Armen Arslanian

Anderson Giang

ArslanianGiang282P3

Design Documentation

NOTE: This design doc explains the full features of the software. (Including things done in project 2)

F = G + H (TOTAL COST)

G = The distance from the source waypoint to its adjacent waypoint

Example: If the path from A -> B was 3, Node's B G value would be 3

A->B->C would be the distance from B -> C + C's parent's g cost (B).

H = HEURISTIC

Each individual waypoint has accessors and mutators for these three variables.

In the main class, we had to use a minimum priority queue that implemented the Comparable interface to compare the cost of our WayPoint object for the open set, and we decided to use a hash set for the closed set. We felt that it did not matter if we implemented the A* algorithm with a HashMap or a HashSet because if we implemented it with a HashMap, we would simply be iterating over the keyset() to see if our closed_list contained adjacent waypoints or not and we would simply be doing the same thing with the HashSet<Point>.

In our aStar(), we felt like we did not need to create a Node object, because we would treat our WayPoint object as our node. The closed set represents waypoints that have already been evaluated and are essentially removed from the path finding process. The open set represents all the possible waypoints that still need to be evaluated. We started with an initial waypoint and checked all of its neighbors. If they were not in the open set or close set, we calculated the distance from the initial waypoint to the neighbors and calculated the heuristic distance from mark to the destination waypoint. "Mark" refers to the individual waypoint

neighbors. We set the F, G, and H values for mark, set a parent for mark, and finally added mark into the open set and continue repeating this process for all of its neighbor waypoints. We also created temporary drawables after a waypoint or point was put into the closed set or the open set and we cleared those temporary drawables every single time that our program called the aStar() again because we felt like the entire terrain got way too messy and it was difficult to debug.

Using a stack when you reached the destination would be fairly simple to implement, which is why we chose to use a stack to find our path. We would add the parent of each waypoint to the stack until we reached the initial waypoint and we would pop a waypoint off the stack and that would be the waypoint that the player would move to. After each move, we had to check if the waypoint that the player moved to was a magenta waypoint. If it was, the player would calculate the path from the magenta waypoint to the treasure waypoint. We added a boolean field to the player called treasure to see if the player had a treasure map or not. If it did, the player would pick up the treasure map coordinates and go to it and if the player already had a treasure and it walked over another magenta waypoint it would just ignore it and continue on its path. We accomplished this all in our movePath(). In addition, if our player walked over a magenta waypoint it would set it's mapX and mapY values to be 0, so it remembers that we already took the treasure map, so it would not go in a cycle. Whenever we reached that treasure, we would also set its gold value to 0. Any gold waypoint that is visited, we collect the gold value, and also set it to 0. Note that all rules apply to the initial and final chosen waypoints. They are not treated as special cases. For example, if there is a treasure map on the beginning point, the player moves towards it.

For Project 3, we added a total of four players who each have a start and destination. They randomly take turns and proceed the same way discussed above. However, player,

treasure, map, and city information are now fetched from the SQLite database. If more than one player lands on the same WayPoint, they have a contest. The player who has the most wealth wins a third of the other players' wealth and the other players' lose a third of their wealth. These operations are only done on positive wealth players. Once all players reach their destinations, the player who has the most wealth is the winner.