

Pathfinding Algorithms: A*, BFS, DFS, UCS, and A* with a different heuristic function

1. Introduction

In this project, we implemented five pathfinding algorithms and used Unity 3D to visualize them. Given a seeker (white capsule) a target (red capsule), and some obstacles (unwalkable nodes) on a plane with a grid of dimensions 200 x 200, the objective of each algorithm is to find a path from the seeker to the target.

In all the upcoming configurations, the paths are drawn with different colors:

A* → **Black**

A* Alternative → **Cyan (light blue)**

BFS → **Dark Blue**

DFS → **Pink**

UCS → **Yellow**

In some cases, the paths may intersect and thus cover each other. Therefore, we made sure to show the path of each algorithm separately as well.

A* alternative is the same as the A* algorithm, but it uses a different heuristic which is explained in the coming section.

2. Heuristic used in A* alternative

Since an admissible heuristic must satisfy the following condition:

$$0 \leq h(n) \leq h^*(n)$$

Where $h^*(n)$ is the true cost to a nearest goal

The different heuristic we implemented for the A* algorithm considers the minimum among the greedy cost and the uniform cost to be the fCost:

```
if(gCost > hCost)
    return hCost;
return gCost;
```

This heuristic will still satisfy the condition and it is optimistic since it returns the minimum among the two costs, but unlike A*, it does not take into consideration both the gCost and the hCost concurrently.

This is how this heuristic function works: In the first half of the path, the algorithm expands the nodes in the order of how close they are to the start; In the second half of the path (i.e., as we get closer to the goal), the nodes are expanded in the order of how close to the goal they are.

Ultimately, the path is formed following the parent nodes from the goal state till the start state.

3. General comparison on a random configuration

For the sake of comparing these algorithms in terms of their performance in general cases, we executed them on many random configurations.

In this section we will discuss these algorithms using one random configuration which can be reproduced using the seeker and target positions shown below:

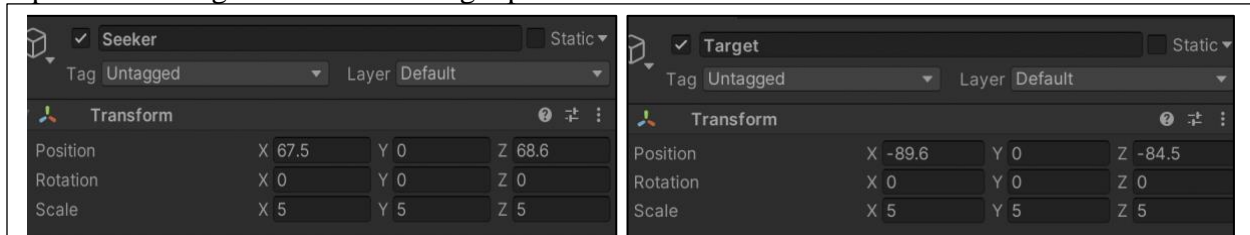


Figure 1: Configuration 1 - Seeker and Target Positions

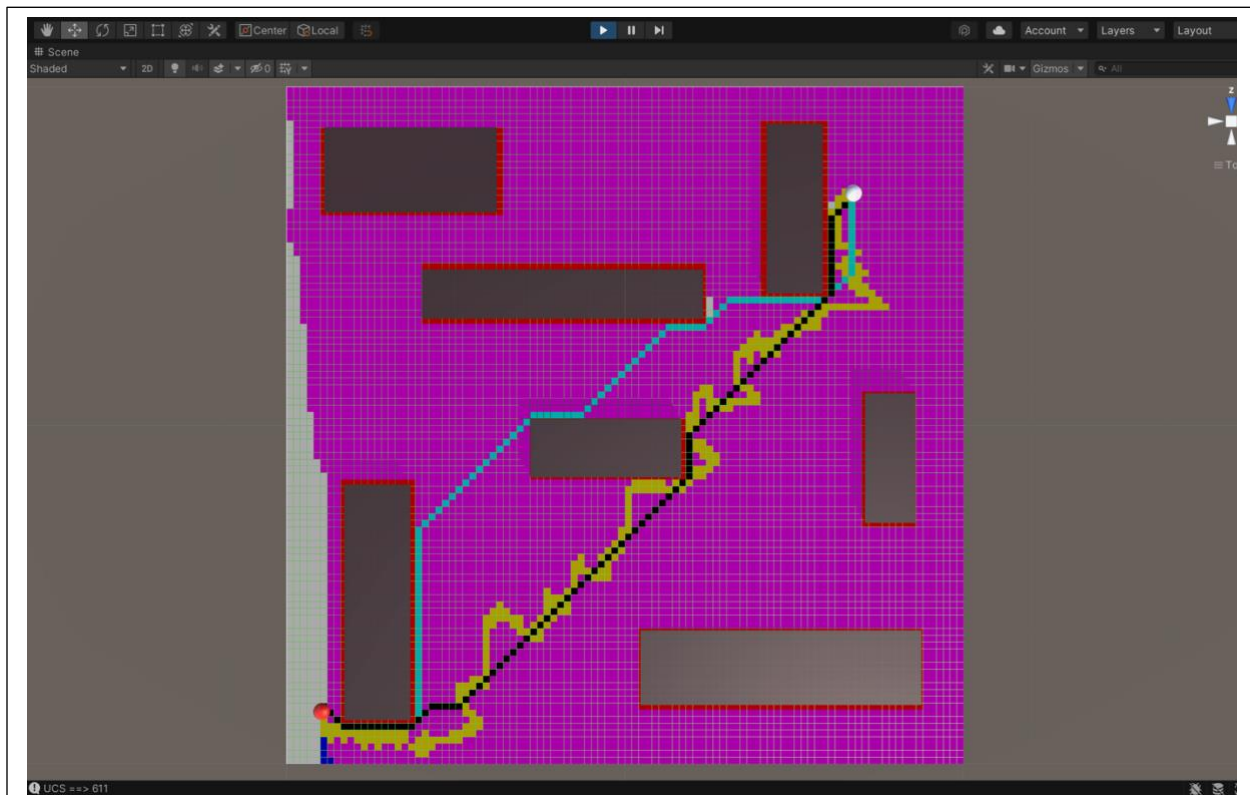


Figure 2: Random configuration 1

- Informed Search Algorithms:

As we can see from the paths shown in figure2, A* (black path) is not only complete but is also optimal; it gives the shortest path between the seeker and the target.

A* **alternative** (cyan path) seems to give a suboptimal path (close to the shortest path).

- Uninformed Search Algorithms:

UCS (Yellow path) assigns to each node a cost which is the cumulative distance from the start node. And these costs are used to prioritize the nodes in a priority queue and expand the least costly nodes first. Since we used the same cost for all directions, UCS here gives more priority to the diagonal nodes in the direction of the goal since going diagonally is always shorter than

BFS in the above configuration is partially hidden by other paths, here's the full BFS path: In this configuration, BFS shows another suboptimal path. When the cost of all nodes is the same ($= 1$) BFS gives an optimal path.



- ! [18:59:24] Time Elapsed:
UnityEngine.MonoBehaviour:print (object)
- ! [18:59:24] A* ==> 59 ms
UnityEngine.MonoBehaviour:print (object)
- ! [18:59:24] A* Alternative ==> 84 ms
UnityEngine.MonoBehaviour:print (object)
- ! [18:59:24] DFS ==> 9 ms
UnityEngine.MonoBehaviour:print (object)
- ! [18:59:24] BFS ==> 16 ms
UnityEngine.MonoBehaviour:print (object)
- ! [18:59:24] UCS ==> 38 ms
UnityEngine.MonoBehaviour:print (object)

This figure shows the time elapsed by each algorithm to find the path shown in configuration 1. Since the grid is small, the overhead of maintaining the heap in A* and A* alternative is affecting the time elapsed by these two algorithms. This overhead will decrease the bigger the grid is. Therefore, using time to compare these algorithms may not be relevant. The number of expanded nodes may be a better criterion to compare time complexities.

```
[18:59:24] Number of Expanded Nodes:
UnityEngine.MonoBehaviour:print (object)
[18:59:24] A* ==> 1632
UnityEngine.MonoBehaviour:print (object)
[18:59:24] A* Alternative ==> 3305
UnityEngine.MonoBehaviour:print (object)
[18:59:24] DFS ==> 7388
UnityEngine.MonoBehaviour:print (object)
[18:59:24] BFS ==> 7464
UnityEngine.MonoBehaviour:print (object)
[18:59:24] UCS ==> 7480
UnityEngine.MonoBehaviour:print (object)
```

We can see here that A* only expands just 1632 nodes out of the 40000 nodes of the grid. A* alternative is around double that number. On the other hand, BFS, DFS, and UCS all expand beyond 7000 nodes.

Once again, these numbers would go higher and the difference would be more significant in the case of a bigger grid which we could not implement.

```
[18:59:24] Max Fringe Size:
UnityEngine.MonoBehaviour:print (object)
[18:59:24] A* ==> 273
UnityEngine.MonoBehaviour:print (object)
[18:59:24] A* Alternative ==> 253
UnityEngine.MonoBehaviour:print (object)
[18:59:24] DFS ==> 20962
UnityEngine.MonoBehaviour:print (object)
[18:59:24] BFS ==> 536
UnityEngine.MonoBehaviour:print (object)
[18:59:24] UCS ==> 611
UnityEngine.MonoBehaviour:print (object)
```

Regarding space complexity, we measured it by keeping track of the maximum fringe length for each algorithm. As expected, DFS in this configuration has a significantly highest fringe size compared to the other algorithms. A* and A* alternative seem to perform similarly, and same goes for UCS and BFS.

4. BFS optimal performance

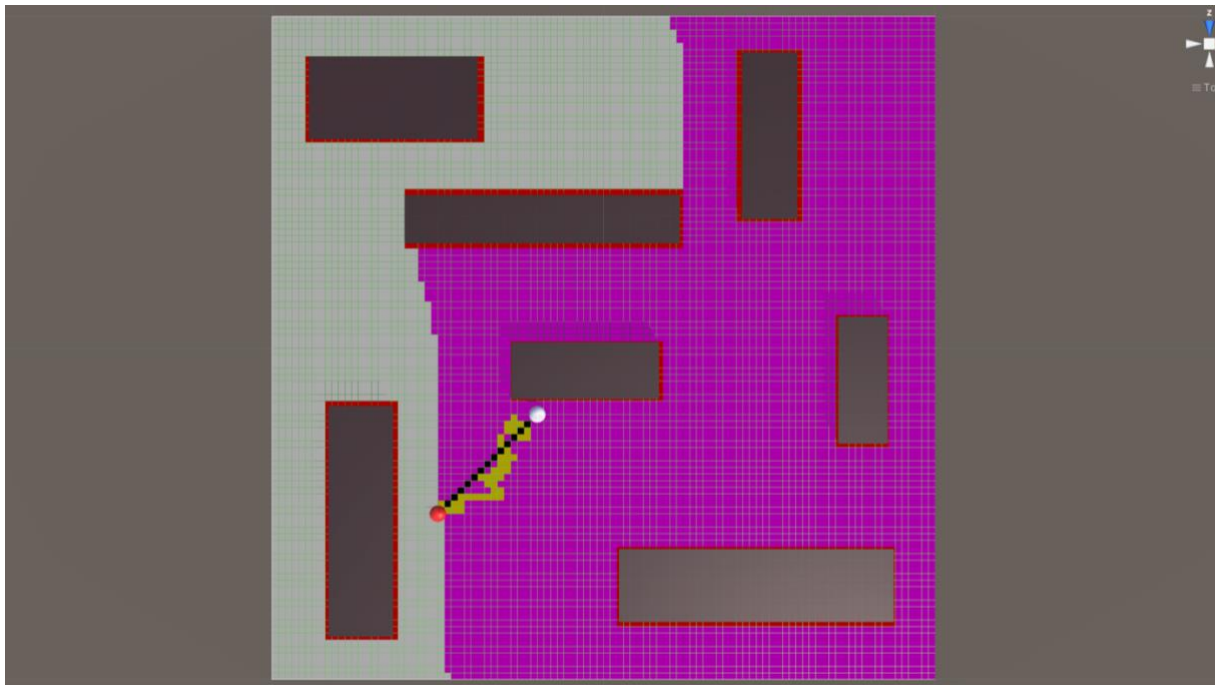


Figure 5: Configuration 2

In this configuration, all BFS, A*, and A* alternative result in the same shortest path. A* happens to cover the paths of the two other algorithms in the above screenshot, here they are:

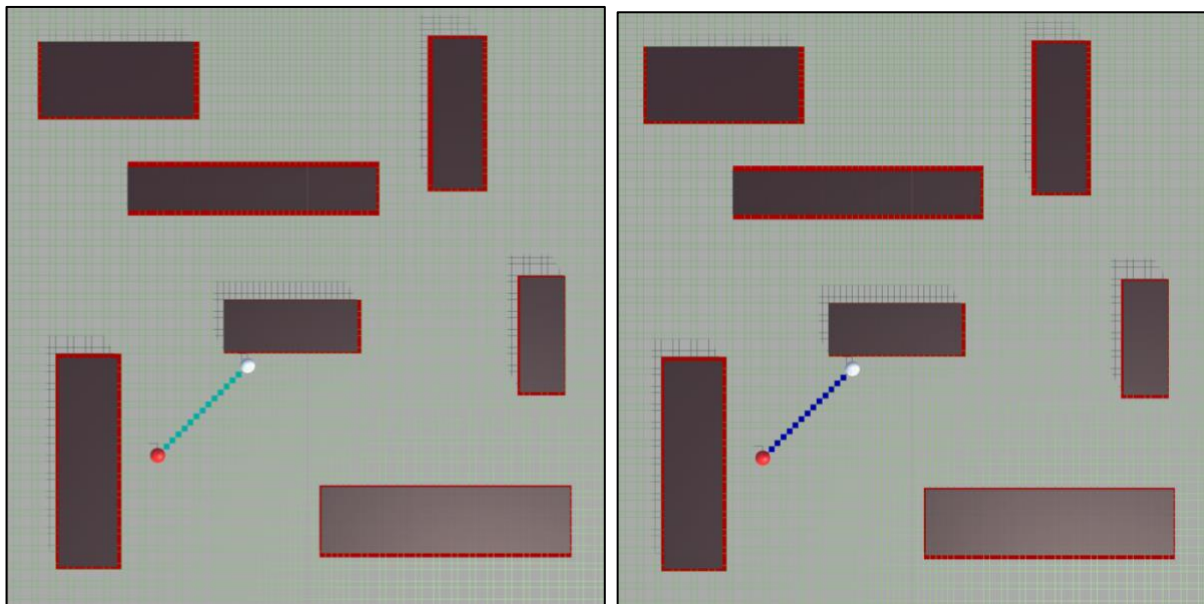


Figure 6: A* alternative and BFS Optimal Cases

To reproduce this configuration, the following seeker and target positions can be applied:

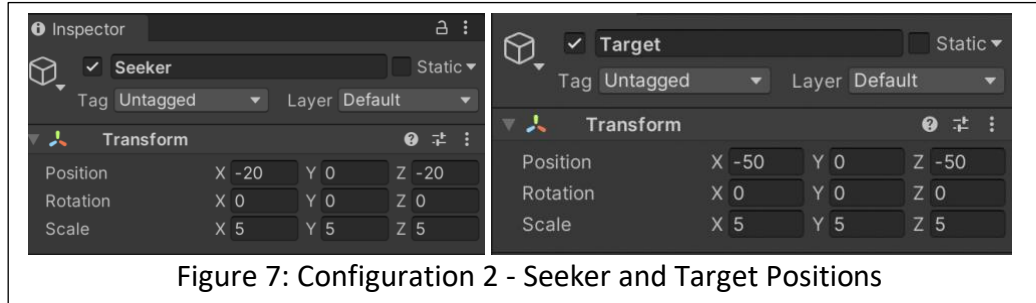


Figure 7: Configuration 2 - Seeker and Target Positions

In this case, the seeker and target were placed such that the nodes that make up the closest path between them are expanded first in both BFS and A* alternative algorithm. The target was placed diagonally to the right underneath the seeker. However, although these two algorithms lead to the shortest path in this configuration, they still waste time exploring nodes that are not promising.

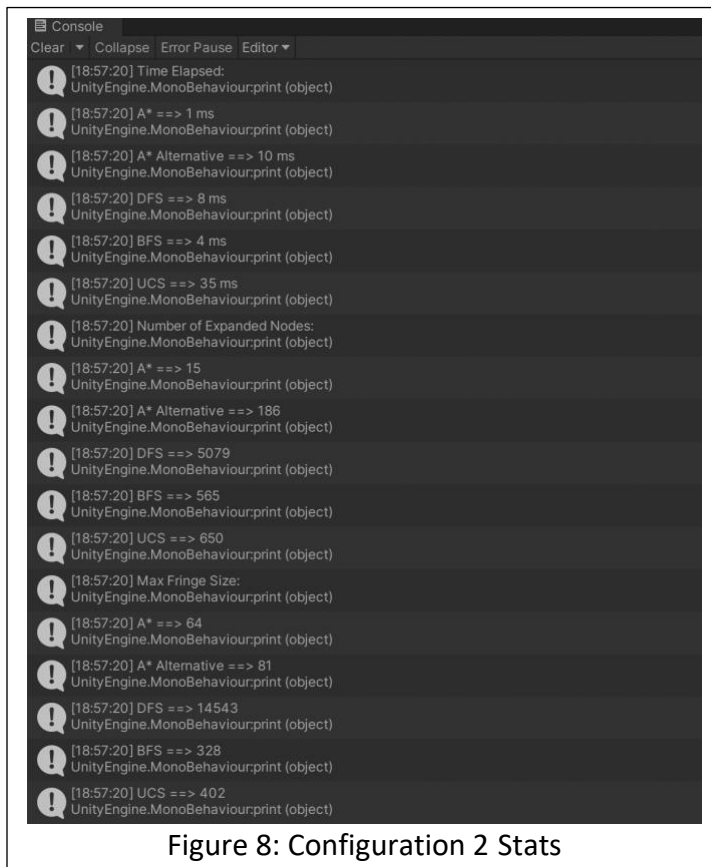
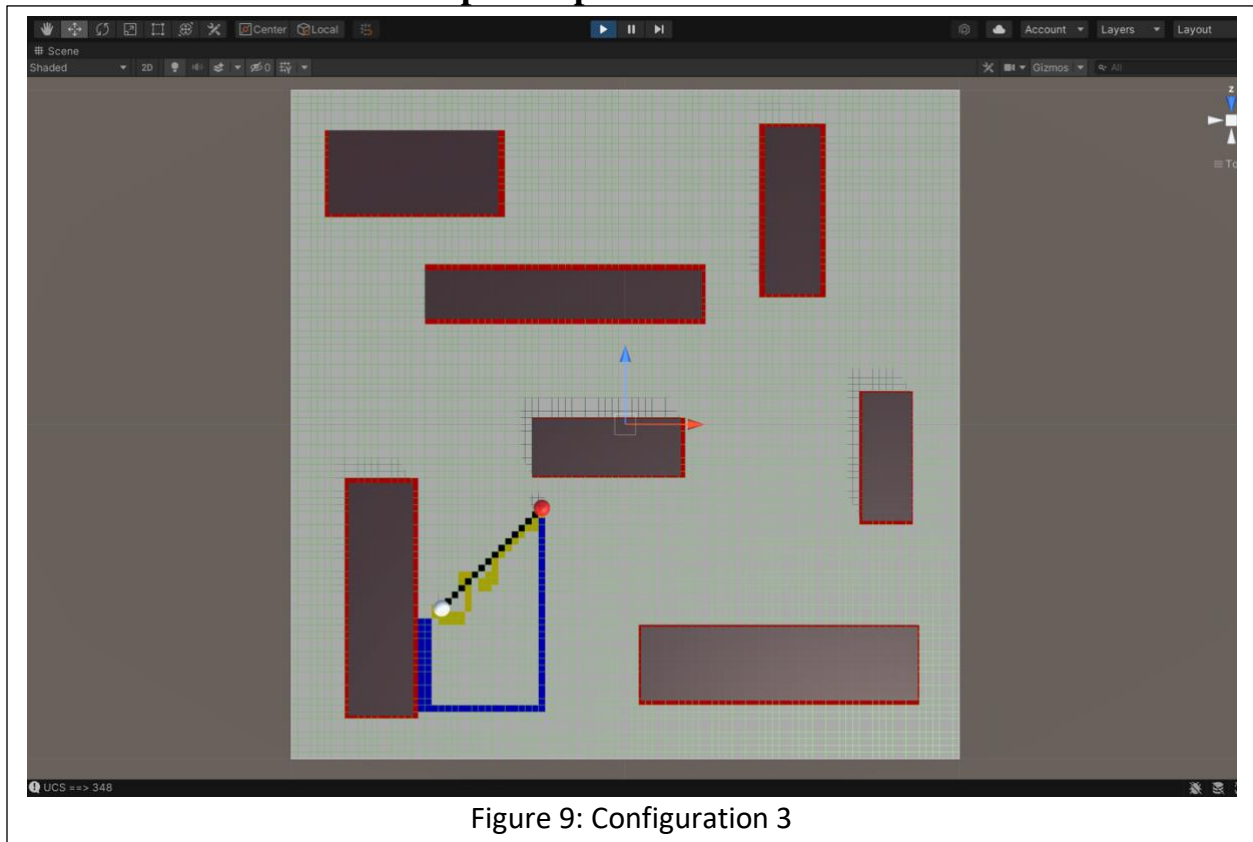


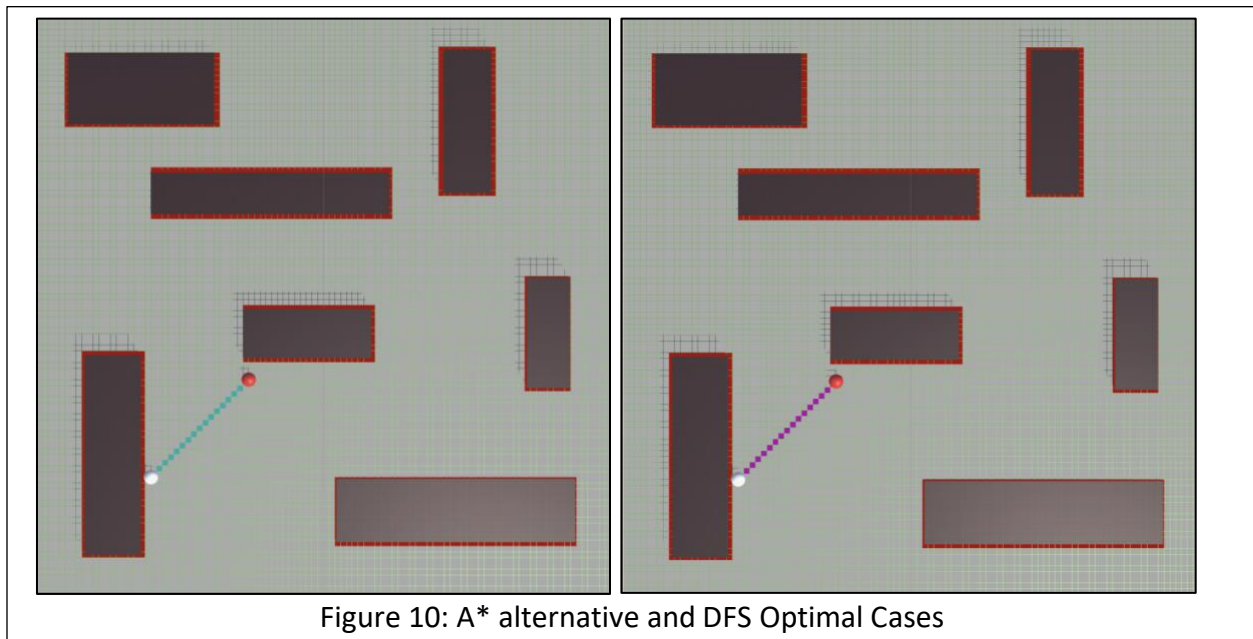
Figure 8: Configuration 2 Stats

This figure shows that while A* expands only 15 nodes and reaches a maximum fringe size of only 64 nodes to find the shortest path, A* alternative and BFS expand 186 and 565 nodes and reach a maximum fringe size of 81 and 328 nodes respectively to find the same path as A*. The time elapsed by A* is also less than the two other algorithms, but since the grid only contained 40000 nodes in total, the time difference does not seem significant.

5. DFS and A* alternative optimal performance



Again, in this configuration, BFS, A*, and A* alternative all result in the same shortest path. A* happens to cover the paths of the two other algorithms in the above screenshot, here they are:



To reproduce this configuration, the following seeker and target positions can be applied:
In this case, the seeker and target were placed such that the nodes that make up the closest path

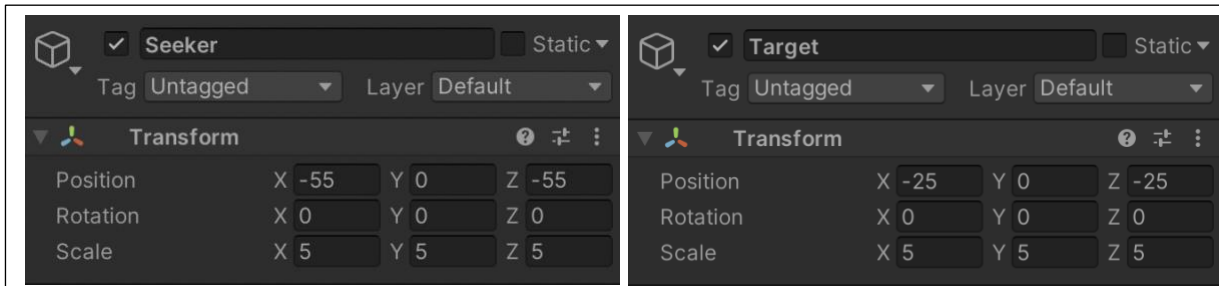


Figure 11: Configuration 3 - Seeker and Target Positions

between them are expanded first in both DFS and A* alternative algorithm. Now, the target was placed diagonally to the left above the seeker. However, the optimality of A* is not only manifested in the path length but also in the space and time complexities.

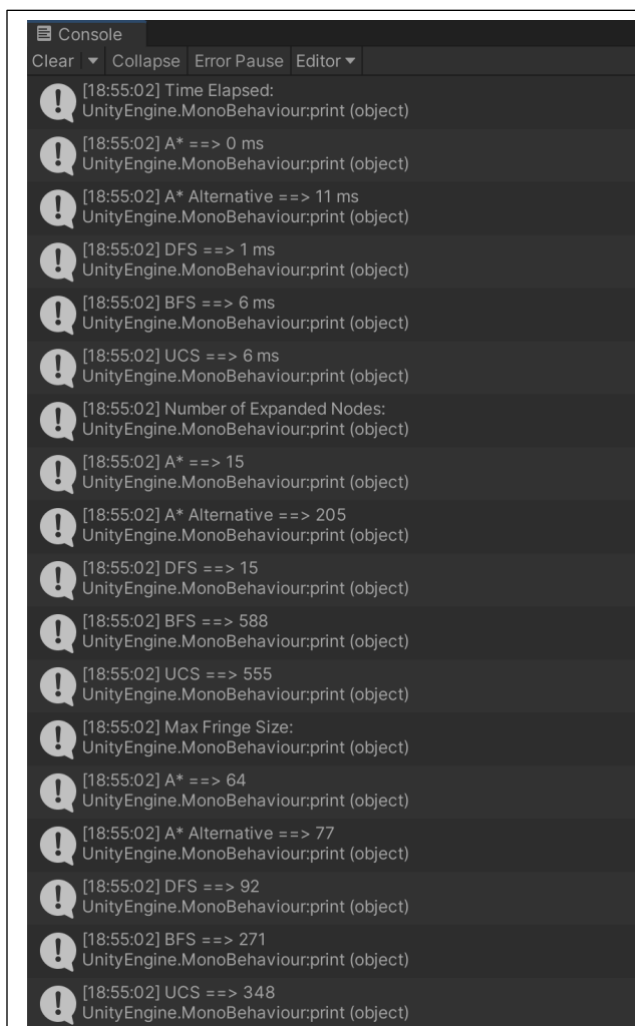


Figure 12: Configuration 3 stats

In figure 12 we can see that the number of expanded nodes in DFS is equal to that of A* (= 15) because all the nodes on the path happen to be the subsequent children of the start nodes that DFS would expand.

However, A* alternative seems to expand 205 nodes instead, because according to the heuristic we used, before reaching the half of the path, A* keeps expanding unpromising nodes that are close to the start, and only halfway through the path, it starts expanding the right nodes only.

The fringe space in this configuration is similar for all the algorithms, but A* still has the least fringe size.

6. UCS and cost function

The cost function we assigned to UCS assigns the same distance in all directions. This allowed it to prioritize diagonal moves. When we gave a higher cost to the diagonal moves, UCS became less efficient as it kept moving horizontally and vertically while a shorter path could be found through the diagonal moves.

The following figure shows the difference between these two cases:

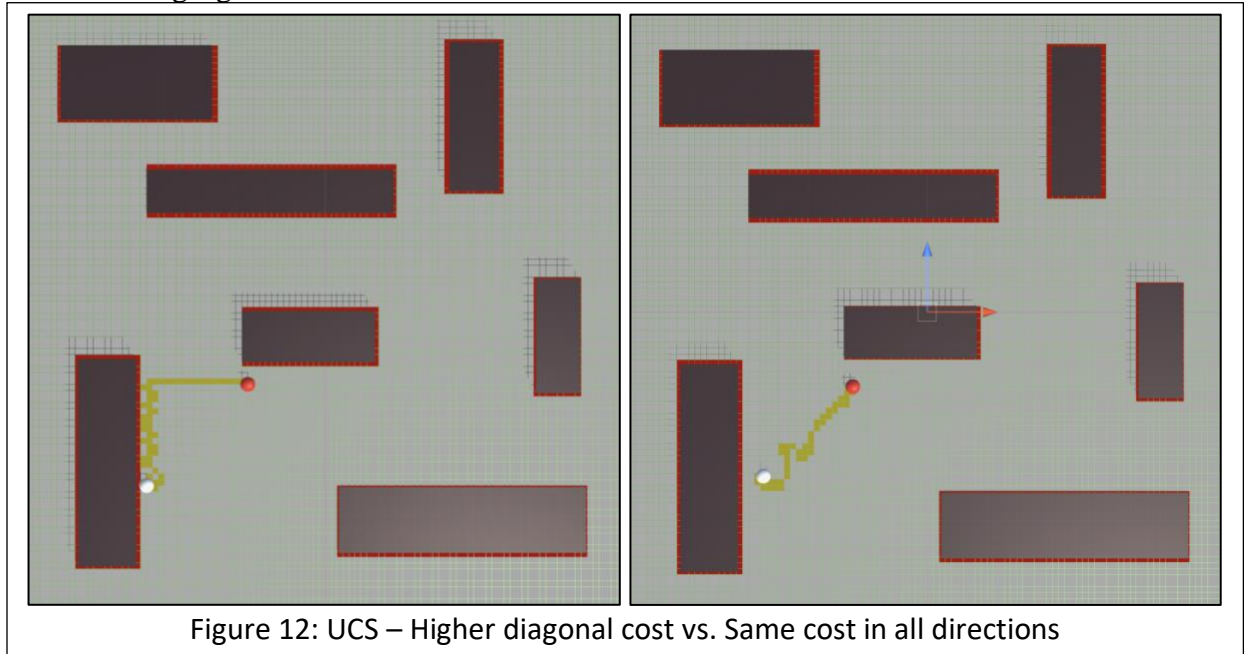


Figure 12: UCS – Higher diagonal cost vs. Same cost in all directions

However, when the target is on a horizontal or vertical move, UCS moves diagonally as shown in the following figure:

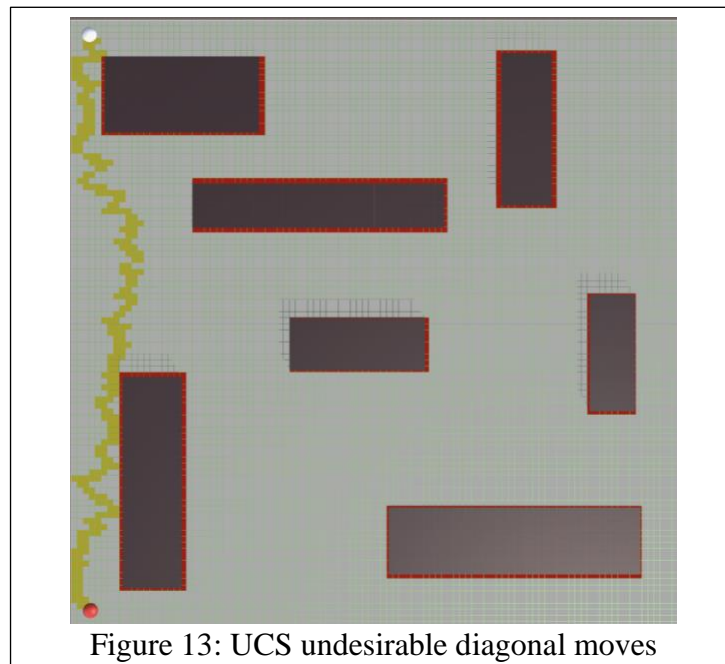
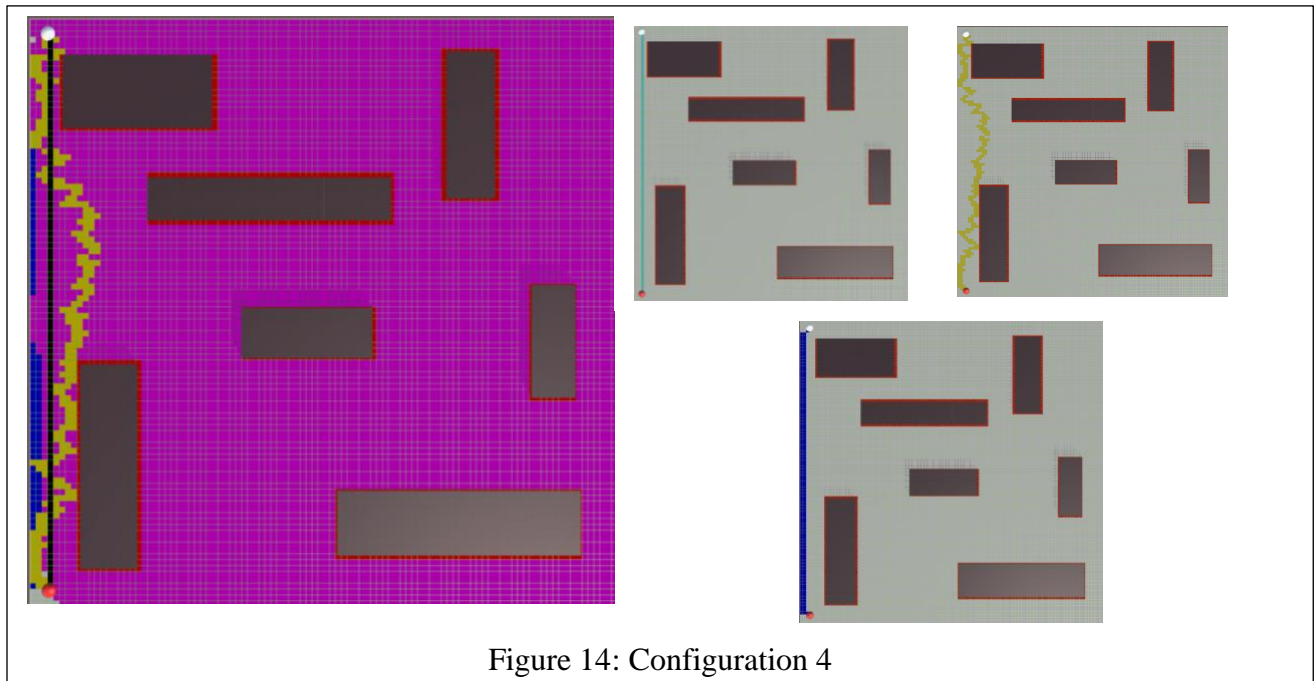


Figure 13: UCS undesirable diagonal moves

7. Additional Notes: Grid Dimensions, Node radius

When it comes to the time spent by each algorithm, we can see that in many cases all the algorithms were quite close to each other and were all completed in the order of a few milliseconds. All the configurations were run on a 200 x 200 grid and a node radius of 1, meaning 40,000 nodes were on the plane. Unfortunately, this number may not enough to see the time difference between the algorithms. However, this grid size was the maximum we could reach since our computers were not capable of handling a larger size.













Following is another configuration that can emphasize what has been discussed above with simpler configurations:



A* alternative found the shortest path because the nodes on that path happen to have the minimum cost according to the heuristic.

DFS misses the shortest path since it always expands nodes following a specific order regardless of the position of the goal (it is an uniformed search algorithm).

UCS makes fluctuating diagonal moves that took it far from the target at first. BFS expands many nodes and once it encounters the goal it draws the path. Although BFS found a short path compared to DFS they still both expanded a similar number of nodes:

 [18:40:52] Time Elapsed: UnityEngine.MonoBehaviour:print (object)	 [18:40:52] Number of Expanded Nodes: UnityEngine.MonoBehaviour:print (object)
 [18:40:52] A* ==> 5 ms UnityEngine.MonoBehaviour:print (object)	 [18:40:52] A* ==> 94 UnityEngine.MonoBehaviour:print (object)
 [18:40:52] A* Alternative ==> 36 ms UnityEngine.MonoBehaviour:print (object)	 [18:40:52] A* Alternative ==> 1298 UnityEngine.MonoBehaviour:print (object)
 [18:40:52] DFS ==> 11 ms UnityEngine.MonoBehaviour:print (object)	 [18:40:52] DFS ==> 7645 UnityEngine.MonoBehaviour:print (object)
 [18:40:52] BFS ==> 13 ms UnityEngine.MonoBehaviour:print (object)	 [18:40:52] BFS ==> 6462 UnityEngine.MonoBehaviour:print (object)
 [18:40:52] UCS ==> 35 ms UnityEngine.MonoBehaviour:print (object)	 [18:40:52] UCS ==> 6545 UnityEngine.MonoBehaviour:print (object)

References

- Sebastian Lague YouTube Tutorial: https://www.youtube.com/watch?v=-L-WgKMFuhE&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW
- Sebastian Lague A* Algorithm Source Code: <https://github.com/SebLague/Pathfinding>