CMPT 310
Assignment 1 report
haozheng
301266328

For part 1

I used the A start search as my strategy to find the shortest path from the start point to the goal point.

First of all, I create a class called SearchNode to represent the node with attributes x y coordinates, node-id, G(the node's movement cost) and H(the Heuristic cost), a pointer pointing to it's parent node. Also, I included the default constructor and parameterized constructor, GetF function to get the total costs(F = G + H) and GetH function(use Manhattan distance which add the x difference and y difference between the current node and the goal node).

After that, I created the class called PathSearch used to find the path from the start node to goal node. For the implementation of each function in the class, they are included in the Astarsearch.cpp. The main idea of the A star search is that we first create two lists, open list and close list to hold nodes. we start with the start node and put it into the open list, then expand the adjacent nodes of the current node retrieved from the open list with the smallest value(first time the node would be the start node). Then put the current node into the close list. The adjacent nodes of the current node would be right node, left node, upper node and down node. Next, we check the expanded nodes, if the node is the block one or is out of boundary, then just skip the node. If not, we add the node into the open list and continue the process recursively. Moreover, if the expanded node is already in the open list, we need to check if its value and pointer(to its parent) need to be updated. After some expansion, we will get the goal node.

Example input and output

input start (0 0) goal (17 17)
output
The length of the shortest path are: 34

```
showing the path

.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   _   _   _   _   _   _   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   |   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   |   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   |   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   |   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   |   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   |   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
.   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   *
*   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *

The length of the shortest path are: 34

The passed nodes are (from start to goal):
(0, 0) (1, 0) (2, 0) (3, 0) (4, 0) (5, 0) (6, 0) (7, 0) (8, 0) (9, 0) (10, 0) (11, 0) (12,
0) (13, 0) (14, 0) (15, 0) (16, 0) (17, 0) (17, 1) (17, 2) (17, 3) (17, 4) (17, 5) (17, 6)
(17, 7) (17, 8) (17, 9) (17, 10) (17, 11) (17, 12) (17, 13) (17, 14) (17, 15) (17, 16) (17,
17)

The total number of nodes placed on the fringe are: 312
```

The
passed nodes are (from start to goal): (0, 0) (1, 0) (2, 0) (3, 0) (4, 0) (5, 0) (6, 0) (7, 0) (8, 0) (9, 0) (10, 0) (11, 0) (12, 0) (13, 0) (14, 0) (15, 0) (16, 0) (17, 0) (17, 1) (17, 2) (17, 3) (17, 4) (17, 5) (17, 6) (17, 7) (17, 8) (17, 9) (17, 10) (17, 11) (17, 12) (17, 13) (17, 14) (17, 15) (17, 16) (17, 17)

The total number of nodes placed on the fringe are: 312

The search algorithm will find the shortest path from start to goal(expand right node first, then upper node, left node, down node)

For part 2, the strategy would be: use A* search to find the shortest path from the start position to the nearest landmark(search the 4 landmark and get the one with the shortest path). then do the same with the goal node. Next, precomputed the path from the two(maybe one) chosen landmarks. Then get the frontiers(all leaf nodes available for expansion) from the first paths(since the path between the landmarks has already precomputed) and compare it with the frontiers retrieved from going directly from the start point to the goal point. The idea is that the less expanded nodes we got,

the less time required to find the path from the start node to the goal node(which means more efficient to find the path, but not necessarily be the optimal path). Finally, we will get the path we want.

Input and output example
(0, 0) to (17, 17)

```
The length of the shortest path are: 36

The passed nodes are:
(0, 0) (1, 0) (2, 0) (3, 0) (4, 0) (5, 0) (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (6, 5) (6, 6)
(6, 7) (6, 8) (6, 9) (6, 10) (7, 10) (8, 10) (9, 10) (10, 10) (11, 10) (12, 10) (12, 11)
(12, 12) (13, 12) (14, 12) (14, 11) (15, 11) (16, 11) (16, 12) (16, 13) (16, 14) (16, 15)
(16, 16) (16, 17) (17, 17)

The frontiers from the second method: 146
```

The length of the shortest path are: 36

The passed nodes are:
(0, 0) (1, 0) (2, 0) (3, 0) (4, 0) (5, 0) (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (6, 5) (6, 6) (6, 7) (6, 8) (6, 9) (6, 10) (7, 10) (8, 10) (9, 10) (10, 10) (11, 10) (12, 10) (12, 11) (12, 12) (13, 12) (14, 12) (14, 11) (15, 11) (16, 11) (16, 12) (16, 13) (16, 14) (16, 15) (16, 16) (16, 17) (17, 17)

The frontiers from the second method: 146

Justification: since the frontiers obtained from the strategy in part 2 are 146 which is less than the frontiers obtained from the strategy in part 1(312). So the second one is more efficient.