COS: 350 Project

By: Cory Sollberger and Zach Hutchins

We chose to focus on path recognition by simulating a very simple map that a car could take within a city. Our map focused on small clustered sections of a city with multiple paths with varying complexity. We wanted to simulate a simple GPS system that a car might take trying to find the quickest path to a destination on the map. We chose to make vertices on our graph intersections between roads, where roads were connecting weighted edges between these vertices. We define weight in two ways, as traffic and as speed. The weights of an edge are defined with an integer value between 20 and 50 mph, simulating the speed limits of roads. We also included the element of traffic where each edge gets a modifier between > 0 and <= 1 which changes the speed of that edge. Numbers closer to 0 simulate bumper-to-bumper traffic and numbers closer to 1 act like open roads. Combining these weights we wanted to simulate a somewhat real experience of finding a route to a destination which takes the shortest amount of time. We also created an algorithm to randomly generate the map that shows the algorithm working for a variety of potential cases.

We used the paint component from java's graphics class to visually show the map we render and the vertice points including the edges between intersections. We also show the start destination as a yellow vertice, and the destination as a red vertex. When the user selects the start button the program will run Dijkstra's algorithm to determine the cumulative weights between all edges between the start position and destination. Using this information we were able to find the shortest path quite easily and we mapped it onto the canvas with a blue line indicating the path.

Dijkstra's time complexity is $O(|E| + |V| \log |V|)$, and we expected the algorithm to run in that time. Because of our implementation our tests could not calculate edges and vertices above a set amount because it is mapped onto a 10x10 grid. We chose to create maps that correspond to a natural placement of architecture like you'd see in the real world so our algorithm cannot be tested at an asymptotic level. However we do know it to be accurate and running at the established time complexity.

If we chose to do this project again we would definitely consider a map size much bigger than 10x10, somewhere on the scale of 100x100 also featuring edges that correspond to curves rather than straight perpendicular intersections. It was a fun experience implementing a pathing algorithm that visualizes a real world product.