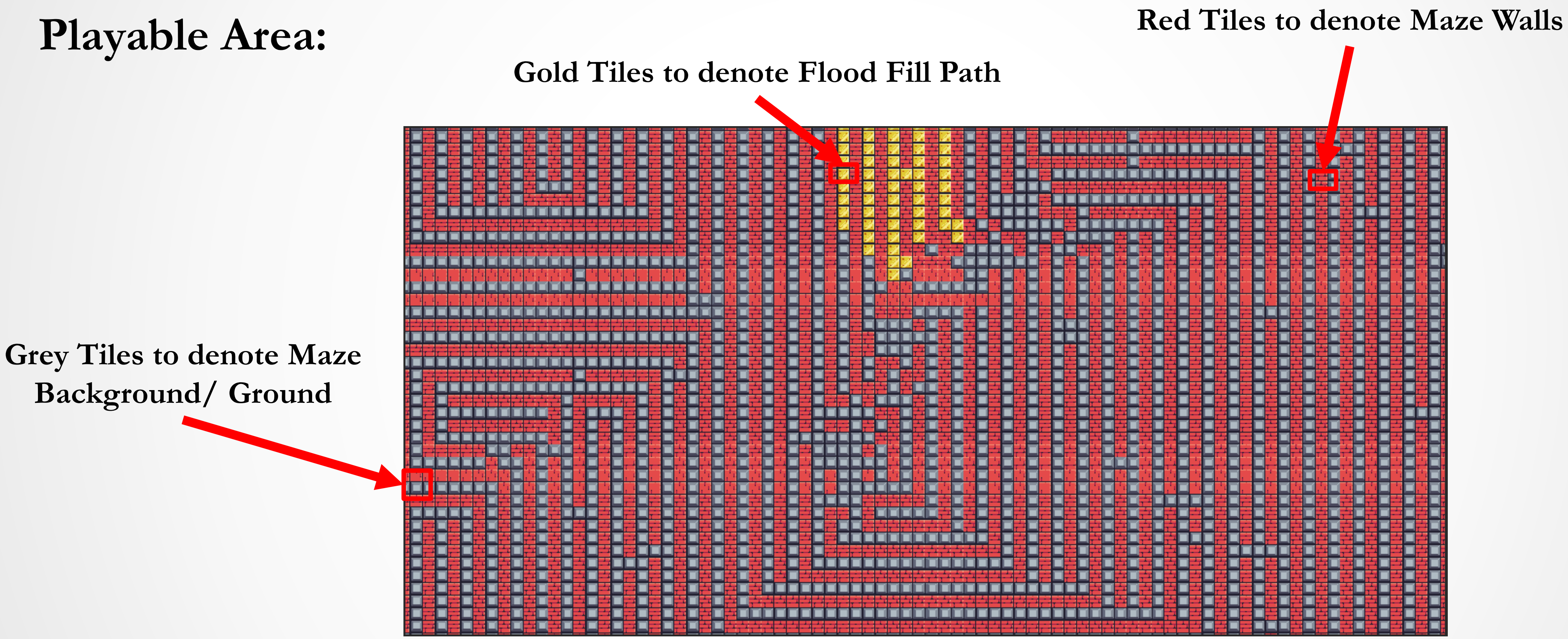
# Cellular Automata Maze ~ AI Explanation (7)

**Identifying Unreachable Areas:**



# Cellular Automata Maze ~ Mini-Game Implementation (1)

**Playable Area:**

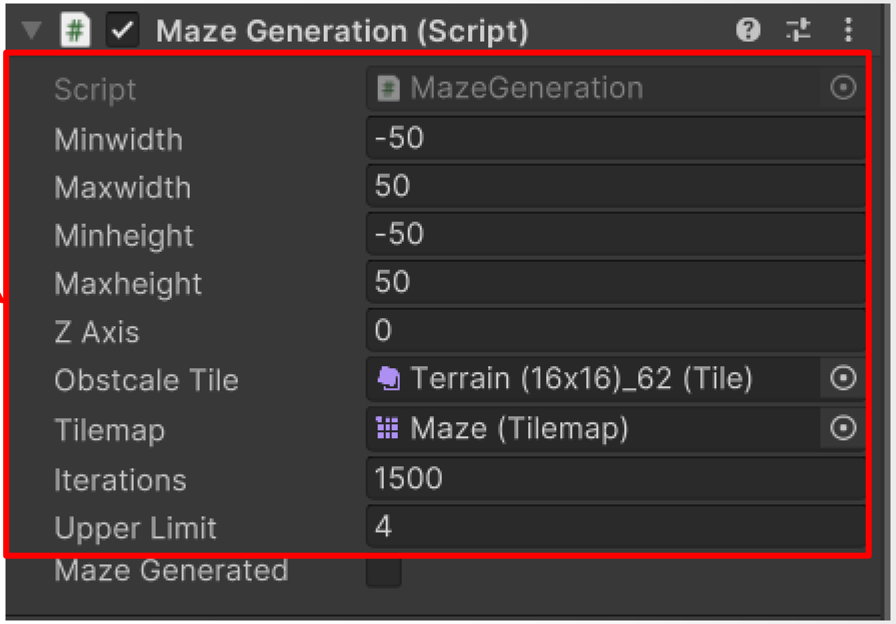

Red Tiles to denote Maze Walls
Gold Tiles to denote Flood Fill Path
Grey Tiles to denote Maze Background/ Ground

# Cellular Automata Maze ~ Mini-Game Implementation (2)

## Developer Interface:

- As can be seen in the following image, developers, are given a large degree of freedom, in which they can configure the algorithm.
- Developers, can choose the area over which the algorithm will apply to, as well as fine tune other algorithm parameters to their liking, through the different input boxes, on the Maze Game Object.
- Developer, can also choose to utilise the **Mazectric** Cellular Automata instead of the **Mice** Cellular Automata by changing the **Upper Limit** from 4 to 5, and vice versa.

**Customization of algorithm parameters**

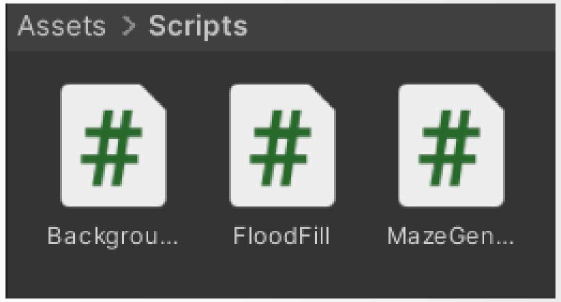


Maze Generation (Script)

| | |
|---|---|
| Script | MazeGeneration |
| Minwidth | -50 |
| Maxwidth | 50 |
| Minheight | -50 |
| Maxheight | 50 |
| Z Axis | 0 |
| Obstcale Tile | Terrain (16x16)_62 (Tile) |
| Tilemap | Maze (Tilemap) |
| Iterations | 1500 |
| Upper Limit | 4 |
| Maze Generated | |

## Cellular Automata Maze ~ Mini-Game Implementation (3)

- Implementation of the Mini Game was inspired from [2,4], and sprites used to create the game were retrieved from [5].
- **The Game is Composed of the following scripts:**
  - **Background** script – This script is being used to generate the Maze Background/Ground.
  - **MazeGeneration** script – This script is being used to Initialise and Populate the Maze Grid, as well as form the Maze Walls.(contains most of the computation for the Cellular Automata Maze algorithm).
  - **FloodFill** script – This script is being used to initiate the Flood Fill Algorithm on the newly formed Maze.



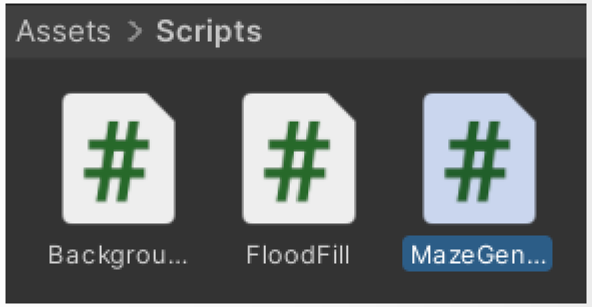
# Cellular Automata Maze ~ Exercise (1)

**Now Its your turn to Code! – Let's implement the Cellular Automata Maze Algorithm ☺**

1. Navigate to the Assets> Scripts folder, and open the **Maze Generation** script
2. In the Maze Generation script find the **Next Generation()** method
3. Utilise the following Pseudocode to populate this method (Next Generation() method)

**Pseudocode:**

1. Loop through all the Cells in the CurrentGeneration Grid
   - (Hint: use a nested for loop, and utilise the variables:rowsInGrid and colsInGrid )
2. Check whether current Cell(i.e., cell in CurrentGeneration Grid) is alive or dead
   - (Hint: if current Cell is 1, then cell is alive)
3. If the cell is alive, set the tile in the tilemap, which has the position of new Vector3Int(x+minwidth, y+minheight, zAxis), to ObstacleTile

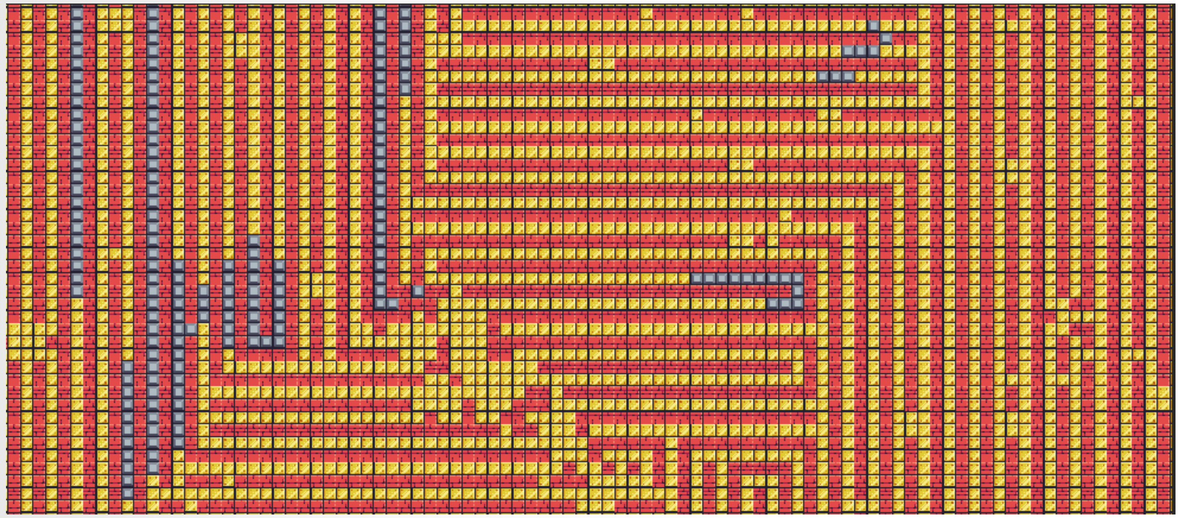


# Cellular Automata Maze ~ Exercise (2)

**Pseudocode Continue…**

4. Else If cell is dead, set the tile in the tilemap, which has the position of new Vector3Int(x+minwidth, y+minheight, zAxis), to null
5. Next inside the nested for loop, also check whether the cell will live in the next generation
   - (Hint: use the CellLivesNextGen() method)
6. If the Cell lives in the next generation, then set the NextGenerationGrid with the index of the current Cell to 1 (alive)
7. Else if the Cell does not live in the next generation, then set the NextGenerationGrid with the index of the current Cell to 0 (dead)
8. At the end of the nested for loop, Overwrite the CurrentGeneration Grid, with the NextGeneration Grid
   - (Hint: use the NextGenerationGrid.Clone() as int[,])

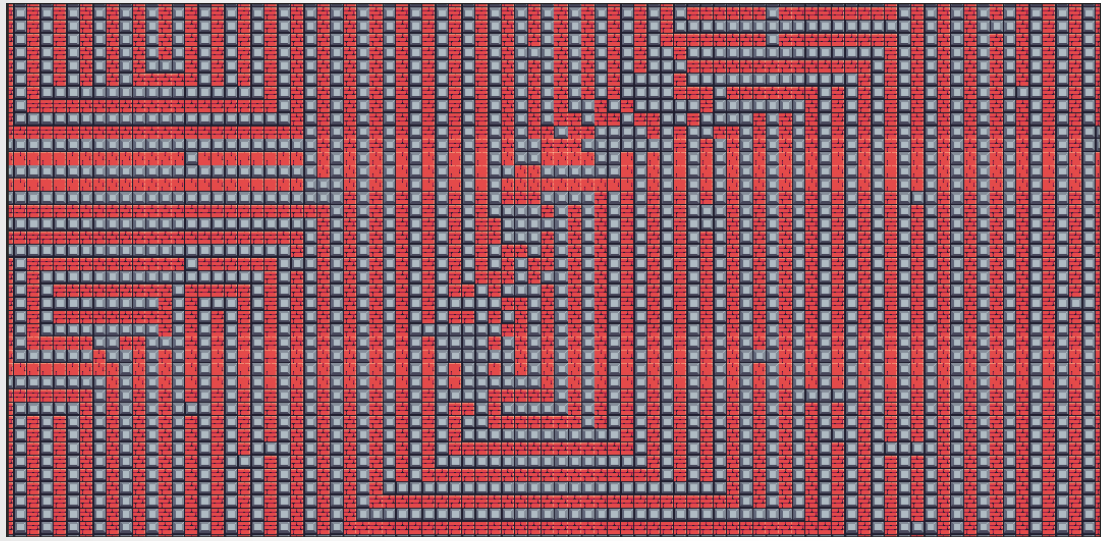
# Cellular Automata Maze ~ Conclusion (1)

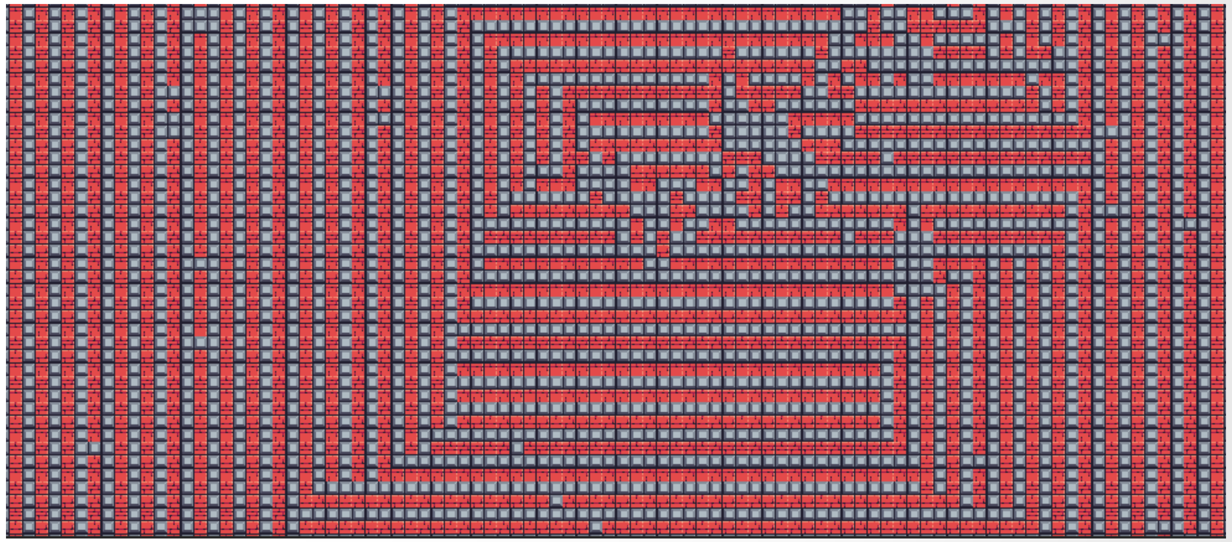


- Cellular Automata Maze can be quite simple to implement, and a different maze is generated every time the algorithm is run.
- The algorithm which was covered in the following PowerPoint, only focussed on the implementation of a Binary Cellular Automaton, i.e., Cells could only be living or dead. In reality, there are other models which implement multiple states and models which utilise a genetic algorithm, to determine the fitness of each maze.
- Through this PowerPoint, the Student would be able to know and identify ways of how Cellular Automata Maze can be implemented, as well as its benefits, and experiment with a type of Procedural Map Generation.

# Cellular Automata Maze ~ Conclusion (2)

**Running the Maze Algorithm, different times, would present a unique solution every time:**





# Cellular Automata Maze ~ References

[1] – LifeWiki, "OCA:Maze" 2022 [Online]. Available: https://conwaylife.com/wiki/OCA:Maze [Accessed: 18-Mar-2023]

[2] – Prof. A. Dingli, ICS2211: "Level5_AutomatedContentGeneration" [Online]. Available: https://www.um.edu.mt/vle/pluginfile.php/1122580/mod_resource/content/1/Level5AutomatedContentGeneration.pdf [Accessed: 18-Mar-2023]

[3] – liquisearch, "Maze Generation Algorithm - Cellular Automaton Algorithms" [Online]. Available: https://www.liquisearch.com/maze_generation_algorithm/cellular_automaton_algorithms [Accessed: 18-Mar-2023]

[4] – Khan Academy, "Maze and Mazectric" [Online]. Available: https://www.khanacademy.org/computer-programming/maze-and-mazectric/6747287273930752 [Accessed: 18-Mar-2023]

[5] – Pixel Frog, "Unity Asset Store: Pixel Adventure 1" 2019 [Online]. Available: https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360 [Accessed: 18-Mar-2023]