Algorithms and Analysis Assignment 2

Approach Description

Dijkstra's algorithm is one of the many greedy algorithms that helps us find the shortest path from a source to a destination. In Dijkstra's algorithm, we can guarantee that the path from the source to that specific location is the shortest if it has been removed and added to the 'visited' list.

In our implementation of the algorithm, we made two hash maps, one named 'unvisited' and the other one name 'visited'. In the 'unvisited' hash map, we give it a key from the class Coordinate and its value being a double - the cost to that node.

Before we run the algorithm, we load each of the passable coordinates into the unvisited hash map and set its value to infinity. This is because at the start of Dijkstra's algorithm the source node does not have 'access' to any of the nodes except itself. Furthermore, we then add the source nodes into the unvisited hash map and give it the value of 0. This is because the cost of the source node to reach its source node simply is, 0.

We also set the source node's coordinates to have a previous coordinate of itself, as the previous coordinate of the source node will simply be itself.

We can ensure that at the start of each Dijkstra's pathfinder method, the source node will always be the first coordinate that will be added to the 'visited' list.

The 'visited' hash map has a key value of Coordinate and a value of Coordinate. The 'visited' hash map contains the key of Coordinate, which is the current Coordinate and the previous coordinate, which is the key's value.

Another method has been made in our algorithm called updateSurroundingNodes(Coordinate c). What this method does is it takes in the coordinate we want to check and update the node if the alternate path to that node has a cheaper cost than it currently does. If there is a cheaper cost [(cost to coordinate c + terrain cost of the surrounding coordinate) < current cost] , we set the new value of those surrounding coordinates, and update the previous node of the surrounding nodes, to the current node.

At the end of each update, we go through the unvisited hash map and choose the one with the cheapest cost, remove it, then add it to our 'visited' hash map with the Coordinate and it's previous Coordinate.

This is repeated in a loop until our unvisited hash map is empty.

At the end of the loop we then calculate the path of the source node and destination node. There is another method which is created called findPath(Coordinate start, Coordinate end, hash map visited). This method loops in our visited hash map and back tracks from the end coordinate and keeps back tracking until it reaches the source node. The path is then returned and is displayed for viewing.

This algorithm was implemented in a method called performAlg().

Task A

In Task A, we followed the algorithm above, as there is only one source coordinate and one destination coordinate where each coordinate with a terrain cost of 1. This algorithm is repeated until we reach the destination coordinate and has the path returned.

Task B

In Task B, the same algorithm is followed above, however with the acknowledgement of different terrain costs. Since Dijkstra's is a greedy algorithm the path will eventually be displayed. Hence, we took into consideration of the updateSurroundingNodes(Coordinate c) method. This resulted with Task B.

UpdateSurroundingNode(Coordinate c) enabled the program to keep track of a variable called costToCo - the cost to the coordinate, this variable holds the weight of the path between the coordinate and the source. With this variable multiple paths could be compared to even with the inclusion of terrain to obtain the shortest path.

We also needed to make sure that the algorithm is valid for Task A.

Task C
In Task C, we had to take multiple source and destinations into consideration and then select the shortest path that will be displayed to us. This was done by computing each source coordinate to each destination coordinate. We needed to compute for each source coordinate and treat the other remaining source coordinates as normal coordinates.

At the end of each iteration, we reset the map and check for the next source coordinate.

We then have a checking condition to check if that source -> destination is cheaper than the previous existing source -> destination. If it is cheaper, we update the value and initialise it to the path variable so that cheaper path is returned by the method instead.

We also needed to make sure that the algorithm continued to work for Task B and Task A.

Task D
In Task D, we needed to consider the waypoints and that each one needed to be passed before arriving to the source. To deal with this, we performed Dijkstra's algorithm form the source to each waypoint, while comparing and selecting the closest waypoint to the source/origin. As each waypoint is reached, it becomes the new source.