CMPT 310 Assignment 1 Report

Joshua Campbell

301266191

In this assignment, I constructed a pathfinding program that finds the shortest path from a start point to an end point using the A* algorithm and I constructed a more efficient pathfinding program that finds a not always optimal path between two points by using landmarks to fill in the gaps.

Program Information:

For part 1, the program finds the shortest path using the A* pathfinding algorithm. For a heuristic, I chose the length of the shortest possible path from one tile to another (the length of the difference in the X distance between the two nodes plus the length of the difference in the Y distance). For example, if we have a node at (1, 1) and the goal node is at (17, 15), the shortest possible path has a length of (17 – 1) + (15 - 1) = 30. This heuristic calculation is applied to every tile on the map.

For part 2, the program finds a path between two points using the A* pathfinding algorithm as well as precalculated paths between landmarks. The program starts by finding the closest landmark to the start point and the end point (the closest landmark is determined using the heuristic calculation found above). If the closest landmarks are the same, the program will omit the landmark and it will instead find the shortest path between the start point and the end point using the A* algorithm (same as part one). If they do not share the same landmark, the program will find the paths between the start point and its closest landmark, the end point and its closest landmark, and it will join those two paths with the precalculated path between the two landmarks. This can result in suboptimal paths, but, generally, it is more time and space efficient than the pathfinding performed in part one.

For example, in part one, finding the path between (0, 0) and (17, 17) requires the expansion of 310 nodes and runs for 0.396 seconds on my machine (i7 6700k). This finds the shortest path with a length of 35 (including the starting node). For part two, finding the path between (0, 0) and (17, 17) requires the expansion of 93 nodes and runs for 0.0139 seconds on my machine. This finds a suboptimal path with a length of 37 (some nodes are visited more than once on the path), but it's execution time and number of nodes expanded is much less than part one.

Pathfinding Implementation:

When finding the path, the program will begin at the starting node and it will expand its neighbours to the north, south, east, and west (if they exist and aren't walls). These neighbours will be assigned the cost of the current node plus one and their parent node will be set to the current node. These neighbours will then be added to an open nodes list. On each iteration of the program loop, the program will find the lowest cost node (cost to get there from the starting node plus the heuristic cost) in the open nodes list and it will expand its neighbours. If one of its neighbours is the goal node, the program will backtrack from that node adding each of the nodes parents to the path until it returns to the starting node (this also signifies the end of the program loop). If none of the current nodes neighbours is the goal node, the current node will be added to a closed nodes list and the program will run through the algorithm again (starting by finding the lowest costing node in the open nodes list).

Notes:

For more thorough testing, both part one and part two can accept variable start and end points that can be run against the default map. See the readme for examples.

In order to verify the accuracy of my implementation of the A* algorithm, I have added optional map files and provided a few more complex map files with examples on their usage in the readme file. These map files can be used with part one along with any valid combination of start points and end points. (Note: the example with map2.txt should find no path since the start point is walled in).