BBM405 - Fundamentals of Artificial Intelligence
Spring 2021

Homework 1

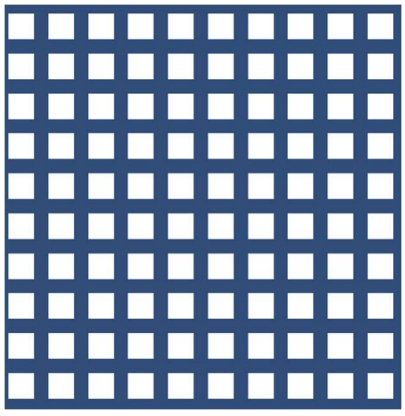
Mustafa Candan 21627059

Introduction

For this homework as said in the pdf we could use any languages. To see the maze and solutions visually I didn't know how to do it in Python or other languages so I decided to do this homework in Unity. Unity is a game engine and game developement program, so visuality would be possible.

My approach with using Unity, creating Object Orianted Enviroment for the maze. This was my main mistake with this project, because trying to create ten thousand and even one million objects was not efficient but i couldn't see this mistake in the beginning.

Before Part 1, which is generating own maze using randomized DFS, i created base design for my maze. As can be seen in below image, this white squares are maze Cells. All of the Cells are created as an Object with class of Cell. You can see the Cell Clas below.



```
Unity Script | 11 references
public class Cell : MonoBehaviour
{
    public bool mazeGenerateVisited = false;
    public List<Vector2> neighbours;
    public List<Vector2> mazeneighbours;
}
```

This Maze Base is generated automatically with a written script which takes arguments of width and depth as can be seen below. As I mentioned before, because of this method searches objects and adds neighbours to that objects it makes the function less efficient, I could've tried a diffirent approach which could be using dictionaries.

```
1 reference
public void CreateMaze(int x, int y)
{
    for(int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            GameObject vertex = Instantiate(vertexPrefab, new Vector3(i, -j, 0), Quaternion.identity, vertexParent);
            vertex.name = i.ToString() +" " + j.ToString();

            if (i - 1 >= 0 )
            {
                vertex.GetComponent<Cell>().neighbours.Add(new Vector2(i - 1, j));
            }
            if (i + 1 <= x-1)
            {
                vertex.GetComponent<Cell>().neighbours.Add(new Vector2(i + 1, j ));
            }
            if (j - 1 >= 0)
            {
                vertex.GetComponent<Cell>().neighbours.Add(new Vector2(i , j - 1));
            }
            if (j + 1 <= y-1)
            {
                vertex.GetComponent<Cell>().neighbours.Add(new Vector2(i , j + 1));
            }
        }
    }
}
```

Connections (Edges)between Cells which I called Connections are Object as well which has Connection Class as can be seen below. This Class holds the information of which cells this edge connected.

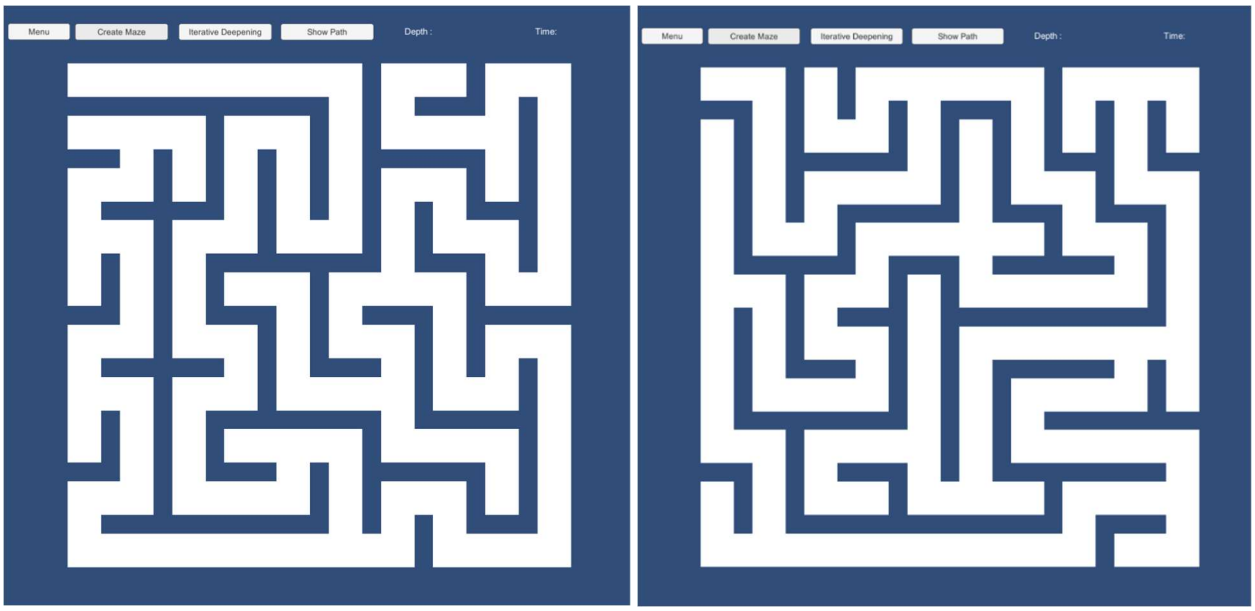
```
Unity Script | 2 references
public class Connection : MonoBehaviour
{
    public Vector2 parent1;
    public Vector2 parent2;
}
```

Part 1 – Generating my own maze using Randomized DFS

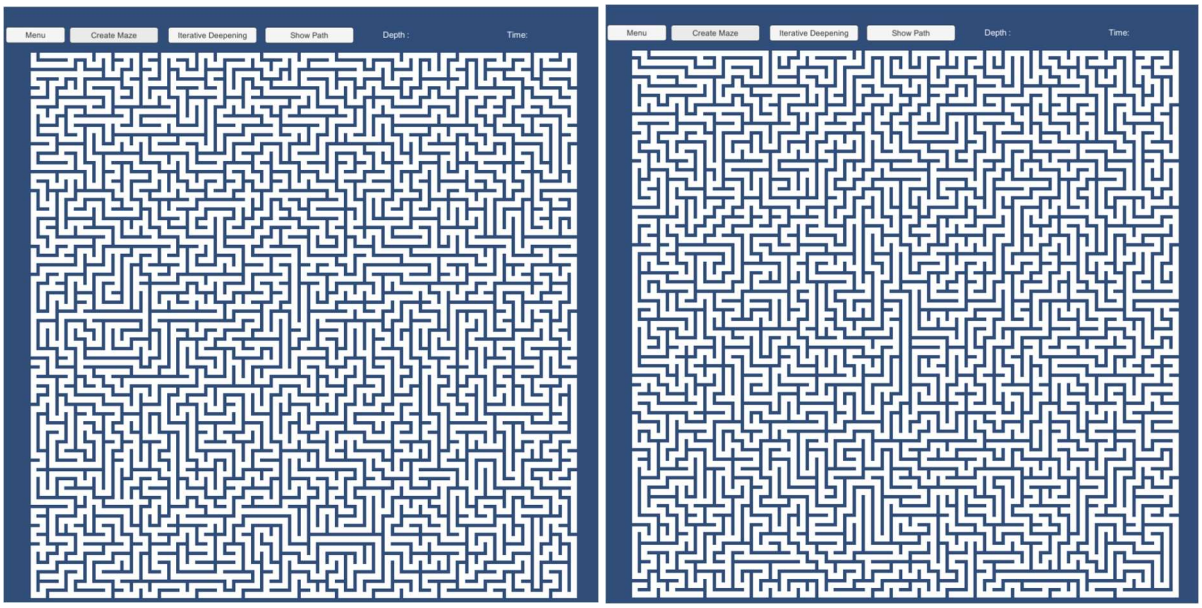
For this part I used the outline algorithm from the Homework pdf and I adapted that to my code. Because of the approach I used, this creation uses Objects as well so generating maze bigger than 500 width and 500 height will tire the computer. You can see my Maze Create function below.

```
0 references
public void CreateMaze()
{
    Vector2 startVertex = new Vector2(0, 0);
    RandomizedDFS(startVertex);
}
2 references
public void RandomizedDFS(Vector2 vertex)
{
    visited.Add(vertex);

    shuffle(GameObject.Find(vertex.x.ToString() + " " + vertex.y.ToString()).GetComponent<Cell>().neighbours);

    foreach (var nextVertex in GameObject.Find(vertex.x.ToString() + " " + vertex.y.ToString()).GetComponent<Cell>().neighbours)
    {
        if(visited.Contains(nextVertex))
        {

        }
        else
        {
            ConnectCells(vertex, nextVertex);

            RandomizedDFS(nextVertex);
        }

    }
```

Maze 10 x 10

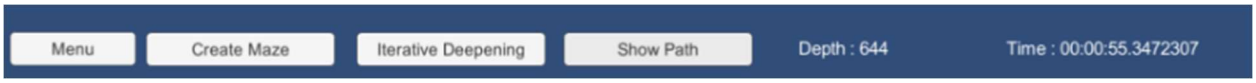


Maze 59 x 59



Part 2 – Application of search strategies

For this part with my project I could do only Iterative Deepening Search. After generating maze you can call this search function with button in UI as seen below. After search ends you can see de depth and the time passed in UI. If you push Show path button the path search took will be shown in green.

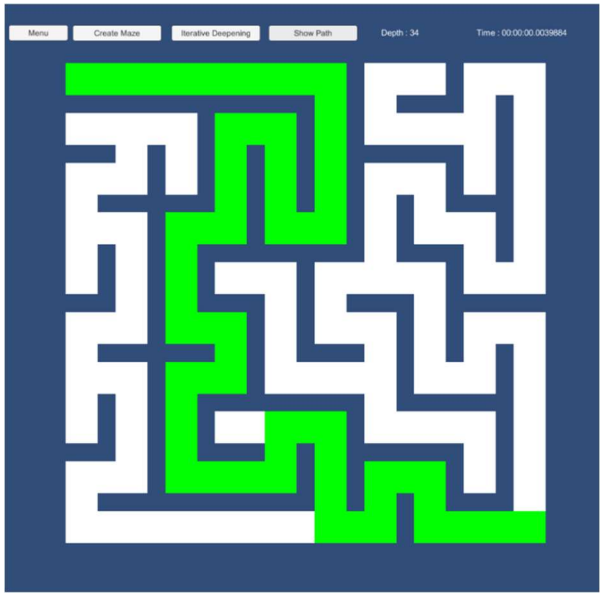


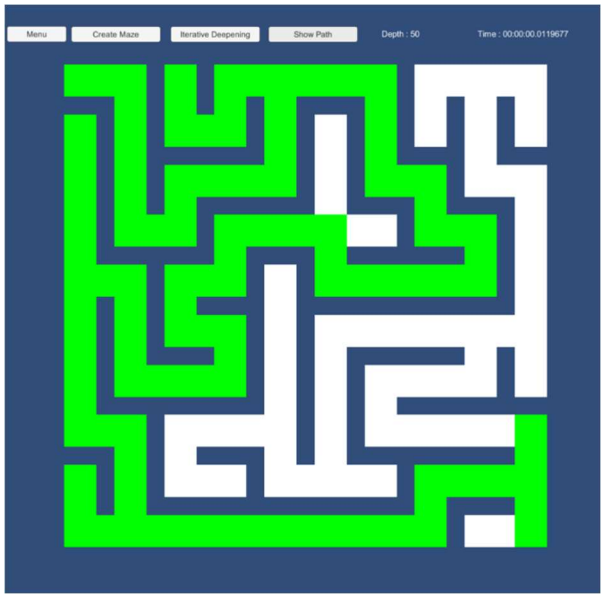
You can see the code for Iterative Deepening Search below. You can find references for this code in the last page.

```
public bool IDDFS(Vector2 src, Vector2 target, int depth)
{
    for(int i = 0; i< depth; i++)
    {
        IDFPath.Clear();
        IDFPath2.Clear();
        //Debug.Log("IDDFS depth is " + depth + " i = " + i);
        IDFvisited = new List<Vector2>();
        if (DLS(src, target, i) == true)
        {
            DepthText.text = "Depth : " + i;
            return true;
        }
    }
    return false;
}
```

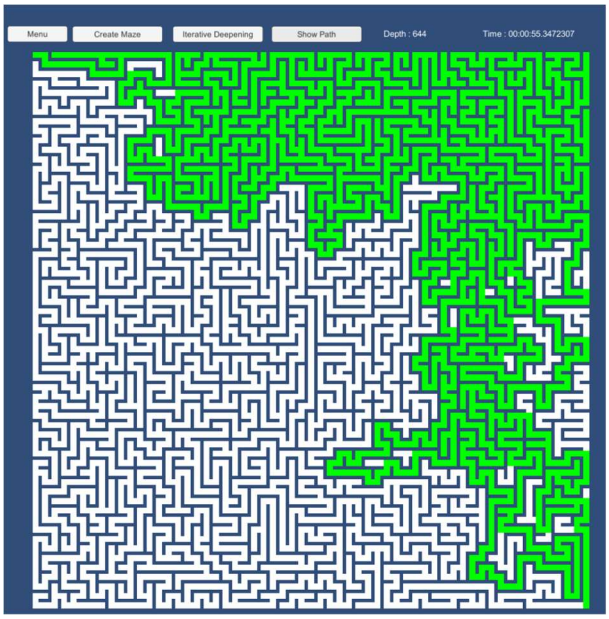
```
2 references
public bool DLS(Vector2 src, Vector2 target, int depth)
{

    IDFPath.Add(src);
    if (src.x == target.x && src.y == target.y)
    {
        Debug.Log("Girdim");
        endtime = System.DateTime.Now;
        System.TimeSpan timeSpan = endtime.Subtract(starttime);
        FinishTime = timeSpan;
        TimeText.text = "Time : " + timeSpan.ToString();
        Debug.Log(timeSpan);
        return true;
    }
    if (depth <= 0)
    {return false;}

    IDFvisited.Add(src);

    foreach (var nextVertex in GameObject.Find(src.x.ToString() + " " + src.y.ToString()).GetComponent<Cell>().mazeneighbours)
    {
        if (IDFvisited.Contains(nextVertex))
        {
        }
        else
        {
            IDFPath.Add(src);
            IDFPath2.Add(GameObject.Find(src.x.ToString() + " " + src.y.ToString() + " - " + nextVertex.x.ToString() + " " + nextVertex.y.ToString()).gameObject);
            if (DLS(nextVertex, target, depth - 1))
            {
                return true;
            }
        }
    }
        return false;
}
```

Iterative Deepening Search for Maze 10x10





Iterative Deepening Search for Maze 59 x 59



Part 3 – Analysis of the search strategies

For this part I could do analysis for only Iterative Deepening Search. If we look the criterias specified:
When we start with the first criteria I could say generally the length of the path for 59 x 59 maze it is between 600 and 900. But the longest I saw in 100 x 100 maze was 1644 which took 20 minutes of searching.

The second criteria was hard to store information because when we try to collect information in Unity it would take 7-8 GB of memory for 59 x 59 and at the end it would crash the program.

The third criteria is, max time, instantly for 10 x 10 maze and 3 minute for 59 x 59 maze. For 100 x 100 with my test I saw 20 minutes but in some of my test it went more than 30 minutes and because after passing that time it crashed the program.

Part 4 – Extending the limits

For this part I couldn't even complete first requirements which inculudes the 1000 x 1000 maze. With my wrong approach I couldn't generate more than 500 x 500 mazes. For Iterative Deepening Search algorithm, which is a algorithm that has most time and storage comparted to the other algorithms, my limit was seen in 100 x 100 mazes. If the maze generated is too complex and has the depth more than 2000 program will not respond and crash.

References

Maze Generation pseudo code :

Homework Pdf, https://www.baeldung.com/cs/maze-generation

Iterative Deepening Search Code References:

Homework Pdf, https://github.com/chitholian/AI-Search-Algorithms

https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/