README for Astar-Pathfinder code.

Greg Smith (B00308929) & Jack Smith (B00308927)

**Path Result:**

With a start position of 0 and an end position of 63, our algorithm found the shortest path to be:

0->3->8->11->9->10->4->12->18->30->49->54->60->63.

With a total travel cost of 1172.

**Algorithm Choice and Explanation**

This was tested using the A\* pathfinding algorithm implementing both the Euclidean distance heuristic and the Manhattan distance heuristic. We had expected to end up with a small variation in the final path when testing heuristics with a slightly different method of calculating the distance to the goal, but instead we found that the path was the same for the given start and end nodes, with the Manhattan method generally reaching the goal faster.

In more extensive testing using different paths we found that the two methods did not choose the same path on every occasion, the path between nodes 3 and 20 being one example of a variation where the Manhattan method found a different path and had a larger travel time.

This is most likely due to the Manhattan method having a simpler computation as a return value of the heuristic, with the Euclidean distance containing a square root calculation which is more computationally expensive but produces a slightly more accurate result. There were time variations in the calculations, but they were too small to be of any substance. Generally, <0.002s between the two in most cases. Overall on a graph of this size there is little difference in the effectiveness of using one heuristic over the other.

The A\* pathfinding search algorithm is a greedy best-first search algorithm, meaning that it uses a heuristic, such as the Manhattan distance, to predict how close the end of a path is to a solution and the paths which are considered closer to the goal are extended first, which can also be represented by the equation f(n) = g(n) + h(n), where h(n) is the heuristic modifier. A\* pathfinding also immediately exits when the goal is reached and will not consider any other possible paths that could lead to an ultimately shorter travel distance in terms of edge weighting.

We choose the A\* pathfinding algorithm as we wanted to test the difference between the Euclidean distance and Manhattan distance heuristics in terms of path choice and execution time, and we know already that in practise the quality of results from the A\* pathfinding method are typically good but also that accuracy is traded for performance.

We chose this over other algorithms such as Dijkstra’s pathfinding, which is essentially a breadth-first search that takes edge costs into consideration and uses a heuristic that is identically equal to 0. We predicted that this would give considerably worse completion times compared to A\* pathfinding because Djikstra’s will explore all nodes in different directions uniformly. This would lead to a Dijkstra implementation searching most of the nodes to find the shortest path, whereas the A\* pathfinding will take a more direct path and will typically reach the goal quicker. We also did not choose algorithms such as concurrent or bidirectional Dijkstra’s algorithm as the implementations were either not relevant to our graph type or were predicted to suffer the same slower computational speed as the other Dijkstra types.