CSCI 630 Lab1 Writeup- Michael Lee (ml3406)

My input function reads the elevation information and path (points) information from separate text files. (Terrain image is read in the constructor using ImageIO for convenience.)

Although the cost of A* search we learnt during the course is the distances among all cities, distances are not a proper cost for lab 1 because your speed differs according to the terrain types and elevations. One reliable cost unit for lab1 is the time spent while traveling from point to point (pixel to pixel). In order to determine the time cost among all points, I started from deciding the speeds for every terrain type (as shown in the table below) Besides terrain types, up/down hill with degree between 30° ~ 45° will decrease/increase your speed by 20%, and any hill with degree more than 45° or any place having an elevation of 3000 meters (too cold) will be treated as impassible.

Terrain Type	Color on Map (RGB)	Speed (m/s)
Open Land	248, 148, 18	3.5
Rough Meadow	255, 192, 0	2.7
Easy Movement Forest	255, 255, 255	2.4
Slow Run Forest	2, 208, 60	2.0
Walk Forest	2, 136, 40	1.5
Impassible Vegetation	5, 73, 24	0.0
Lake/Swamp/Marsh	0, 0, 255	0.0
Paved Road	71, 51, 3	3.5
Footpath	0, 0, 0	3.0
Out of Bounds	205, 0, 101	0.0

My cost function calculates and returns the time cost from one point to one of its adjacent ones (4 for most points) according to the rules mentioned above. If one point is considered impassible, my cost function will return 1000000, which is large enough that my A* algorithm will avoid those points.

Before finding the optimal path from one point to another using A* search, heuristic values of all points must be determined. My heuristic function keeps scanning every point and checks if there a better heuristic value for every point from its four adjacent points. The algorithm will keep doing scan and updates until there is no

update in a single scan. By doing this, every point's heuristic value will be optimal.

My A* searching function just performs normal A* algorithm. It first adds the starting point (along with its f(n) value) to the priority queue (sorted by f(n) values). After that, it starts exploring the starting point's neighbors and also theirs with the lowest f(n) value first until it reaches the visiting point. During the searching process, the optimal path is also recorded for calculating the total distance and drawing the path on the map image.

At last, my output function outputs an image with the optimal path drawn on it and a text file showing the total distance of the optimal path.