Vivian Nathaniel Arul Rajkumar
ID: 20242020
E-mail: vnar@connect.ust.hk

COMP 3211 Final Project

Weighted A* Search using the Unity Engine (Spring 2017)

PROBLEM

You may have heard of games in which you click on a point on the game world using your mouse, and the main character walks to that point (such as in Diablo III for PC or other point-and-click games). Or you may have seen games in which the enemies all walk towards you and you have to escape. My idea was to make a slightly modified A* search in which non-player characters in games can take into account the different types of terrain they move across while moving towards a certain point. For instance, moving through a flat piece of land is easier than moving through grass, which is easier than having to move through water. I thought I could develop a weighted search method in which the agents stay as far away from water as possible, while formulating a path towards a point.

SOLUTION

In the 'game' developed, there is a target object (light green, on the right-side of the image) and 5 seekers/agents (white) scattered across the world. There are multiple obstacles, namely rectangular blocks (dark teal) and water (light blue). There are two bridges connecting the two separated parts of land, across which the seekers can travel.

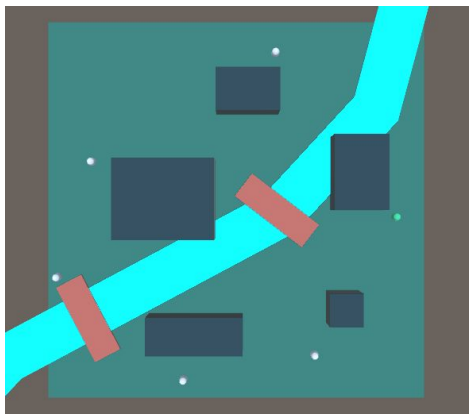

Fig. 1: Game world

The A* search that makes use of an open-set and a closed-set of nodes. First I divided the plane (on which the game was designed) into sections, and converted those into nodes. Each node will be classified under 4 terrain types: Unwalkable, Grass, Water, and Bridge. The A* search will not take into account any of the unwalkable nodes (where the dark teal rectangular blocks are), and will make use of the penalties assigned to each of the other terrain types.

For heuristics, I made use of three types of heuristic functions: Manhattan distance, Euclidean distance, and Diagonal distance. The code can be modified slightly if you'd like to test the program with either heuristic.

First I implemented the search using a list for the open-set (set of open nodes), but that was quite slow. So I improved the search a little by making a Heap class and then using a heap for the open-set. I did this because with a heap we can easily get the top-priority node (node with the lowest f-cost) from the top of the heap, and recursively sort all the other nodes in the heap. The speed of the search was also improved significantly (details to follow in the results section).

When run, the seekers will draw an optimal path towards the target, and then move towards it.

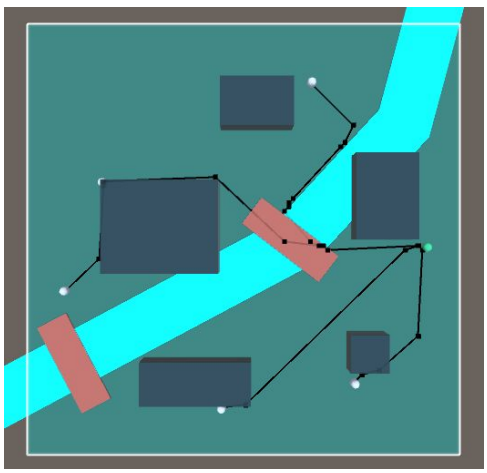

Fig. 2: Drawn paths

TUNING PARAMETERS AND TESTING

Here are the different parameters that can be tuned:

1. Terrain penalties (cost of travelling across each type of terrain)
2. Heuristics (Manhattan, Euclidean, and Diagonal)

Testing terrain penalties: If the penalties of each type of terrain are modified, the path of the seekers will change. For instance, if the penalty value for water was reduced a certain amount, seekers will be more likely to move through water to get to the target. The results of the tests were seen by the black paths drawn upon path calculation.

Testing heuristics: If the heuristics were switched, the time taken to calculate the A* path will change. Some heuristics take less time to calculate than others. In many cases, the heuristic chosen will also affect the path taken by the seeker. For example, when I used the Euclidean heuristic, the seeker on the far left did not take the lower bridge. However when I used the Diagonal heuristic, the seeker took the lower bridge.

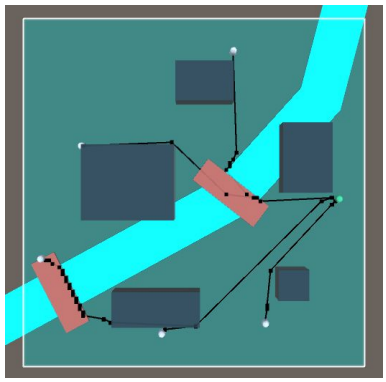

Fig. 3: Manhattan Heuristic

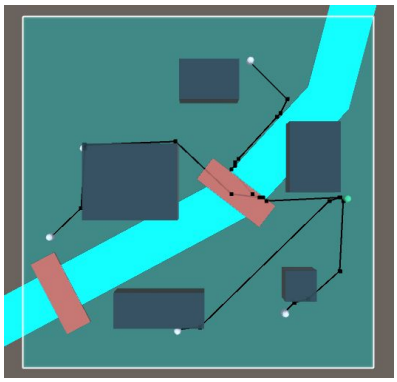

Fig. 4: Euclidean Heuristic

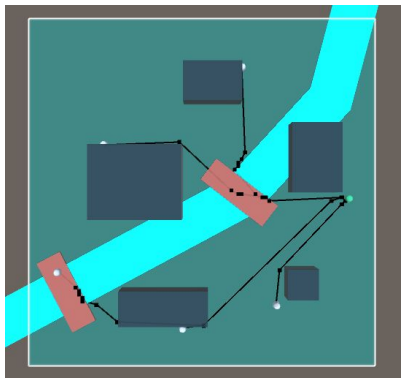

Fig. 5: Diagonal Heuristic

RESULTS

Implementation of open-set	Heuristic	Average time (ms) to calculate path
List	Manhattan	78
List	Euclidean	12
List	Diagonal	55
Heap	Manhattan	11
Heap	Euclidean	0
Heap	Diagonal	3

CONCLUSION

From the experiments, it can be seen that the Euclidean heuristic returned the best results. Another observation that was made was that the penalties for water had to be very high (around 1000) for the seekers to completely avoid the water while calculating a path. Performance was also good, since the heap open set combined with the Euclidean heuristic calculated paths in less than 1 millisecond on average. Additional improvements to the existing project could be that the paths can be smoother, since a smoother path would make the movement of the seekers more realistic.