

Assignment 2 Process Letter

This assignment involved developing a path finding system for our agents, which allows them to navigate the game world using either Dijkstra's algorithm or A*. I started off by creating two graphs to test out A* and Dijkstra's. The first graph was a representation of a Central European city tour. It is a directed, weighted graph containing the cities I visited during a two-week period, with the weights representing the real-world distance (in km) between the cities. At first, I set up my graph to contain vertices represented by strings (i.e. "Geneva", "Budapest", etc.). This made accessing the graph's vertices easy, but problems arose later when I attempted to introduce a division scheme to the game world (which I'll discuss later). This graph included the following cities:

1. Geneva
2. Bern
3. Zurich
4. Vaduz
5. Innsbruck
6. Salzburg
7. Graz
8. Budapest
9. Bratislava
10. Vienna
11. Munich

Each of these cities also has an edge to its neighbor (Geneva → Bern, Bern → Zurich, etc.). This was a simple graph that I used frequently to test my initial implementations of A* and Dijkstra's. It was effective in letting me test the outputs of the algorithms, but not so much for testing if the best path was found, which is what I used my second graph for.

The second graph essentially divides the screen space (by default 1920 x 1080) into cells that are 120 pixels in both width and height. The neighbors of each of these cells are the cells directly bordering them (excluding diagonals). Like my Europe graph, this graph used strings to represent the vertices, which was adequate for my initial tests, but gave rise to some problems later down the road. With my two graphs in place, I moved on to implementing the algorithms themselves.

I started with Dijkstra's, referencing the Wikipedia pseudo code along the way. The implementation of A* and Dijkstra's algorithm has always confused me because they're a bit difficult for me to visualize, so I was a bit lost at first. Following the pseudo code (and also looking at some visual representations of the algorithm), I was able to implement the first iteration pretty quickly. Problems began to arise, however, when I realized that I still needed to keep track of the positions of each of the vertices within the world, something that was necessary for my A* heuristics to work properly.

For context, my graph class contains two C++ vectors, which hold the graph's vertices and edges. The vertices were at first represented by a string, which made accessing vertices easy (in the case of looking up the distance and visited nodes in maps). With this implementation, I had no way to know the in-game distance between two vertices, which made it impossible for me to figure out how far two vertices were from each other. At that moment, I decided to completely revamp my Graph and Edge classes in order to make use of a new Vertex class, which contained all sorts of data, such as the vertex's row, column, x position, y position, and if it was considered a wall or not. While it helped me quickly get A* working again, I have some doubts that this change was even necessary in the first place, but I'll discuss that later.

A* wasn't too different from Dijkstra's in terms of difficulty—I had it completed in relatively the same amount of time, thanks to Wikipedia's page on the topic. I started out with a heuristic that just returned zero so that I could focus on implementing the algorithm itself. I then implemented a heuristic that calculates the straight-line distance between two points (using openFrameworks's `distance()` method). After that, I implemented Manhattan distance, an admissible heuristic which is calculated by subtracting the x and y coordinates of both locations, taking the absolute value of the results, and then summing them together.

I then spent a good chunk of my time on trying to combine my path finding algorithms with my already existing boid movement code. This was a challenge at first, mostly because of the aforementioned problems with my graph class. Once I made those changes, however, everything started to fall into place. I had to do a bit of extra work to make the boid's target a vertex instead of the mouse's position, but once that was accomplished I had a boid that could follow a path from its current position to a destination (in this case, wherever the player clicks)!

Lastly, I added in some random walls to make the path finding more interesting, and the breadcrumb system I implemented in the first assignment. All in all, I'm pleased with how this assignment turned out! I was able to gain a better understanding of the A* algorithm, heuristics, Dijkstra's algorithm, and openFrameworks. The more I use openFrameworks, the more drawn I am to it. It is so easy to draw things to the screen, which makes prototyping small 2D games easy and effective. I might even use it in the future to prototype some games of my own!