CSC 3315, Assignment 1

Pathfinding using different search algorithms and heuristic strategies.

Pr. Tajjedine Rachidi

Ziad El Ismaili

Abdelhamid El Hand

Imad-Eddine Ouahidi

Al Akhawayn University in Ifrane

Saturday, June 11, 2022

Table of Contents

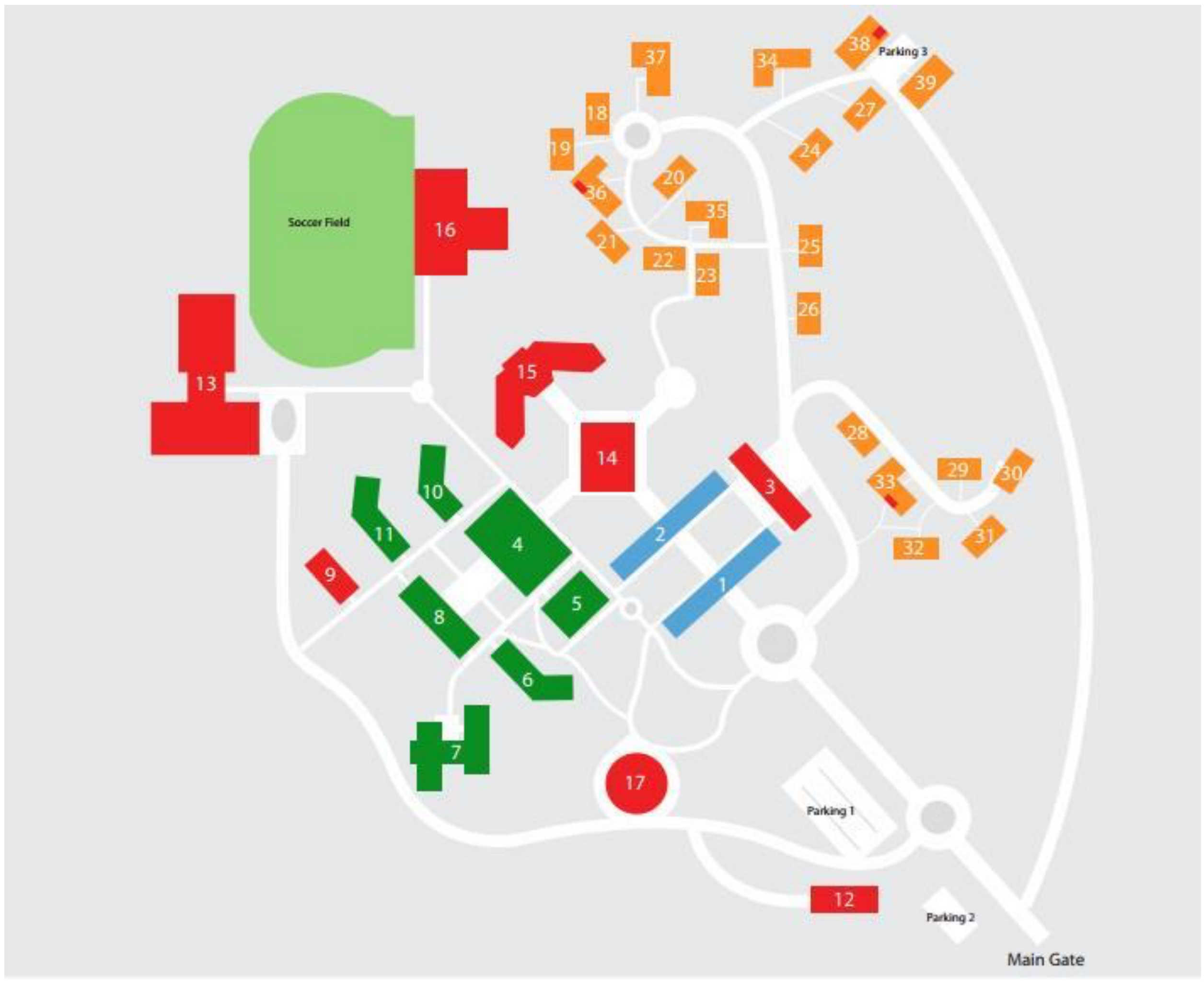
I. Introduction:

In this project, we will be implementing different pathfinding algorithms, such as Depth-First Search *(DFS)*, Breath-First Search *(BFS)*, Uniform-Cost Search *(UCS)* and A* pathfinding algorithms using the Euclidian and Manhattan heuristics. For each strategy, we will be displaying the different data concerning the time, fringe, expanded nodes, etc. We will be using **Unity software,** to draw and structure the external environment, and implementing our code in C# Sharp scripts.

This project is inspired and based on Sebastian Lague's work. You can find the YouTube videos as well as the GitHub code of his work in the references part of the report *(Last page)*.

II. Environment description:

Our team decided in this project to build a logical agent. Therefore, we decided to schematize Al Akhawayn University's campus, using Unity in 2D. Our program will be for instance help you take the shortest from any location on campus to your target. In our project, and for analysis purposes, we will be testing the program starting from anywhere at AUI's campus. The following is a drawn picture of AUI's campus:



Administrative Area

1	**North Wing:** Business Office, Office of the Vice President for Academic Affairs, Office of the Vice President for Finance & Administration, Board Room **South Wing:** Enrollment Services, President's Office, Institute for Economic Analysis & Prospective Studies
2	**North Wing:** Office of the Vice President for Student Affairs, Campus Store, Bookstore, Human Resources Office, Cafeteria **South Wing:** Department of Development & Communication, Event Space

Academic Area

4	Auditorium, Language Center, Trading Room, Center for Learning Technologies, Al Akhawayn Incubator
5, 6, 7	School of Science & Engineering
8	School of Humanities & Social Sciences
10	Center for Academic Development, Office of International Programs
11	School of Business Administration

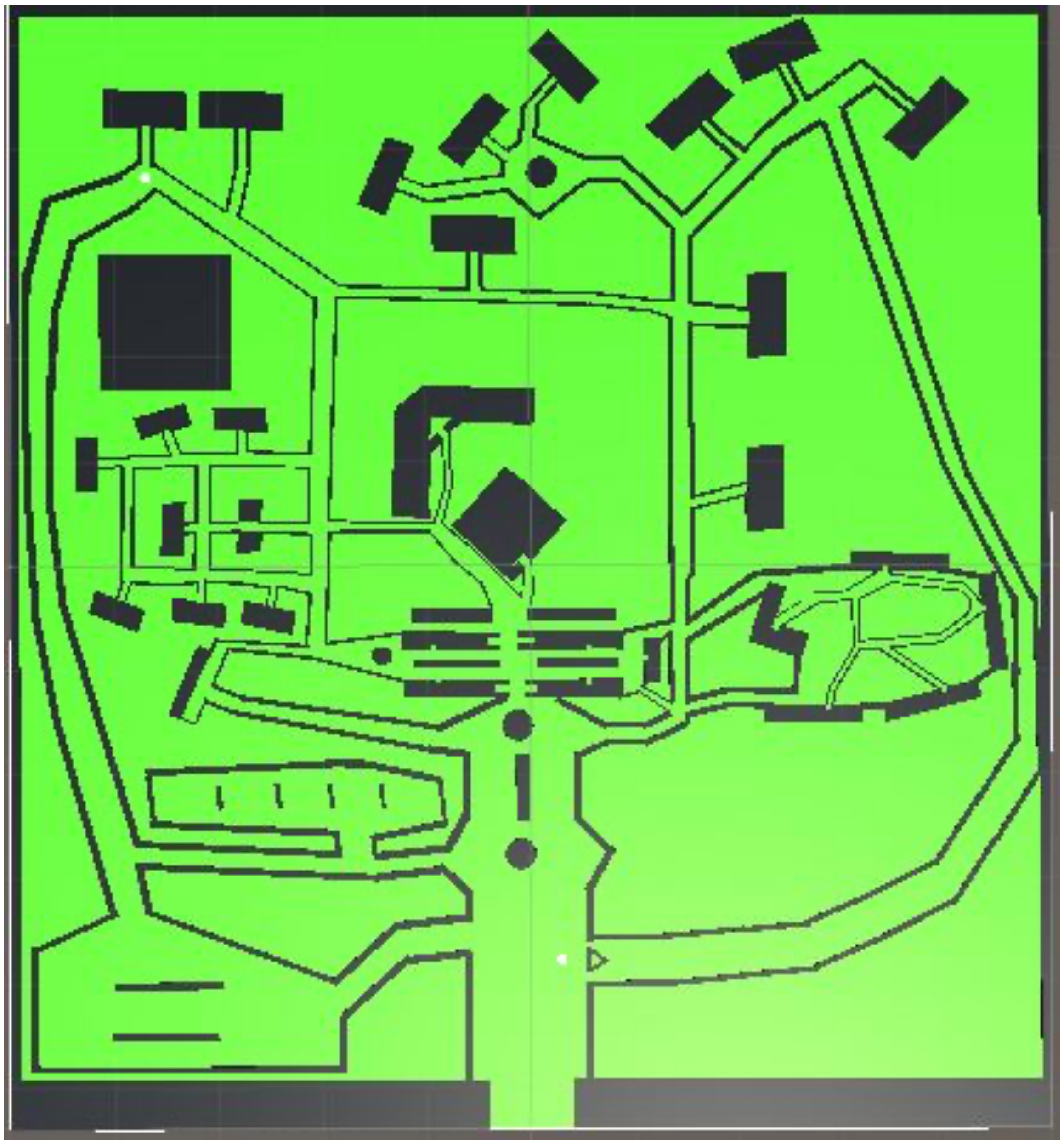
Services & Facilities

3	Restaurants
9	Information Technology Services, Security, Copy Center
12	Grounds & Maintenance, Purchasing Services
13	Swimming Pool, Gymnasium
14	Mosque
15	Mohammed VI Library
16	Student Center, Club House, Student Activities Office Athletics Office, SGA, Community Service Office
17	Grand Auditorium
26	Health Center, Housing Services
33	Post Office/ATM
36, 38	Laundry

Residential Area

18, 24, 25, 26, 27 28, 33, 34, 35, 37	Female Dorms
19, 20, 21, 22 23, 30, 32, 36	Male Dorms
28, 29, 31	Faculty Residences
38	**Left Wing(38A):** Female Dorms **Right Wing (38C):** Male Dorms

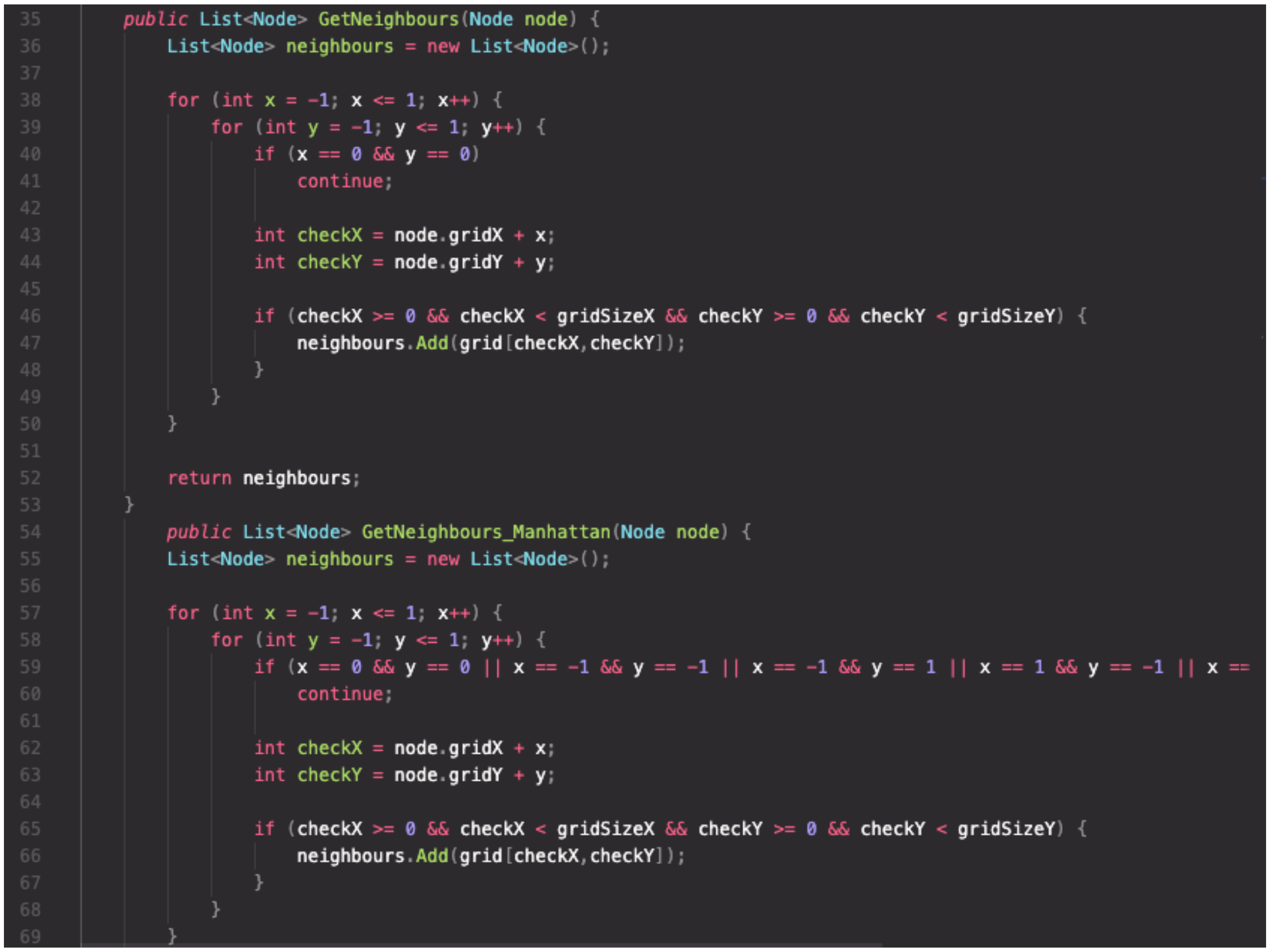
Thus, we tried to create the following environment using Unity, representing AUI's campus:



III. Scripts description:

After designing the complex environment, we started working on the scripts and the implementation of each algorithm. The following are all the C# sharp scripts with their description *(You can find the code in the Assets/Scripts folder)*:

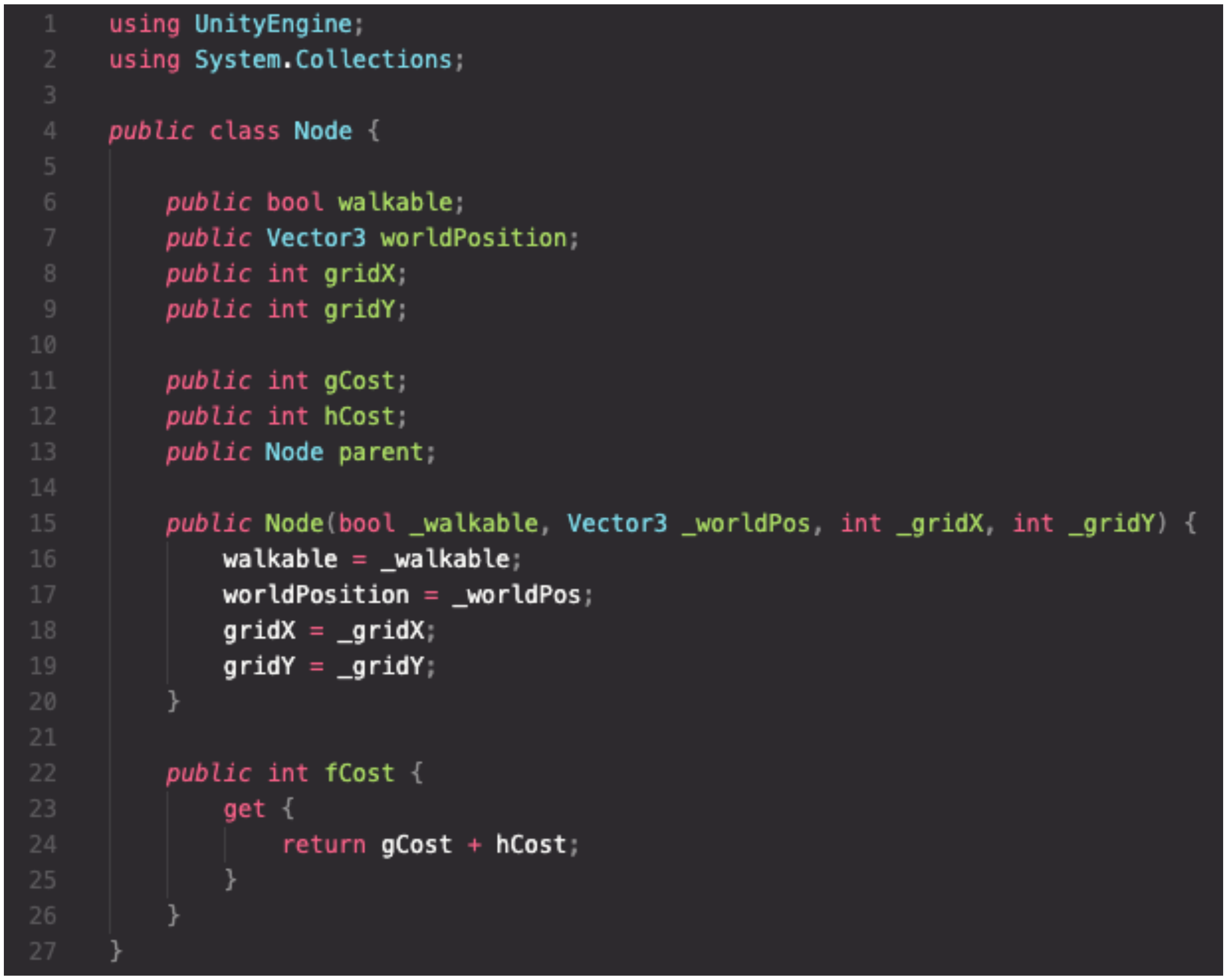
- ***Grid.cs,*** inspired by Sebastian Lague Grid.cs code, it is used to create a virtual grid which will contain nodes. The grid has three parameters, X, Y and the node radius.



```
public List<Node> GetNeighbours(Node node) {
    List<Node> neighbours = new List<Node>();

    for (int x = -1; x <= 1; x++) {
        for (int y = -1; y <= 1; y++) {
            if (x == 0 && y == 0)
                continue;

            int checkX = node.gridX + x;
            int checkY = node.gridY + y;

            if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY) {
                neighbours.Add(grid[checkX,checkY]);
            }
        }
    }

    return neighbours;
}
    public List<Node> GetNeighbours_Manhattan(Node node) {
    List<Node> neighbours = new List<Node>();

    for (int x = -1; x <= 1; x++) {
        for (int y = -1; y <= 1; y++) {
            if (x == 0 && y == 0 || x == -1 && y == -1 || x == -1 && y == 1 || x == 1 && y == -1 || x ==
                continue;

            int checkX = node.gridX + x;
            int checkY = node.gridY + y;

            if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY) {
                neighbours.Add(grid[checkX,checkY]);
            }
        }
    }
```

The Grid.cs also contains 2 functions, which deal with the way the neighbors nodes should be checked. GetNeighbors() will scan all the 8 neighbors, while GetNeighbors_Manhattan() will scan only 4 nodes: the ones on the X and Y axis.

- ***Node.cs,*** where each node will have as parameters a boolean value, to determine if the node is walkable or not, coordinates in terms of the X and Y axis, as well as a function which will calculate the fCost based on the gCost and Hcost when it comes to finding a path using A* algorithms.

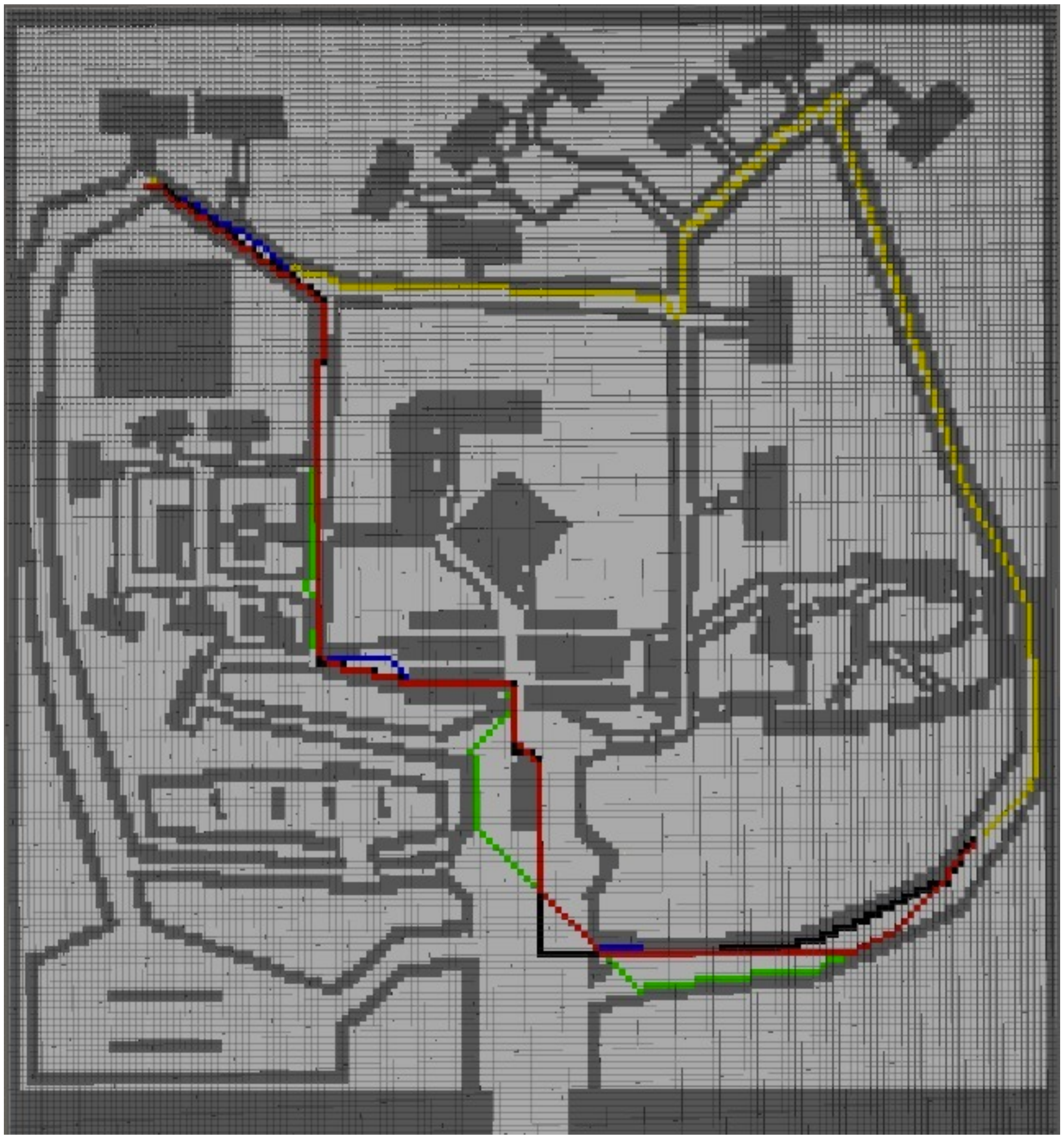


```
using UnityEngine;
using System.Collections;

public class Node {

    public bool walkable;
    public Vector3 worldPosition;
    public int gridX;
    public int gridY;

    public int gCost;
    public int hCost;
    public Node parent;

    public Node(bool _walkable, Vector3 _worldPos, int _gridX, int _gridY) {
        walkable = _walkable;
        worldPosition = _worldPos;
        gridX = _gridX;
        gridY = _gridY;
    }

    public int fCost {
        get {
            return gCost + hCost;
        }
    }
}
```

- ***DFS.cs, BFS.cs, UCS.cs, A_Star_Euclidian.cs, A_Star_Manhattan.cs,*** are the scripts where each algorithm is implemented.

IV. Testing and comparison:

In this part, we will be running simultaneously all the scripts, which means that the different algorithms will be tested according to the following criteria: ***time, expanded nodes (at target goal), fringe (at target goal), completeness, optimality…*** For this, the starting node will be the seeker



Green: BFS Algorithm.

Black: A* strategy with Manhattan.

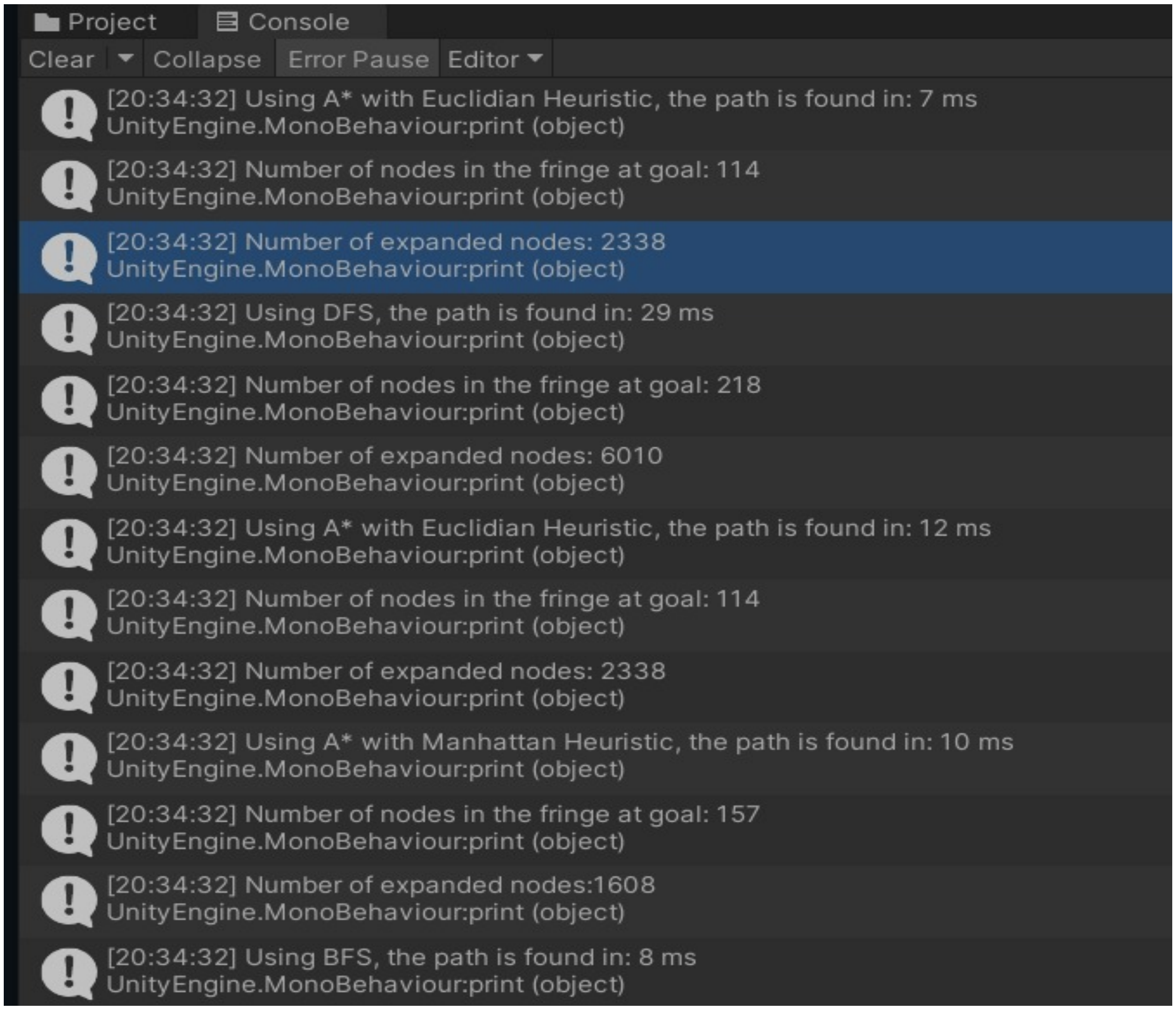
Yellow: DFS Algorithm.

Red: Unified Cost Strategy.

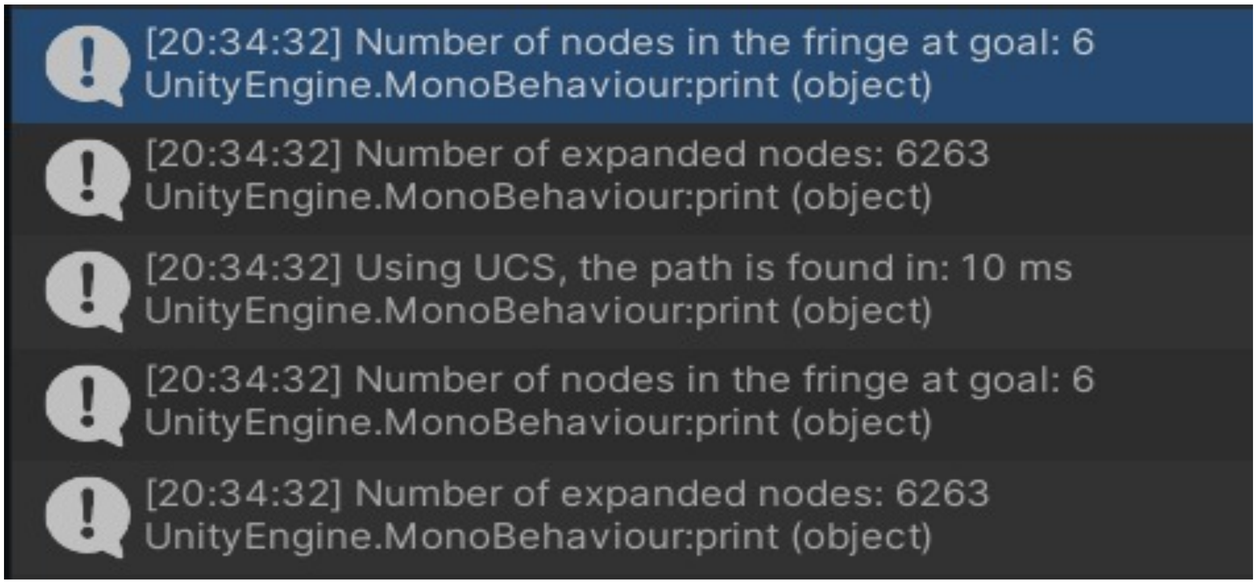
Blue: A* strategy with Euclidean distance

We decided to put between AUI’s main gate and the road of buildings 38/39, and the final target is located near the gymnasium.

According to the console output, the following are the data collected after running the program:



```
[20:34:32] Using A* with Euclidian Heuristic, the path is found in: 7 ms
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of nodes in the fringe at goal: 114
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes: 2338
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Using DFS, the path is found in: 29 ms
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of nodes in the fringe at goal: 218
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes: 6010
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Using A* with Euclidian Heuristic, the path is found in: 12 ms
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of nodes in the fringe at goal: 114
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes: 2338
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Using A* with Manhattan Heuristic, the path is found in: 10 ms
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of nodes in the fringe at goal: 157
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes:1608
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Using BFS, the path is found in: 8 ms
UnityEngine.MonoBehaviour:print (object)
```



```
[20:34:32] Number of nodes in the fringe at goal: 6
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes: 6263
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Using UCS, the path is found in: 10 ms
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of nodes in the fringe at goal: 6
UnityEngine.MonoBehaviour:print (object)
[20:34:32] Number of expanded nodes: 6263
UnityEngine.MonoBehaviour:print (object)
```

Putting the output into a table, we will obtain the following:

Algorithm	*Time*	*Expanded nodes*	*Fringe*	*Completeness*	*Optimality*
DFS	***29 ms***	***6010***	***218***	***Yes***	***No***
BFS	***8 ms***	***6263***	***6***	***Yes***	***Yes***
UCS	***10 ms***	***6263***	***6***	***Yes***	***Yes***
A*_Euclidian	***7 ms***	***2338***	***114***	***Yes***	***Yes***
A*_Manhattan	***10 ms***	***1608***	***157***	***Yes***	***Yes***

In this section, we will be comparing between the 6 algorithms.

<u>Comparison in terms of time:</u>

The best performing algorithm in terms of time in this case is A*_Euclidian, which can be explained by the fact that it uses an admissible distance heuristic to estimate the distance from its location to the goal. In this case, the advantage that A*_Euclidian has over A*_Manhathan is that the first one takes diagonal steps that the second cannot take which reduces the time needed to reach the goal. Also, BFS in this case performed very well since it has O(b^s) resulting in 8s of running time. The slowest one in our case is DFS; it wasted time expending the nodes until reaching the end of the branch before moving to the next one.

<u>Comparison in terms of expanded nodes:</u>

A*_Manhattan algorithm needed the least number of nodes expanded to reach the goal since it only takes in the neighbors on the X and Y axis which reduces the number of the nodes needed to be expended. Also, the other A* algorithm has a reduced number of expended nodes too, but still higher than the first one due to the diagonal neighbors. Both strategies have a low number of expanded nodes thanks to the admissible heuristic that they are following, since they are part of informed search. The other uninformed search algorithms take all the neighbors and expands them until they reach the goal, which explains the high numbers of nodes.

Comparison in terms of fringe:

When it comes to the fringe, the smallest fringe in this case is the one of UCS and BFS, with only 6 nodes yet to be explored. Knowing that each strategy expanded around 6000 node each, we can see the number of nodes that were checked before finding the goal is considerable. Additionally, DFS again here has the highest number of nodes yet to be explored since it expands the neighbors and goes deeper and deeper then comes back to the top to start digging into a new branch of the tree leaving many nodes unchecked yet. A* strategies have an average of 130 nodes yet to be explored, but only expanded around 2 thousand nodes in the first place, which means that they did not need to expand a lot of nodes to reach the goal.

Conclusion:

Based on the results of our test case, we can see that the best strategy in terms of time complexity, fringe, expanded nodes... all combined is A*_Euclidian. Meaning that if you want to find the shortest path leading to the goal in short time and with average memory it is perfect for you.

On the other hand, DFS is the least preferred strategy since it gets to the goal but not with an optimal path guaranteed and with a considerable time and space consumption. Other algorithms are better or worst depending on the case. For example, if you don’t have much memory, you can use the A*_Manhattan strategy, or if you are only interested in having the path with the least cost you can go with UCS.

References:

(2022). Retrieved 12 June 2022, from https://www.youtube.com/watch?v=-L-WgKMFuhE&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW

(2022). Retrieved 12 June 2022, from https://www.youtube.com/watch?v=nhiFx28e7JY&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW&index=2

(2022). Retrieved 12 June 2022, from https://www.youtube.com/watch?v=mZfyt03LDH4&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW&index=3

GitHub - SebLague/Pathfinding. (2022). Retrieved 12 June 2022, from https://github.com/SebLague/Pathfinding