# Unit 3 – Lesson 10. Maze Implementation in Unity and CSharp – Part 1

**Aim:**

* How do we write a simple C# script to create a 10 x 10 maze using backtracking or DFS?

**Objectives:** After the lesson, students should be able to:

* Implement simple 3D maze using backtracking or DFS, C# and primitive cubes.

**CLASS PROCEDURE:**

***Do Now:***

Here’s a piece of pseudo code for DFS maze generation someone posted online:

1. Generate an int array with 4 random numbers to represent directions.
2. Start a for loop to go for 4 times.
3. Set up a switch statement to take care of 4 directions.
4. For that direction, check if the new cell will be out of maze or if it’s a path already open. If so, do nothing.
5. If the cell in that direction is a wall, set that cell to path and call recursive method passing the new current row and column.
6. Done.

Does the pseudo code implement the DFS maze generation algorithm correctly?

***Class Discussion / Presentation:***

1. Here’s some sample C# script online for 2D maze generation using DFS. How does it work?

**using** System;  
**using** System.Collections.Generic;  
**using** System.Diagnostics;  
**using** System.Linq;  
**using** System.Text;  
**using** System.Drawing;  
   
**namespace** MazeGeneration  
**{**  
 **public** **static** **class** Extensions  
 **{**  
 **public** **static** IEnumerable<T> Shuffle<T>**(this** IEnumerable<T> source, Random rng**)**  
 **{**  
 **var** e = source.**ToArray()**;  
 **for** **(var** i = e.**Length** - **1**; i >= **0**; i--**)**  
 **{**  
 **var** swapIndex = rng.**Next(**i + **1)**;  
 **yield** **return** e**[**swapIndex**]**;  
 e**[**swapIndex**]** = e**[**i**]**;  
 **}**  
 **}**  
   
 **public** **static** CellState OppositeWall**(this** CellState orig**)**  
 **{**  
 **return** **(**CellState**)(((int)** orig >> **2)** | **((int)** orig << **2))** & CellState.**Initial**;  
 **}**  
   
 **public** **static** **bool** HasFlag**(this** CellState cs,CellState flag**)**  
 **{**  
 **return** **((int)**cs & **(int)**flag**)** != **0**;  
 **}**  
 **}**  
   
 **[**Flags**]**  
 **public** **enum** CellState  
 **{**  
 Top = **1**,  
 Right = **2**,  
 Bottom = **4**,  
 Left = **8**,  
 Visited = **128**,  
 Initial = Top | Right | Bottom | Left,  
 **}**  
   
 **public** **struct** RemoveWallAction  
 **{**  
 **public** Point Neighbour;  
 **public** CellState Wall;  
 **}**  
   
 **public** **class** Maze  
 **{**  
 **private** **readonly** CellState**[**,**]** \_cells;  
 **private** **readonly** **int** \_width;  
 **private** **readonly** **int** \_height;  
 **private** **readonly** Random \_rng;  
   
 **public** Maze**(int** width, **int** height**)**  
 **{**  
 \_width = width;  
 \_height = height;  
 \_cells = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) CellState**[**width, height**]**;  
 **for(var** x=**0**; x<width; x++**)**  
 **for(var** y=**0**; y<height; y++**)**  
 \_cells**[**x, y**]** = CellState.**Initial**;  
 \_rng = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Random**()**;  
 VisitCell**(**\_rng.**Next(**width**)**, \_rng.**Next(**height**))**;  
 **}**  
   
 **public** CellState **this[int** x, **int** y**]**  
 **{**  
 **get** **{** **return** \_cells**[**x,y**]**; **}**  
 **set** **{** \_cells**[**x,y**]** = **value**; **}**  
 **}**  
   
 **public** IEnumerable<RemoveWallAction> GetNeighbours**(**Point p**)**  
 **{**  
 **if** **(**p.**X** > **0)** **yield** **return** [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) RemoveWallAction **{**Neighbour = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Point**(**p.**X** - **1**, p.**Y)**, Wall = CellState.**Left}**;  
 **if** **(**p.**Y** > **0)** **yield** **return** [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) RemoveWallAction **{**Neighbour = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Point**(**p.**X**, p.**Y** - **1)**, Wall = CellState.**Top}**;  
 **if** **(**p.**X** < \_width-**1)** **yield** **return** [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) RemoveWallAction **{**Neighbour = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Point**(**p.**X** + **1**, p.**Y)**, Wall = CellState.**Right}**;  
 **if** **(**p.**Y** < \_height-**1)** **yield** **return** [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) RemoveWallAction **{**Neighbour = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Point**(**p.**X**, p.**Y** + **1)**, Wall = CellState.**Bottom}**;  
 **}**  
   
 **public** **void** VisitCell**(int** x, **int** y**)**  
 **{**  
 **this[**x,y**]** |= CellState.**Visited**;  
 **foreach** **(var** p **in** GetNeighbours**(**[**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Point**(**x, y**))**.**Shuffle(**\_rng**)**.**Where(**z => !**(this[**z.**Neighbour**.**X**, z.**Neighbour**.**Y]**.**HasFlag(**CellState.**Visited))))**  
 **{**  
 **this[**x, y**]** -= p.**Wall**;  
 **this[**p.**Neighbour**.**X**, p.**Neighbour**.**Y]** -= p.**Wall**.**OppositeWall()**;  
 VisitCell**(**p.**Neighbour**.**X**, p.**Neighbour**.**Y)**;  
 **}**  
 **}**  
   
 **public** **void** Display**()**  
 **{**  
 **var** firstLine = **string**.**Empty**;  
 **for** **(var** y = **0**; y < \_height; y++**)**  
 **{**  
 **var** sbTop = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) StringBuilder**()**;  
 **var** sbMid = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) StringBuilder**()**;  
 **for** **(var** x = **0**; x < \_width; x++**)**  
 **{**  
 sbTop.**Append(this[**x, y**]**.**HasFlag(**CellState.**Top)** ? "+--" : "+ "**)**;  
 sbMid.**Append(this[**x, y**]**.**HasFlag(**CellState.**Left)** ? "| " : " "**)**;  
 **}**  
 **if** **(**firstLine == **string**.**Empty)**  
 firstLine = sbTop.**ToString()**;  
 Debug.**WriteLine(**sbTop + "+"**)**;  
 Debug.**WriteLine(**sbMid + "|"**)**;  
 Debug.**WriteLine(**sbMid + "|"**)**;  
 **}**  
 Debug.**WriteLine(**firstLine**)**;  
 **}**  
 **}**  
   
 **class** Program  
 **{**  
 **static** **void** Main**(string[]** args**)**  
 **{**  
 **var** maze = [**new**](https://www.google.com/search?q=new+msdn.microsoft.com) Maze**(20**, **20)**;  
 maze.**Display()**;  
 **}**  
 **}**  
**}**

1. Can this program be improved? How?

***Pair – sharing Activity #1:*** Here’s another online tutorial on maze generation:

<https://www.raywenderlich.com/82-procedural-generation-of-mazes-with-unity>

Read this tutorial, and work with your partner to answer the following questions:

1. What algorithm does this tutorial use for maze generation?
2. Is the algorithm and implementation more efficient? Why and why not?

***Pair – sharing Practice 2 / Lab Activity:***

1. Do you remember this program we explored in the first week September? What does this program do? How do we generate cubes on the run?

using UnityEngine;

using System.Collections;

public class Test2 : MonoBehaviour {

public float NextTime = 0f;

public float Timer = 0.5f;

public int iCounter = 10;

public int MinHeight = 1;

public int Maxheight = 10;

public float HorizontalSpacing = 2f;

public float VerticalSpacing = 1f;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (iCounter > 0 && Time.fixedTime > NextTime) {

NextTime = Time.fixedTime + Timer;

for (int j = 10; j > 0; j--)

{

int randomNumber = Random.Range (MinHeight, Maxheight);

for (int i = 0; i < randomNumber; i++)

{

GameObject box = GameObject.CreatePrimitive(PrimitiveType.Cube);

box.transform.position = new Vector3(iCounter \* HorizontalSpacing, i \* VerticalSpacing, j\*HorizontalSpacing);

}

}

iCounter--;

}

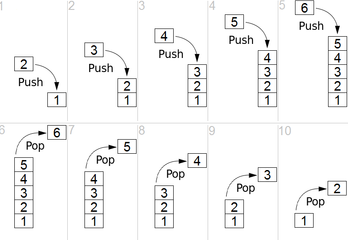
}

}

}

1. What is Stack? How do we use Stack in C#?

Stack<T>: LIFO (last in first out)



Sample C# code:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoApplication

{

class Program

{

static void Main(string[] args)

{

Stack st = new Stack();

st.Push(1);

st.Push(2);

st.Push(3);

st.Pop();

foreach (Object obj in st)

{

Console.WriteLine(obj);

}

Console.ReadKey();

}

}

}

How do we use Stack to track the path / cell visited?

Work with your partner, write a simple C# script to generate a 10 x 10 3D maze. You can use primitive type cube, or import some game assets such as walls or bricks, turn them into prefab and use them to generate a maze.

***Challenges:***

1. How do we place a player controlled character in the maze?
2. What are the different ways to place the camera(s)?
3. In the maze, how can we place some AI controlled characters and let them do some maze solving?

***Project #4: Maze Runner [Due: 2 weeks, Dec. 10th]***

1. Your game needs to random generate a perfect maze.
2. A player controlled character needs to navigate through the maze collecting treasures and defeating monsters.
3. At least some of the monsters need to have AI maze solving and pathing finding algorithms implemented.
4. Creative, funny, no violence!
5. Holiday themes welcome!

You can either use this game as a part of your midyear project or create a separate project for the midterm.