Time Analysis of Pathfinding Algorithms

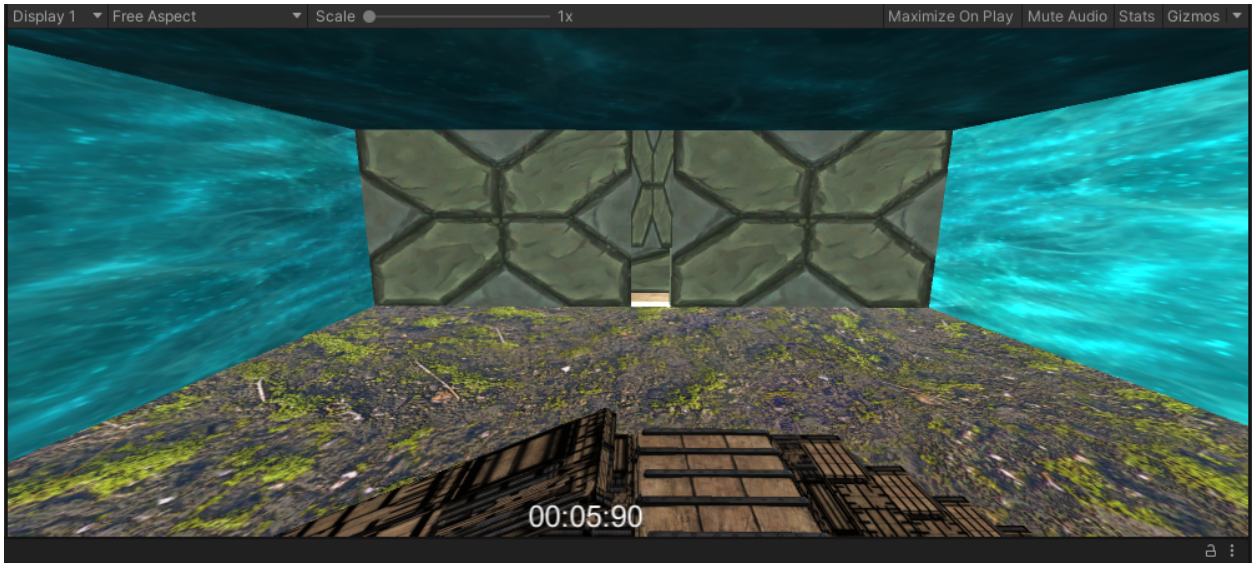
Diana Stephanie Wong
301373179

Introduction

What is the best way to get from one point to another? This is a common question and well studied question with useful applications in the game industry. In the context of computer applications, pathfinding is the plotting of the shortest route between two points [1]. Before we get too far ahead, let us define some more terms. Waypoints are intermediate locations on a route [2] and the A* algorithm is a search algorithm that finds the shortest path between a final and initial state [3]. Unity, which is a game-engine, was used to create an environment where NPCs (non-playable characters) followed scripts that used pathfinding methods. Additionally, NavMesh was used to implement pathfinding, it is an abstract data structure used to find paths [4]. In this report we will take a closer look at two methods used for pathfinding, the first being with the use of waypoints and the second being the A* algorithm. In particular we will analyse whether waypoints or the A* algorithm results in a NPC finding a path between two points more quickly.

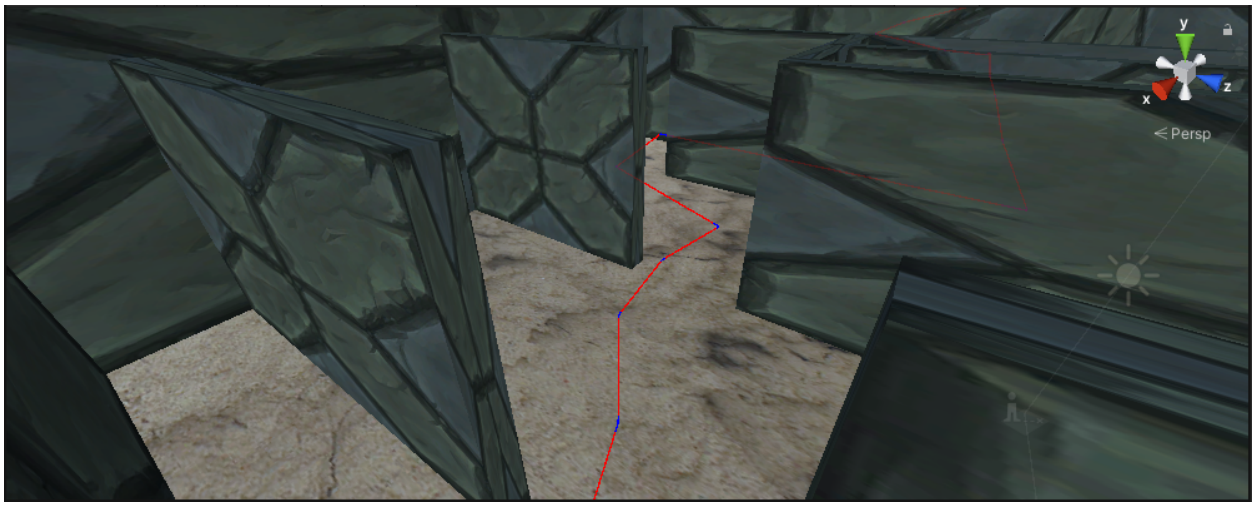
Experimental Setup

To evaluate the performance, I used a timer to record the amount of time it took a NPC to move across the map. The same timer was used for the waypoint and A* methods and it recorded the time in seconds.



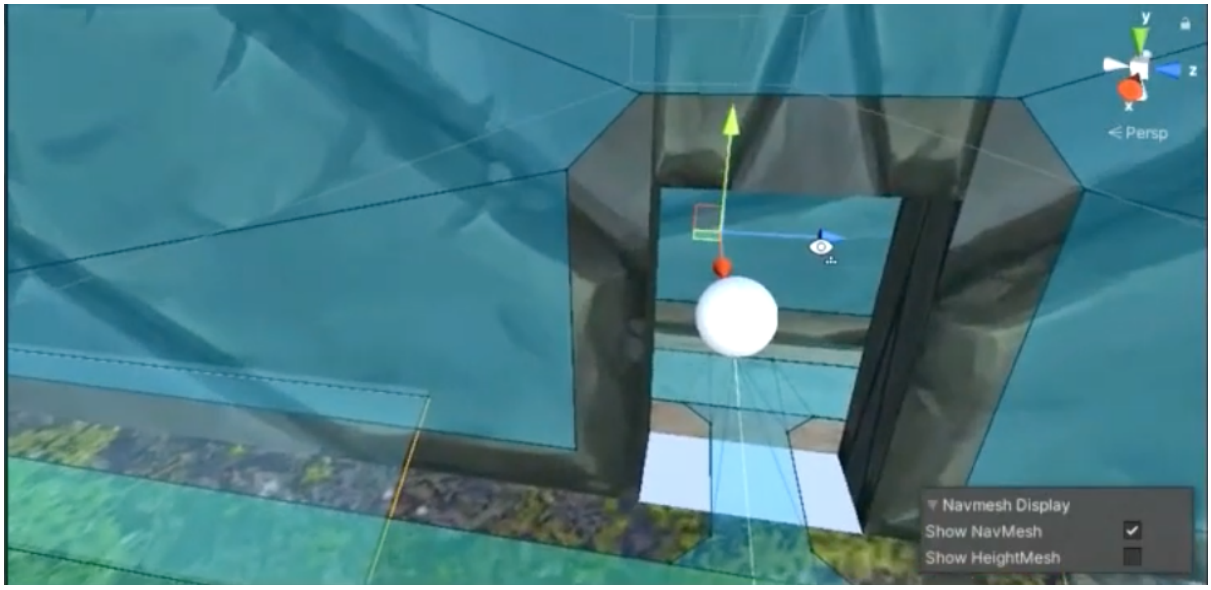
The start location and end location of the NPC were kept the same for all the trials of both algorithms. The same NPC was used for both algorithms and the speeds were set to the same value, to maintain consistency. By keeping as many variables as possible constant, we observed how the two methods for pathfinding differed. Since their performance was evaluated using the time of each algorithm to move the NPC, the one with the shortest average time was deemed to be the better method.

The waypoints method involved manually creating several waypoints in the Unity environment. Those were then placed on the map in order, for example the first waynode was number 1 and it was closest to the starting position of the NPC. The last waypoint was number 24 and it was located across the map at the chosen end location. For the script, I had to create an array that could contain all the different waypoints so that they could be found by the character. Using the waypoint list, the agent turned to and moved towards the next waypoint in the list until it reached its destination. In the image below, the blue parts are where the waypoints are and the red line represents the path the NPC follows using the waypoints.



For the A* method, I used the algorithm included by NavMesh to get the NPC to find the shortest path and move to it. This was done by creating a NavMeshAgent and creating a script that set the location to the end point and made the agent move towards it. There was another implementation of the A* method using waypoints that were not included in the time analysis. The alternate A* script used waypoints as in the first method but then called an A* function to get the best route. It used lists to store visited and unvisited nodes and found the path with the lowest cost. This algorithm version was not included because it did not function in a complex environment and since it was also

an A* implementation, I used the NavMesh one for the analysis. In the image below, the way NavMesh was applied is shown.



Looking at some similarities between the waypoint and A* methods, they both kept track of the NPC's speed. Different data structures were used, the waypoints one used an array, while the A* one was NavMesh. The waypoints method is limited because it requires us to manually add the points and depending on placement the path may not be optimal; however, it is an easy method to implement due to its simplicity. On the other hand, the use of NavMesh requires more software.

To evaluate the algorithms, I used my personal computer. It has an Intel Core i7 3770k processor (8M cache, 3.5 GHz), Radeon RX 560 GPU (4GB), and 20GB RAM.

The environment the NPC was travelling through consisted of a room with spiralling stairs that went to a lower floor. The lower floor was connected to another room with a doorway and the next room was a large empty room. Connected to that was a room with several walls that blocked paths, similar to a maze, and lastly another room was connected to that. The last room had some blocks and stairs. I found that the NPC had issues travelling down stairs with the waypoints method, but it got down. Additionally, the maze room had issues because sometimes the NPC would get stuck in corners, but by moving the waypoints those issues were resolved. The NavMesh did fine for the most part, but when it came to walking on walls the NavMesh Links that linked the floor and wall were an issue.

Results

Using the time metric previously described, I ran the experiment. I started the timer when the game mode started and the NPC started moving toward the end location. I ran the timer 10 different times for the NavMesh A* algorithm and averaged the results. I did the same procedure for the waypoints method and recorded the results below.

Trial	NavMesh A* (s)	WayPoints (s)
1	24.01	34.11
2	23.95	35.09
3	24.61	34.97
4	24.24	35.21
5	24.52	35.25
6	23.79	35.24
7	23.91	35.02
8	24.04	35.22
9	24.79	35.25
10	24.62	34.97
Average	24.248	35.033

Discussion

From the results we can see that the A* algorithm had an average of 24.248 seconds while the waypoints method had an average of 35.033 seconds. We can see that the average of the A* the method is significantly shorter. Now going back to the original question of whether waypoints or the A* algorithm results in a NPC finding a path between two points more quickly. The answer according to the results is that the A* The algorithm is faster because it took a shorter amount of time to go across the map. This was somewhat expected since the A* algorithm is ‘smart’, meaning it uses information about the graph to make decisions which separates it from other algorithms [5]. The A* algorithm calculates heuristics, keeping track of costs and chooses the best values [5]. This makes it better than the waypoints method because the way points method simply has the NPC move to the next waypoint in the list.

I did not expect the averages to differ as much as they did. There is about a 10 second difference in the averages and while it makes sense that there is a difference, 10 seconds is quite large. I think this likely happened due to some issues with the environment. One observation I made was that when the NPC was using the waypoints method, it had some trouble going down stairs and that was especially an issue when going down the spiralling stairs on the map. The NPC seemed to not move smoothly at those parts and while I tried to fix the issue by changing the placement of the waypoints

and varying the speed of the NPC, it still seemed to be a problem. In the end, it still moved but it could have been smoother and that should be investigated in future work. Also, when moving through walls in the maze, the NPC had some trouble turning corners, but that seemed to work better once more waypoints were added. Lastly, a reason for the large difference could be the locations where I placed the waypoints. When creating the waypoints, I just placed them in areas that seemed like they would lead to the final destination. Since I was simply estimating the best areas, the locations were likely not the optimal places for the NPC to go to get to the destination in the shortest amount of time.

Future work

There are parts of the analysis that can be improved on. Due to time constraints, the algorithms were only tested on one path so in future work, it could be useful to see if the algorithms' time performances change when the path is different. Perhaps, stairs or walking on walls would result in different findings. To further study pathfinding algorithms I would also test more algorithms, such as random walk or all-pairs shortest path. Comparing those with the algorithms already studied could provide more insights. In addition, I would implement the A* algorithm again using a function instead of just the NavMesh A* algorithm. Since my initial attempt at the A* algorithm used arrays to store the nodes to store the visited and unvisited nodes. While they are not the most efficient data structures, lists are easy to understand and work for a rough implementation. Using structures such as heaps or hash sets would be better as those will optimize the algorithm and likely result in improved time performance [5].

In addition, I would try to improve the timer that I used to take measurements so that the precision would be higher. I could do this by finding a way to measure milliseconds or nanoseconds instead of seconds and that will help lead to more precise results. Also, since I only took 10 measurements for each algorithm, in further studies I would take more measurements, for example 100 measurements of each one and average those. Testing the algorithms in different scenarios like discussed above would also be beneficial to see how other factors affect the time so more measurements in general would be an improvement.

References

"Coding Games and Programming Challenges to Code Better." *CodinGame*, https://www.codingame.com/learn/pathfinding.

"Waypoint." *Merriam-Webster*, Merriam-Webster, https://www.merriam-webster.com/dictionary/waypoint.

“What Is the A* Algorithms?” *Educative*, https://www.educative.io/edpresso/what-is-the-a-star-algorithm.

Filho, Moésio. “Unity‐Navmesh Tutorial (EN).” *Medium*, Medium, 23 Sept. 2019, https://medium.com/@moesio.f/https-medium-com-moesio-f-unity-navmesh-tutorial-en-bfeeccd6169a.

“A* Search Algorithms.” *GeeksforGeeks*, 28 Oct. 2021, https://www.geeksforgeeks.org/a-search-algorithm/.