SCHOOL OF SCIENCE & ENGINEERING Spring 2023

CSC4301 01 Intro. to A I

Report: Project #1

February 16, 2023

Realized by:

Fatima El Kabir

Douae Kabelma

Supervised by:

Dr. Tajjeeddine Rachidi

Table of content

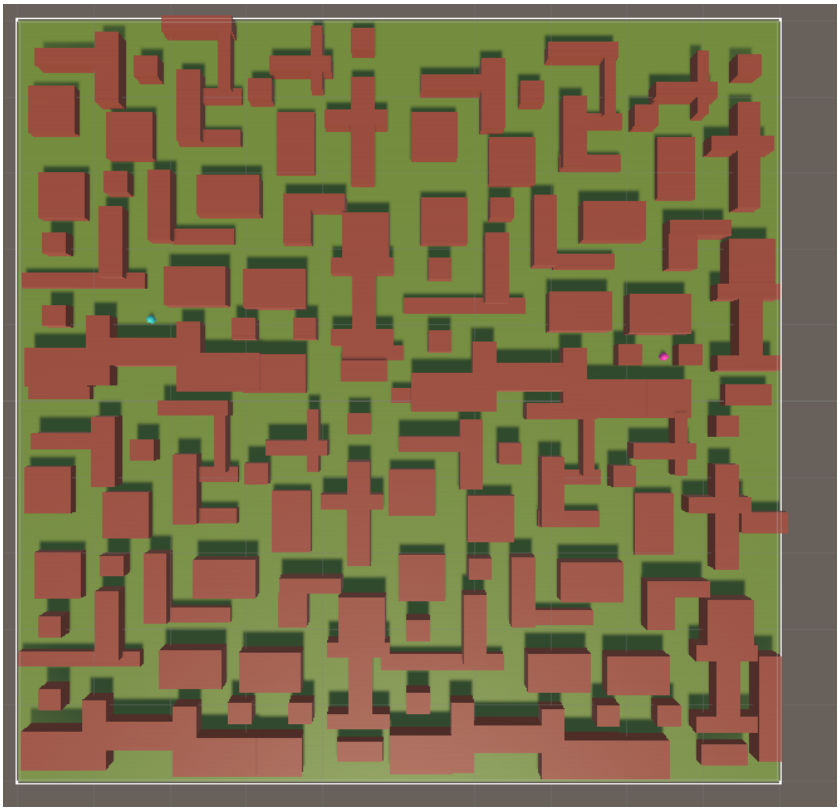
I. Introduction

This project aims to provide a comprehensive overview of pathfinding algorithms and their implementation in Unity. Pathfinding is a crucial component of many games, where characters or objects need to navigate through complex environments while avoiding obstacles and finding the most efficient routes. This project will enable us to create a pathfinding game where we evaluated each algorithm to find the shortest and most efficient path between two points in our environment. Therefore, it will help us explore and put into practice all the algorithms that we have seen in class into a real problem. By the end of this project, we will have a deeper understanding of the mechanics behind pathfinding and how to optimize our game for performance and memory usage by choosing the right algorithm.

II. The environment/ Grid

In order to implement a path finding game, we need an environment, where we can represent the obstacles and the path resulted from each algorithm. Therefore, after watching the YouTube video, we thought of making a larger gride with more obstacles to be complex enough than the one Sebastian has made.

The following screenshot displays the environment/grid we constructed. We believe that it's more sophisticated and utilizes the features of the search methods we intend to examine. As we added more obstacles, and we defined the starting and the target positions.



III. Algorithm's simulations

i. A* algorithms

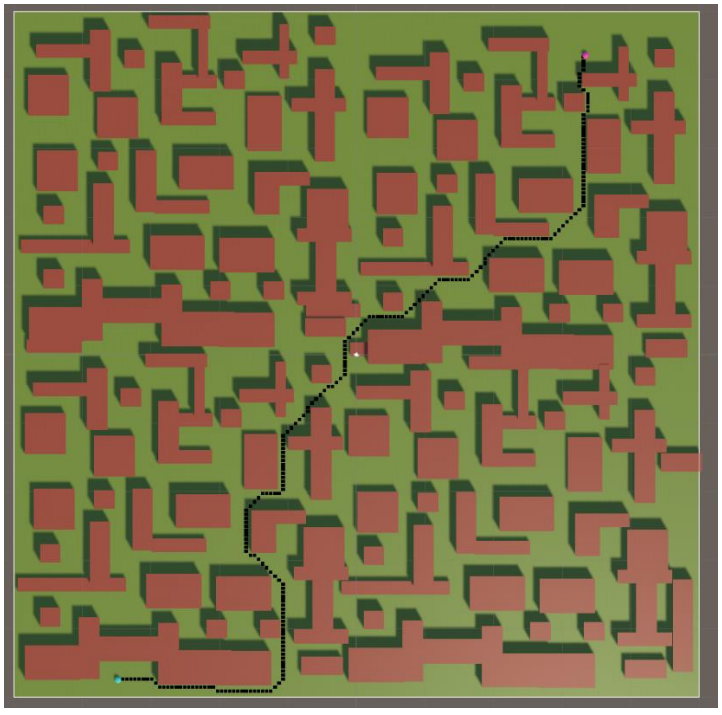
a. A Heuristic from Video:*

In this algorithm, we used the same heuristic in the Youtube video, which is the following:

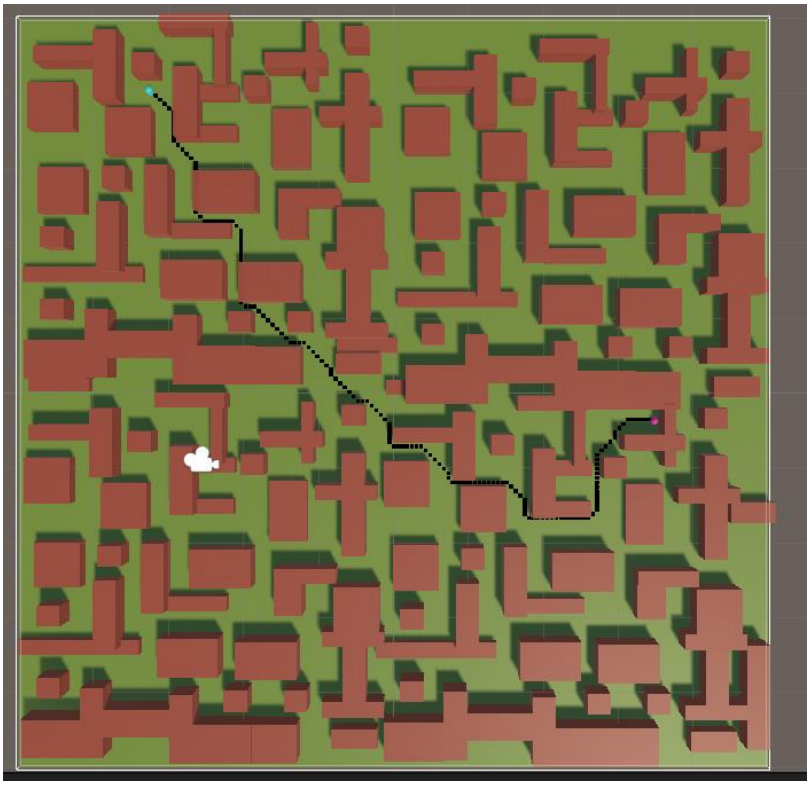
```
distance X = Abs(nodeA.gridX nodeB.gridX)
distance Y= Abs(nodeA.gridY - nodeB.gridY)
    if (distance X > distance Y)
        return 14* distance Y+ 10* (distance X - distance Y)
    else
        return 14* distance X + 10 * (distance Y- distance X)
```

- The following 3 screenshots shows the path obtained from this A* algorithm:

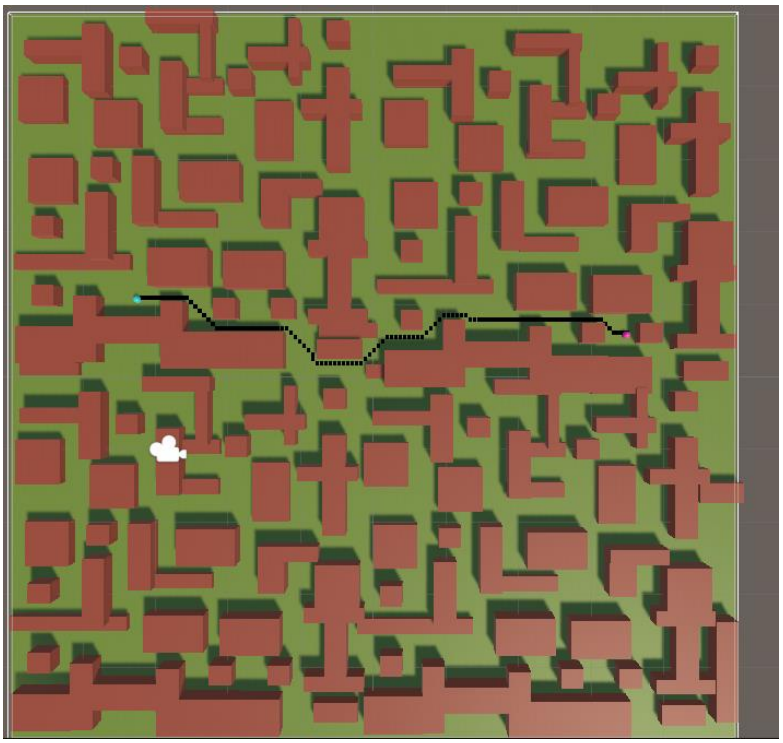
1st simulation



2nd simulation



3rd simulation



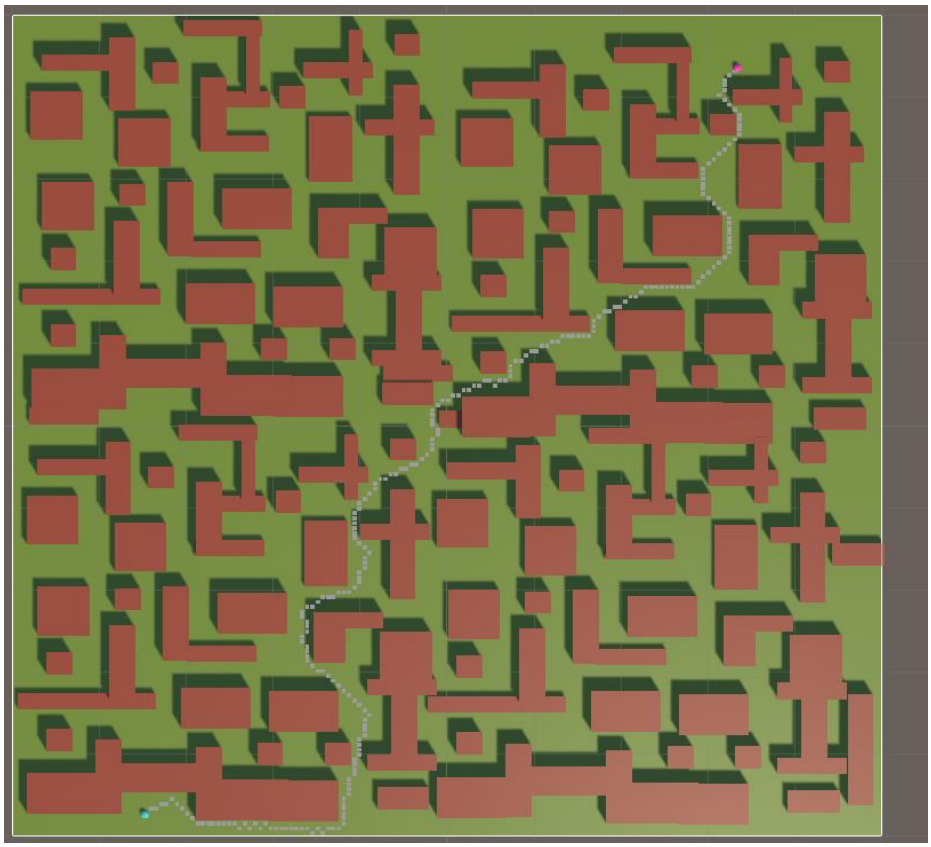
b. A Euclidian Heuristic 2*

In this algorithm, we used the Euclidian heuristic to find the distance between two nodes with the following algorithm.

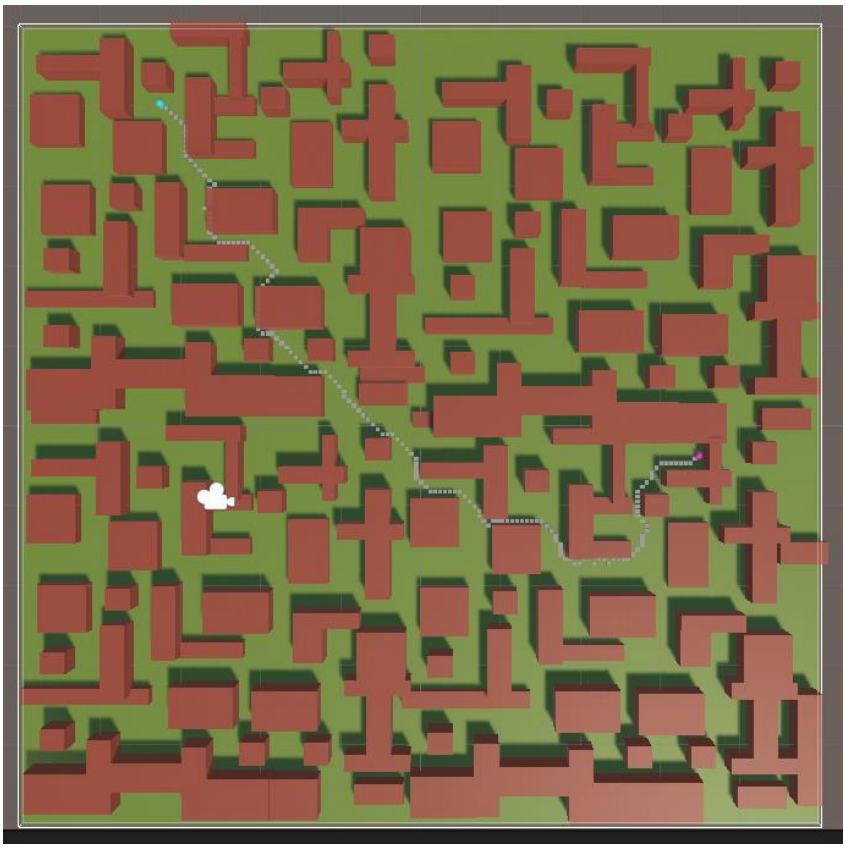
```
int distanceX = nodeA.gridX - nodeB.gridX

int distanceY = nodeA.gridY - nodeB.gridY

        return Sqrt(dstX * dstX + dstY * dstY)
```

The following screenshots shows the path obtained from this A* algorithm.

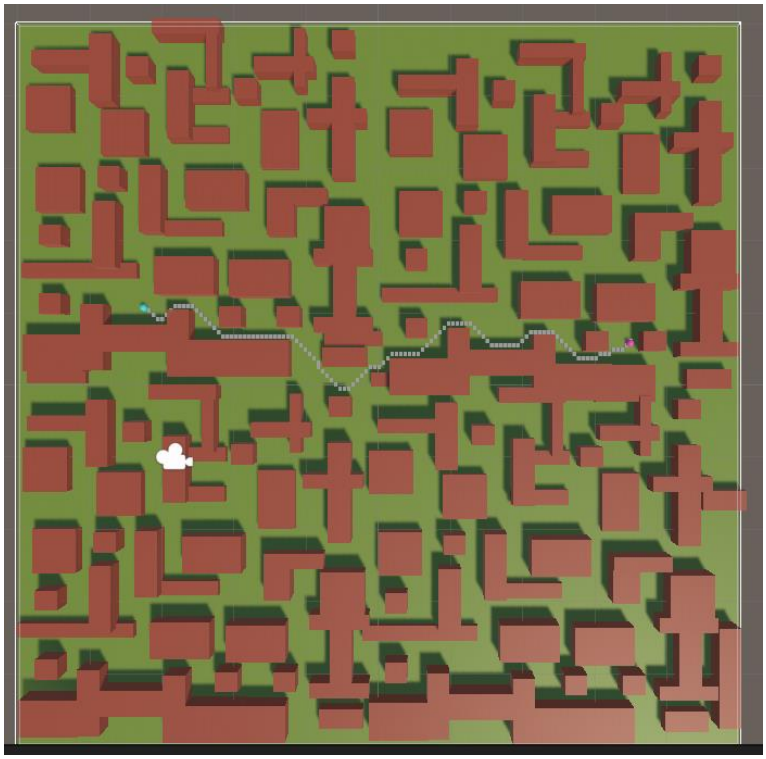
1st simulations:



2nd simulation



3rd simulation



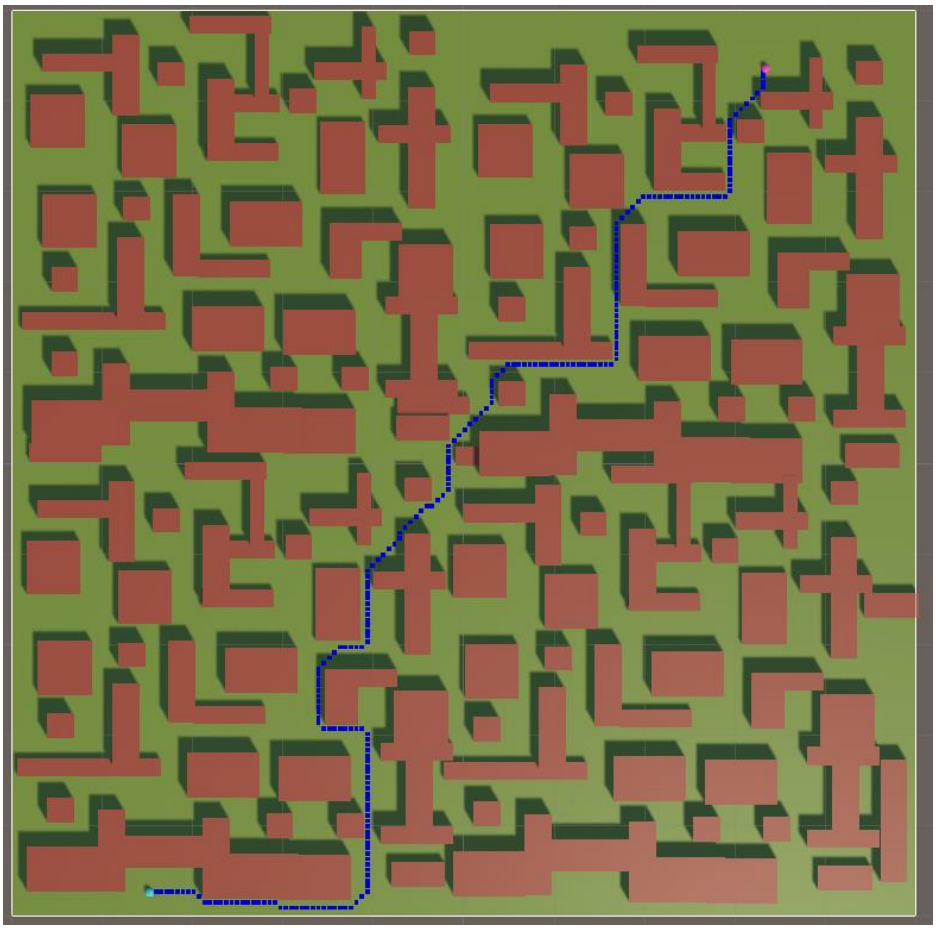
c. *A* Manhattan Heuristic*

In this algorithm, we used the Manhattan Heuristic to find the distance between two nodes with the following algorithm.

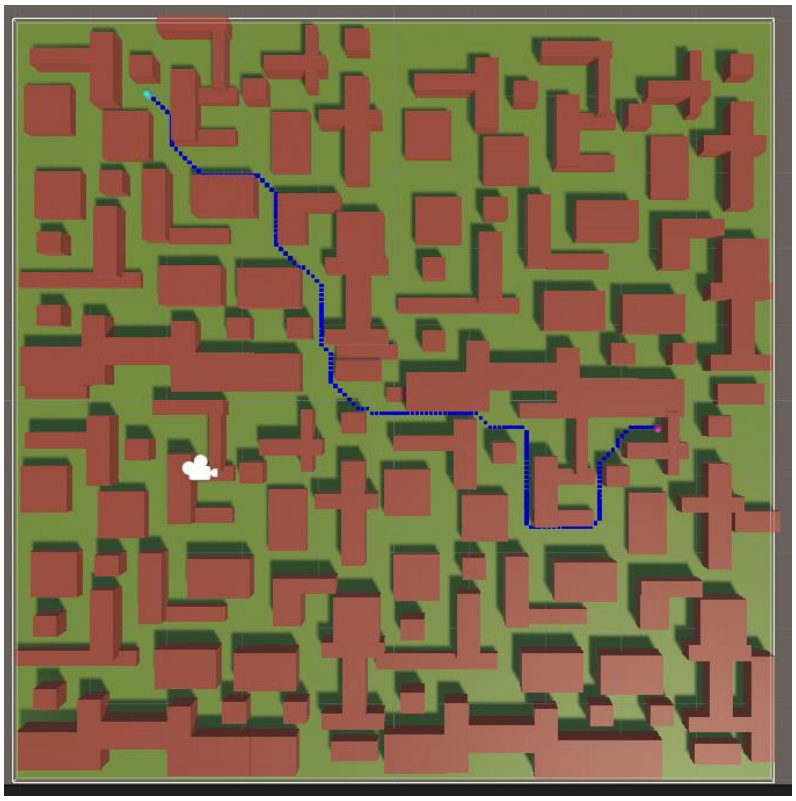
```
distanceX = nodeA.gridX - nodeB.gridX

distanceY = nodeA.gridY - nodeB.gridY

return Abs(dstX) + Abs(dstY)
```

The following screenshots shows the path obtained from this A* algorithm.

1st simulation



2nd simulation

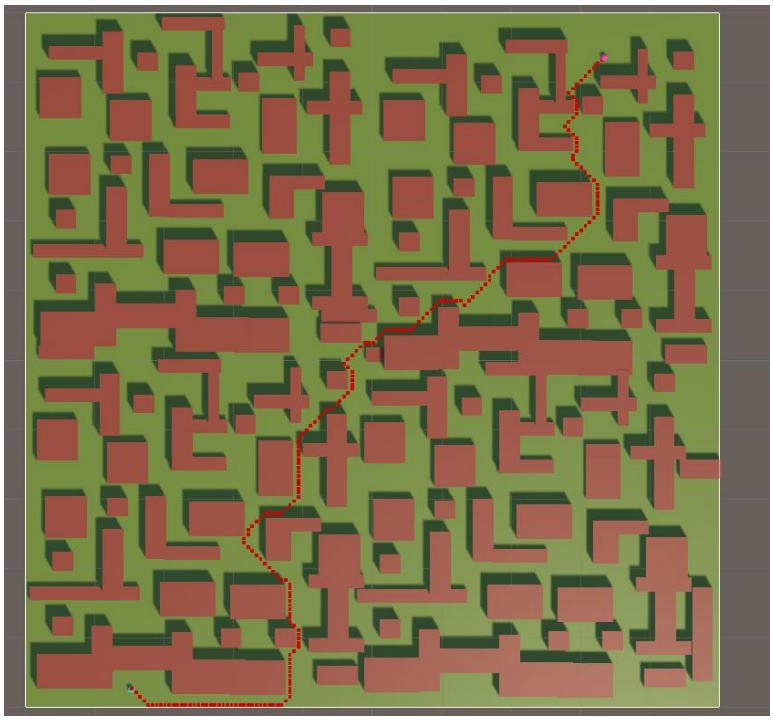


3rd simulation

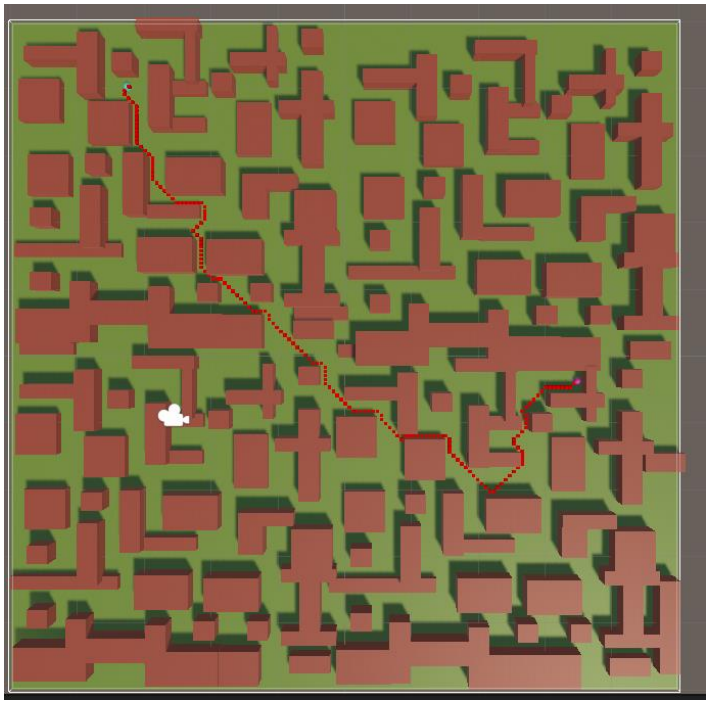
ii. UCS algorithm

The following screenshots shows the path obtained from the UCS algorithm.

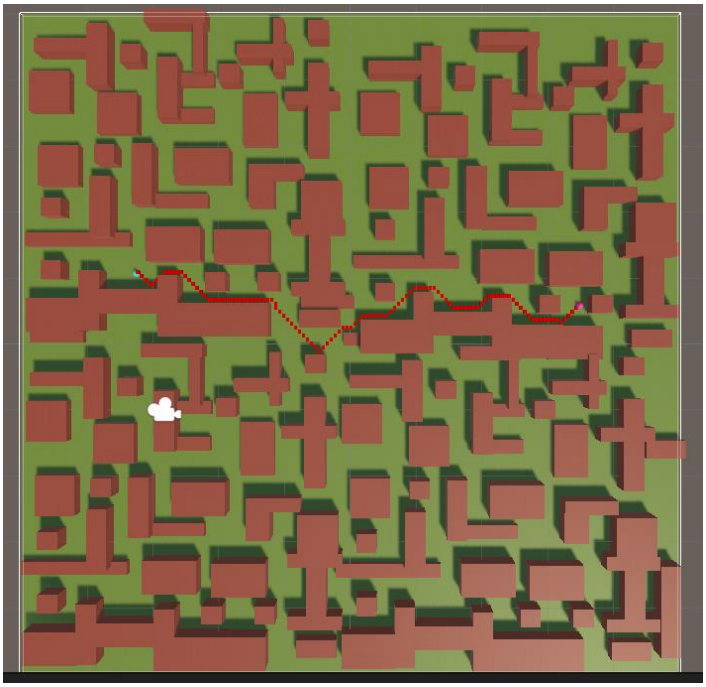
1st simulation



2nd simulation



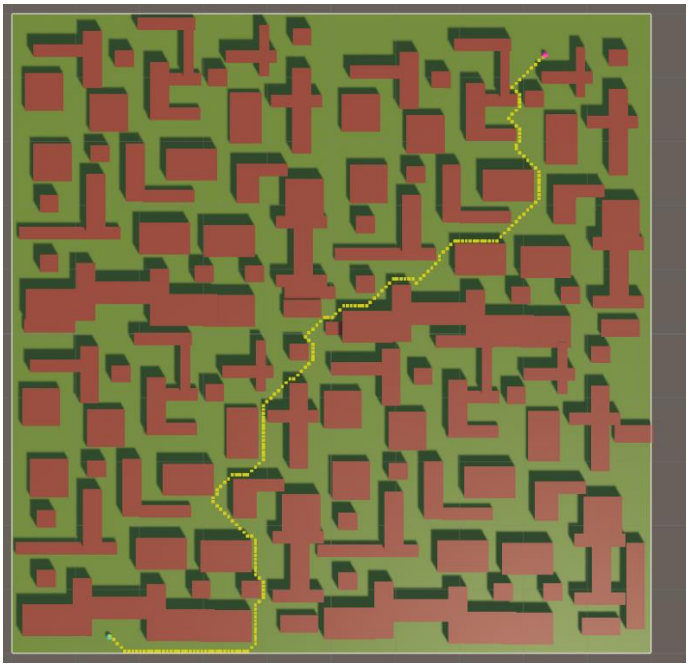
3rd simulation



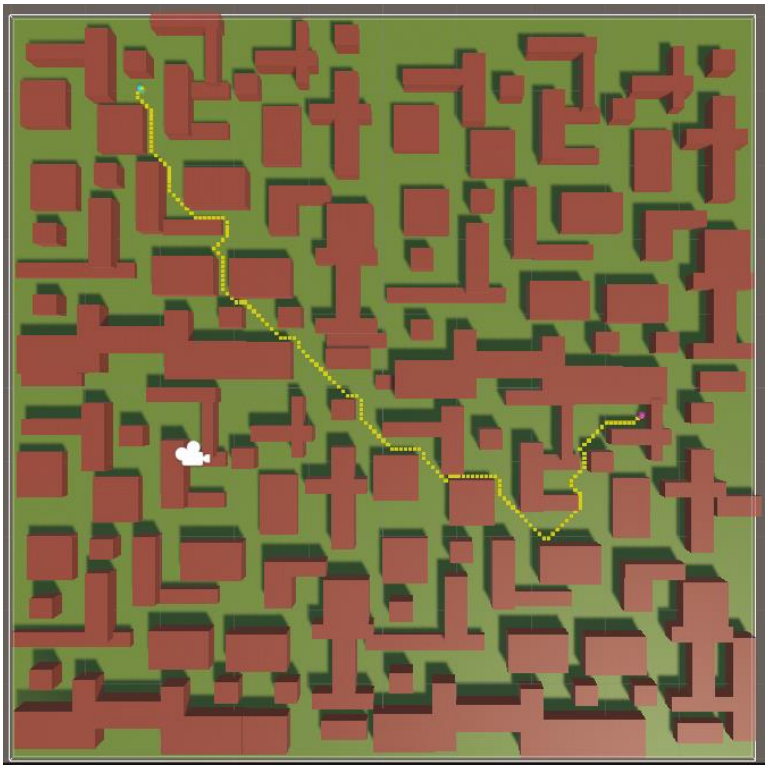
iii. BFS algorithm

The following screenshots shows the path obtained from the BFS algorithm.

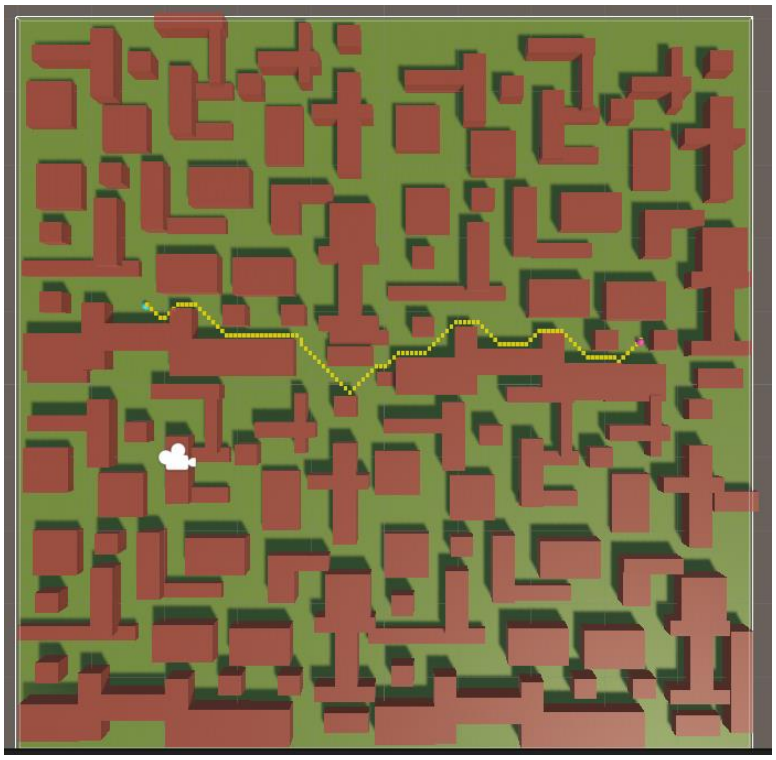
1st simulation



2nd simulation



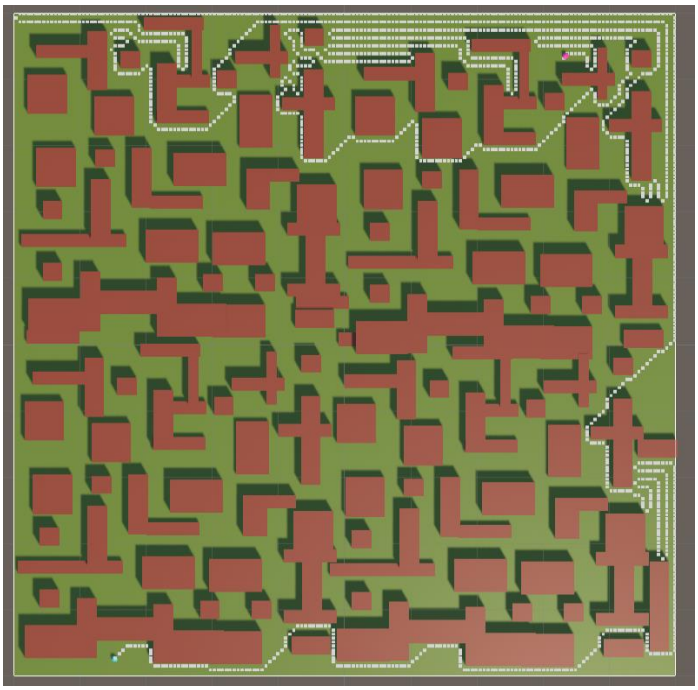
3rd simulation



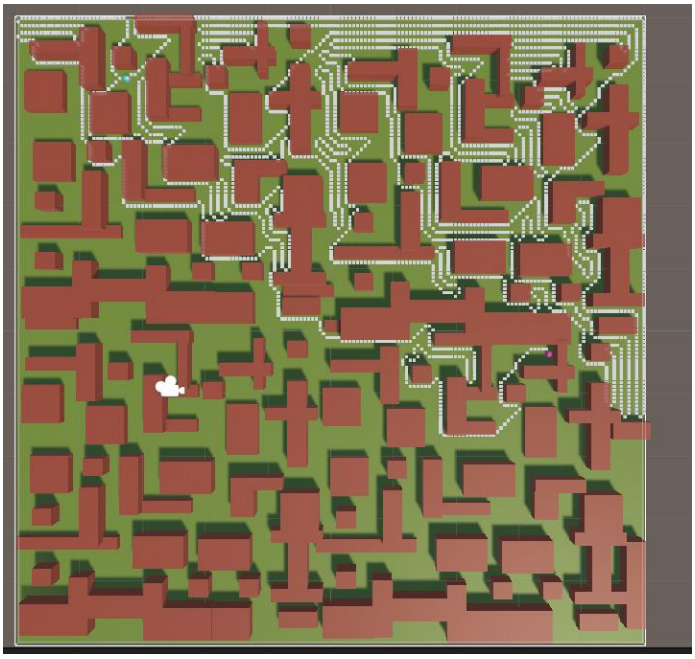
iv. DFS algorithm

The following screenshot shows the path obtained from the DFS algorithm.

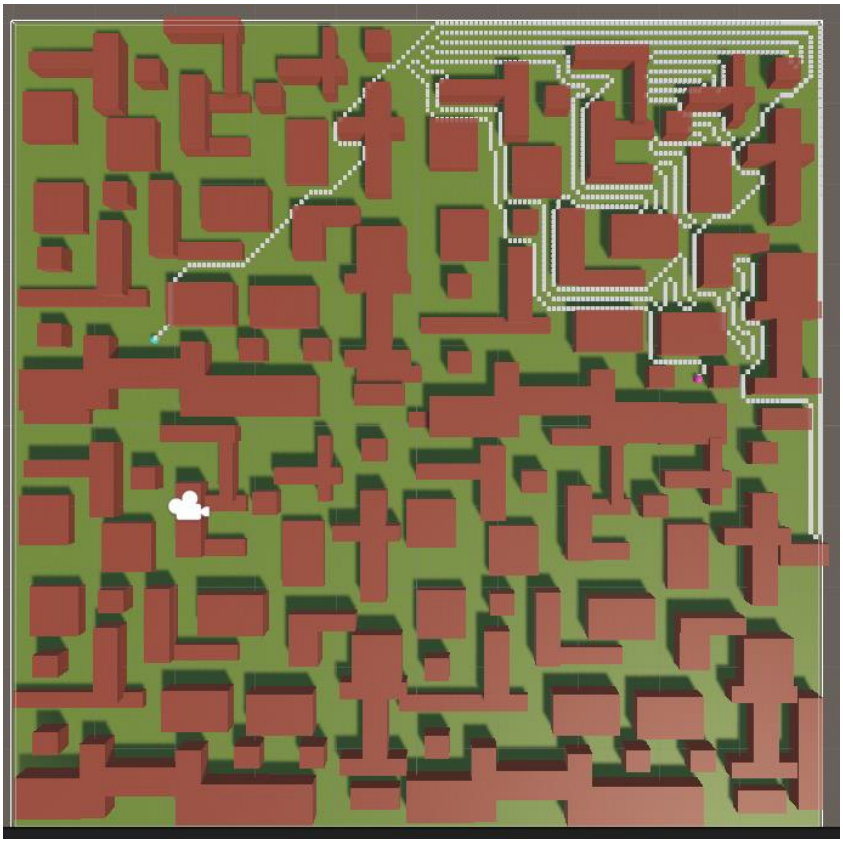
1st simulation



2nd simulation



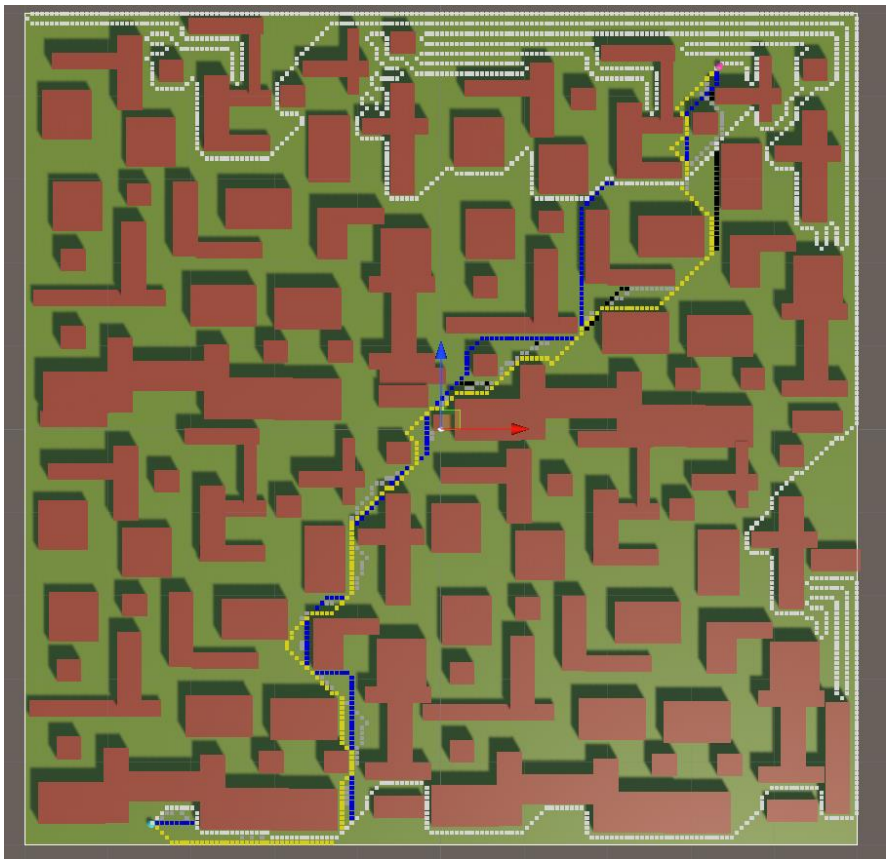
3rd simulation



IV. Performance & Solutions comparison

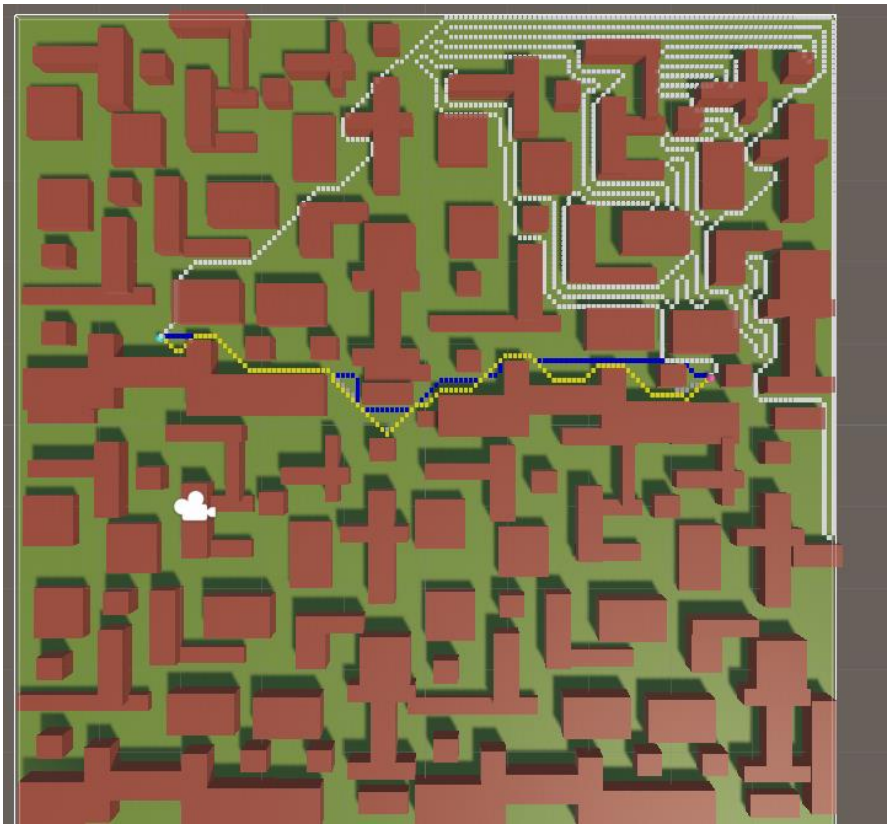
Now, we will do again the same 3 simulation that we did above, but this time lunching all the algorithms at once.

Trial #1



a. *Trial #2*

b. *Trial #3*



After conducting the above simulations, we can see that the UCS p

We created the following table to compare between the performance of the algorithms. Those information were generated by our code.

Table for the execution time

	A* from video	A* Euclidian	A* Manhattan	UCS	BFS	DFS
Simulation 1	20 ms	10 ms	8 ms	32 ms	16 ms	24 ms
Simulation 2	26 ms	23 ms	34 ms	29 ms	16 ms	77 ms
Simulation 3	6 ms	2 ms	3 ms	26 ms	14 ms	44 ms

Table for the max fringe size

	A* from video	A* Euclidian	A* Manhattan	UCS	BFS	DFS
Simulation 1	271	330	415	152	152	1700
Simulation 2	271	276	329	144	144	5051
Simulation 3	134	212	126	180	180	5063

Table for the average number of nodes expanded

	A* from video	A* Euclidian	A* Manhattan	UCS	BFS	DFS
Simulation 1	7995	4911	3355	13849	13849	13925
Simulation 2	1127	8219	12432	13170	13170	13790
Simulation 3	1267	840	1354	9784	9784	14053
Average ndes expanded	3463	4657	5714	12268	12268	13923

From the above tables we can say that:

- The Manhattan algorithm is usually faster than A* Euclidean because it's less computationally intensive. It performs well when the search space is large, and the obstacles are spread out.
- The Euclidean algorithm is computationally more expensive than A* Manhattan because it involves calculating square roots. It performs well when the search space is small, and the obstacles are clustered.
- UCS algorithm is expanding the node with the lowest cost. but can be slow when the search space is large or when there are many obstacles.

- BFS algorithm explores all the neighboring nodes first before moving on to the next level. It is guaranteed to find the shortest path, but it is somewhat faster.
- DFS algorithm explores a single path as deep as possible before backtracking. As we can see it takes a lot of time to find the path, and it may not always find the shortest path.

After seeing the execution time and the fringe size for each algorithm presented in the above table, it's evident that A* algorithms produce the most efficient and brief path in comparison to the other three search methods since it takes less time and the average number of expanded nodes is lesser. And the most efficient one is A* Manhattan heuristic.

➔ it's important to note that these findings may differ based on factors such as processing capabilities, operating system, background procedures, and the initial/final state. In general, UCS is the slowest, BFS is the fastest, A* algorithms provide the least expensive path, while DFS yields the costliest path.

V. Moving Character in the most efficient path

After implementing the different algorithms, we will create a character (ball) that will walk the efficient path. Therefore, once we get the paths of the different algorithms, we created a function that calculates the length of each path, then it compares it to the different paths that we got. And the character will follow and walk through the path that has the minimum distance.

➔ **The videos for the character moving the efficient path for each simulation is attached in the zip file.**

VI. Conclusion

All in all, creating a pathfinding game in Unity using different algorithms is a valuable experience that can improve our technical and problem-solving skills, as well as our ability to work effectively in a team. Therefore, we can say that the experience of working with those algorithms (A*, UCS , BFS, DFS) taught us the strengths and weaknesses of each algorithm and when to use them. We also learnt about data structures, such as priority queues and stacks, that

are necessary for implementing these algorithms. Thus, this project was an excellent way to gain a deeper understanding of game development and programming and can provide a foundation for future projects in this field.