## 

Written Document 2: Final Report

April 8th, 2016

ECE 297 - cd016

Jemin Andrew Choi

Youngchul Han

YoungJo Kim

## 1.0 Introduction

Despite the abundance of mapping software in the market, there are many programs that suffer from slow loading speeds or unintuitive user interfaces. Consequently, users are left frustrated trying to navigate a slow and cluttered map [1][2]. This document will explain how our mapping software improves on current mapping practices, and how our map makes it easy for users to quickly view and understand the geographical features of the map. Our map achieves this goal by selectively displaying features depending on zoom levels and by integrating the terminal into the graphic interface through a command bar. In addition, the document will explore how the map guarantees fast response times through the use of Dijkstra’s algorithm and a combination of efficient data structures, such as unordered maps and priority queues.

## 2.0 Design Overview

In order to effectively convey geographical information to users, our team focused on making the graphical interface simple and easy to understand [3][4][5]. Our graphic interface restricts the number of features that can be displayed depending on the zoom levels. Our software also includes a command bar on the graphic interface to make inputting data more intuitive, and the short response times of our software show how the graphics of the map are smooth and fluid.

### 2.1 Visualization Overview

Our mapping software was designed so the users could be able to clearly and quickly identify key geographical information on the graphic interface. Our program achieves this by prioritizing data so only important geographical features are shown based on different zoom levels. For instance, as shown on Figure 1, users cannot distinguish natural features, such as ponds and parks, if all the major and minor streets are displayed on the lowest zoom level.

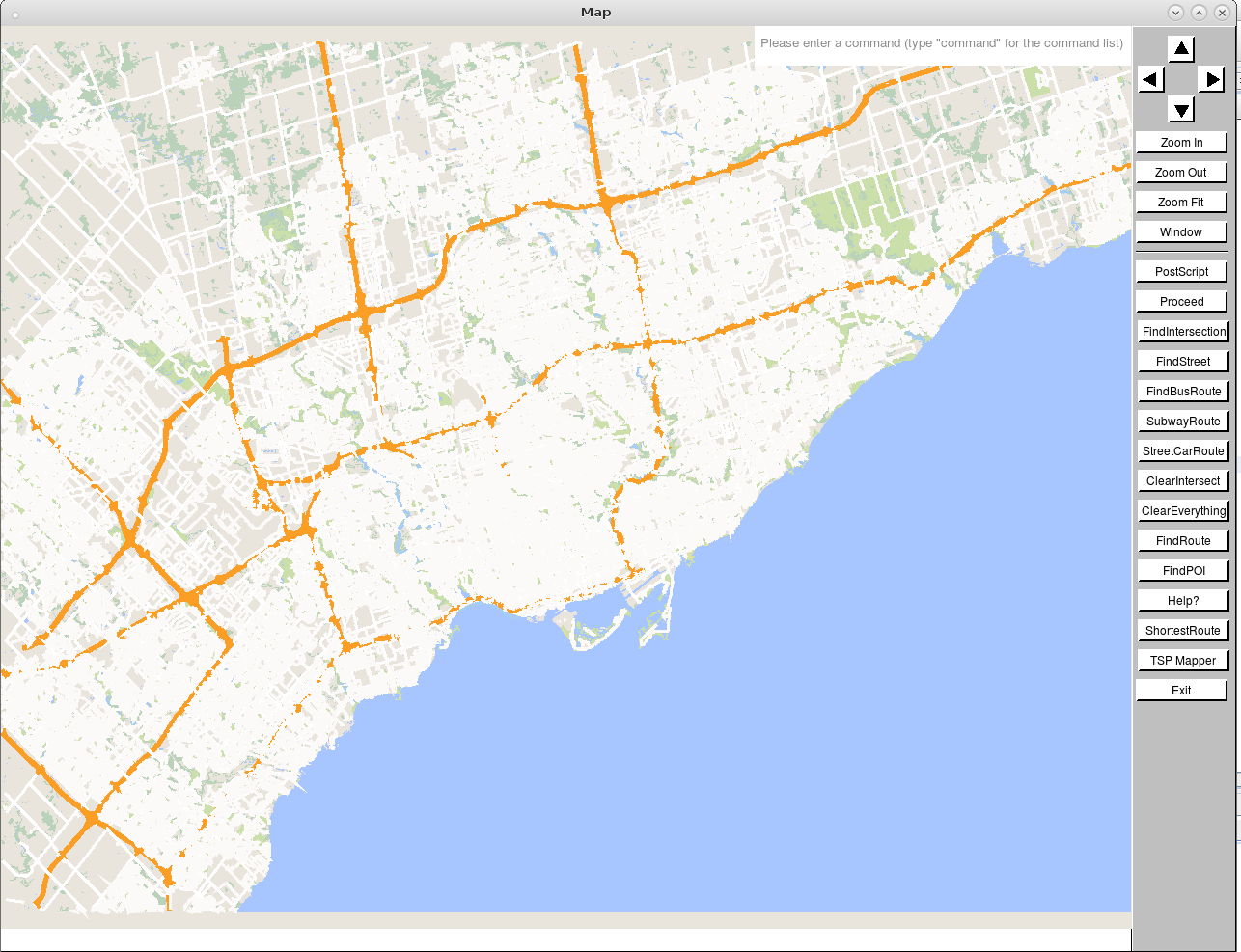


Figure 1: When too much information is presented to users at the lowest zoom level, users cannot distinguish individual streets and geographical features easily.

Consequently, we used the OpenStreetMap (OSM) data to classify different streets, ranging from major highways to residential roads. Depending on the zoom level, our program shows different levels of detail. For instance, at the lowest zoom level, only major roads, such as motorways, primary, and secondary roads are displayed (Figure 2). As the user zooms into higher zoom levels, the widths of major roads widen, and minor roads are displayed on the map (Figure 3). By selectively showing different streets at different zoom levels, our map helps users identify areas of interest with little effort and easily navigate through graphical elements [6]. Table A in the Appendix shows a detailed breakdown of which map features are shown at which zoom levels.

Likewise, our program emphasizes certain roads through different colours. Major motorways, as seen in Figure 2, are highlighted in orange. This allows users to immediately identify and focus on key roads [6].

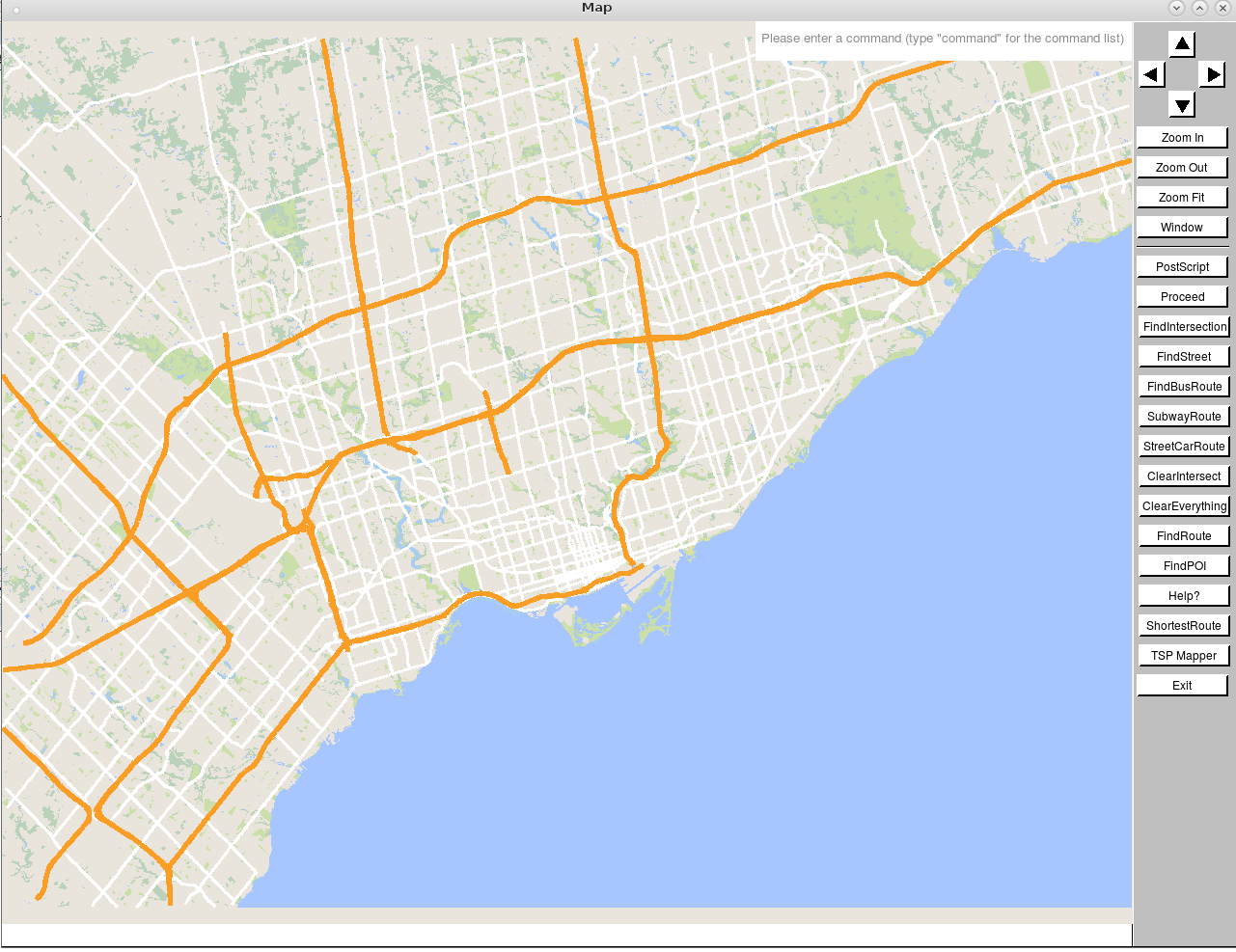


Figure 2: At the lowest zoom level, only major highways, and key geographical features, such as lakes and parks, are displayed on the map. Major motorways such as the Don-Valley Parkway and the 401 are highlighted in orange while residential roads are shown with lighter colours [7].

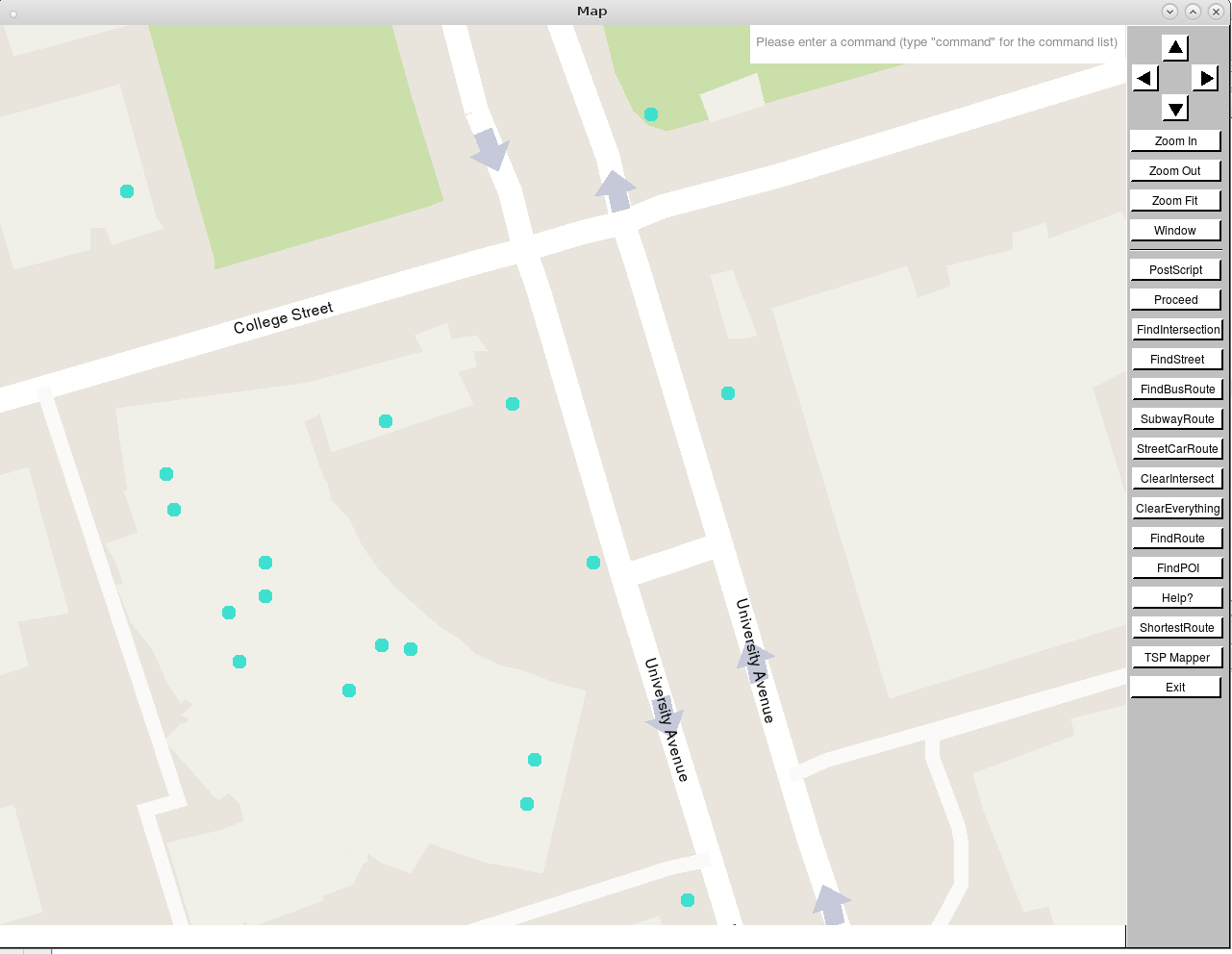


Figure 3: At the highest zoom level, every detail is displayed on the map, including street names, points of interest, and one way streets. The width of bigger streets are also wider than the widths of minor and residential roads.

As shown in Figure 4, when users zoom into the map, points of interest (POI) are shown as blue circles. This reduces on-screen clutter and displays where each POI is located on the map. Since displaying all the POI names would crowd the screen with letters, we decided to only display the POI's name when it is clicked. As seen on Figure 5, the POI's name appears right above the blue circle when clicked. We found that this technique is effective in displaying essential information while keeping the mapping interface tidy and interactive[8].

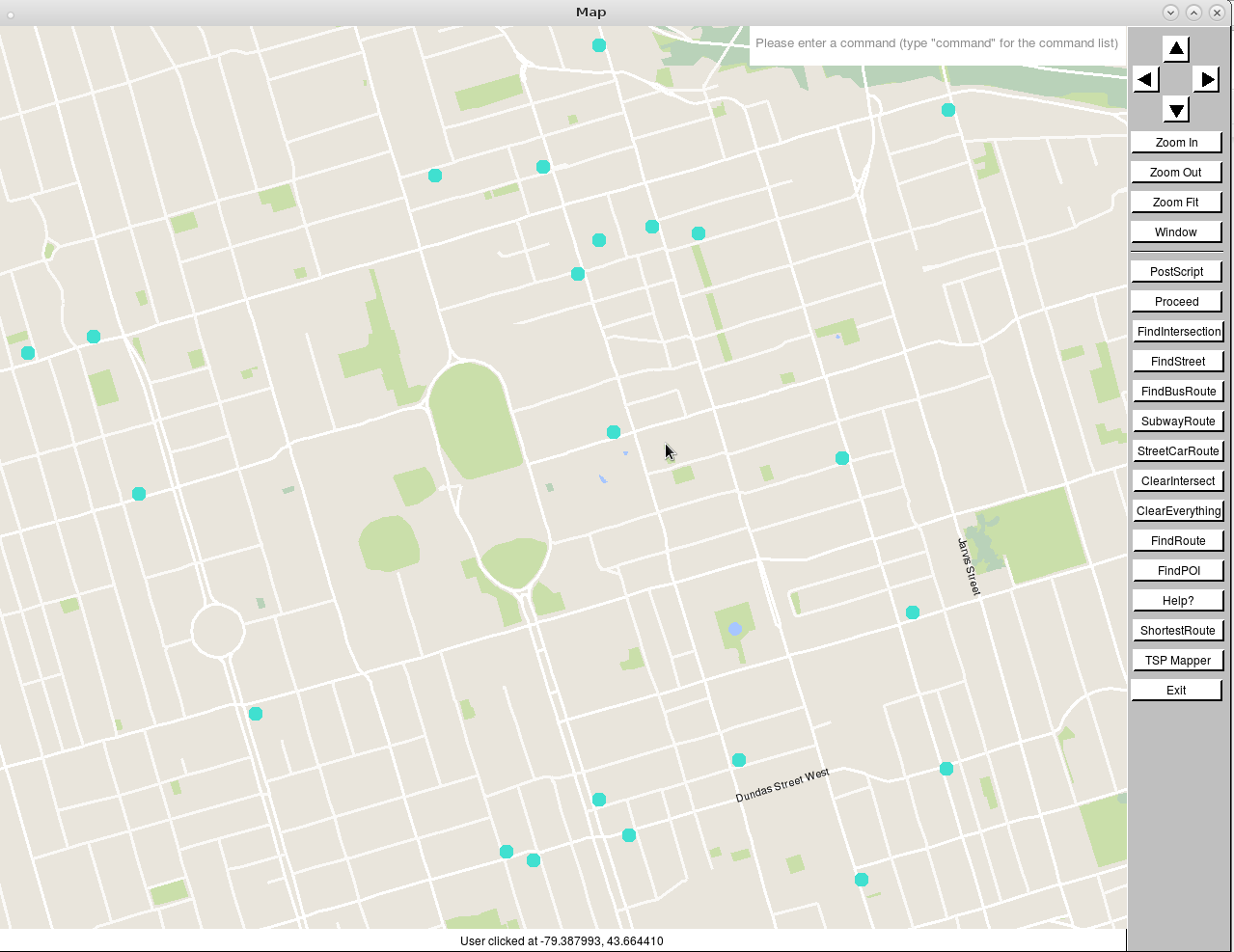


Figure 4: The circles represent where each POI is located on the map.

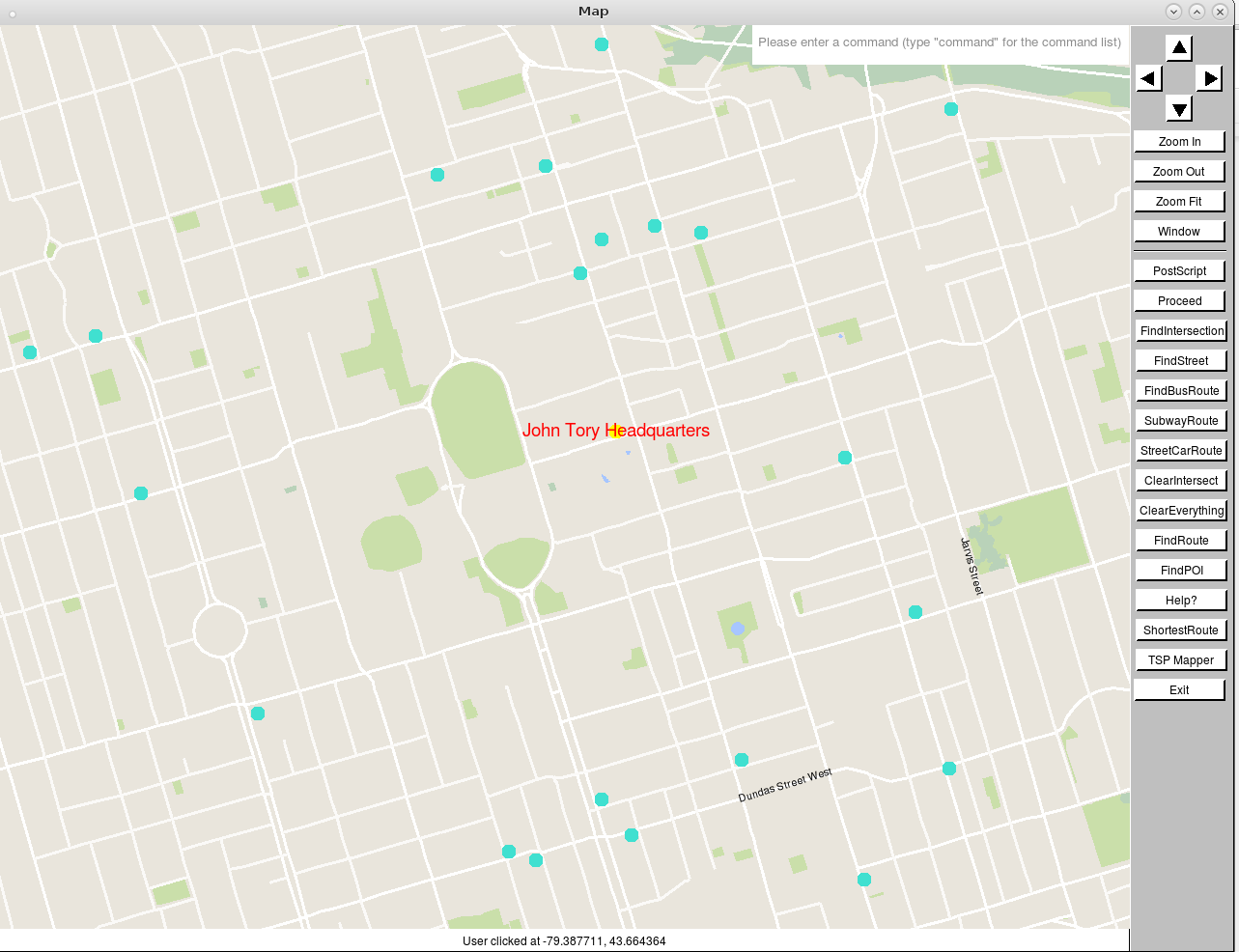


Figure 5: When a user clicks on a point of interest, the name of the point of interest appears on top of the selected circle. The map only displays the name of one point of interest at a time in order to keep the mapping interface visually tidy.

### 2.2 User Interface

#### User Inputs

To keep our map consistent with the principles of user convenience and simplicity, we added a command bar, an autocompletion feature, and an invalid input-prevention mechanism to the map.

Our program makes it easier for users to access specific commands or functions through a search bar. We included a search bar, as seen on Figure 6, because its utility is proven by many state-of-the-art software, such as Google Maps seen on Figure 7, that have similar features that allow users to quickly find certain features of the software[9]. In addition, by implementing common features from other mapping software, users can familiarize the features of our map more easily [9].

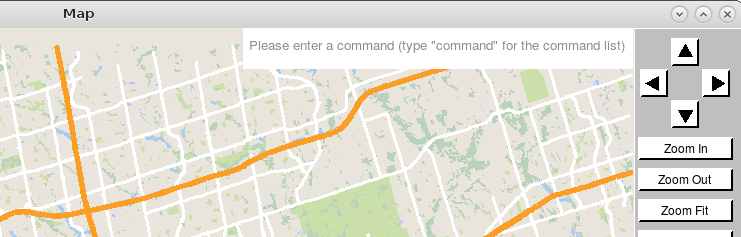


Figure 6: The command bar of our mapping software

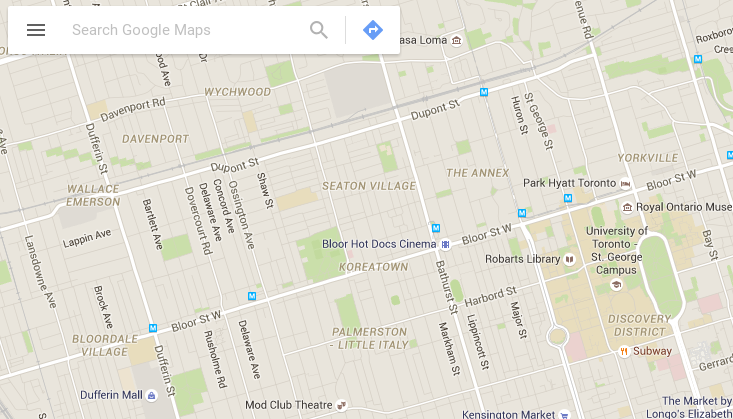


Figure 7: Search bar from Google Maps [10]

Moreover, our map includes an auto-completion feature on the terminal and on the search bar. Auto-completion allows users to type less and to save time by making smart suggestions for words. Figure 8 shows how the auto-completion is implemented to our program.

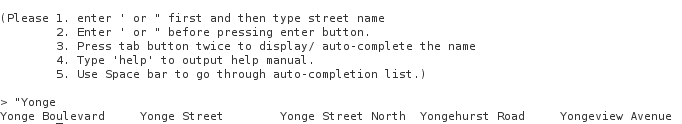


Figure 8: The auto-completion feature suggesting possible street names that user wants to search.

Errors often cause severe frustration to users [11]. If the program is unstable and crashes upon invalid input, the user will be annoyed by the program and the user may not use the software in the future [11]. To prevent this from happening, our software is protected from invalid inputs by a series of safety mechanisms. For instance, as seen in Figure 9, when a user provides invalid inputs, the program safely returns an error message instead of crashing.

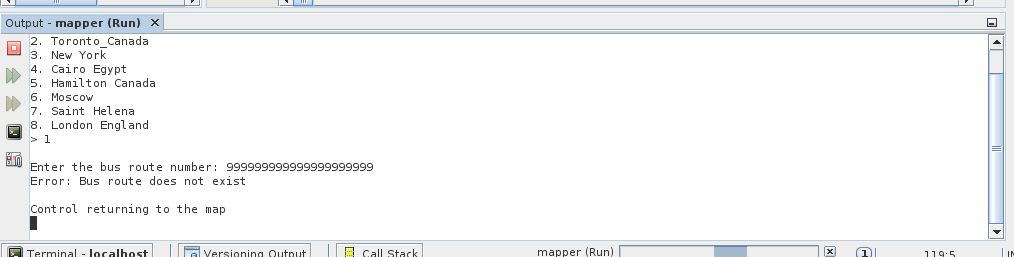


Figure 9: Display of an error message after a user input invalid bus number.

#### 

#### Direction Information

Our map displays path directions through the graphic interface and the terminal. For example, if a user searches for the shortest path between two intersections, the start and end of the path is indicated by a purple and green flag, respectively. Figure 10 shows how the map highlights the shortest travel route in blue. To simplify the graphical interface, detailed driving instructions, such as turning left or right at a specific intersection, are only displayed through the terminal. The graphic representation of travel routes helps users visualize the path, while the detailed directions of the travel path on the terminal gives users a more concrete view of the path (see Figure 11).

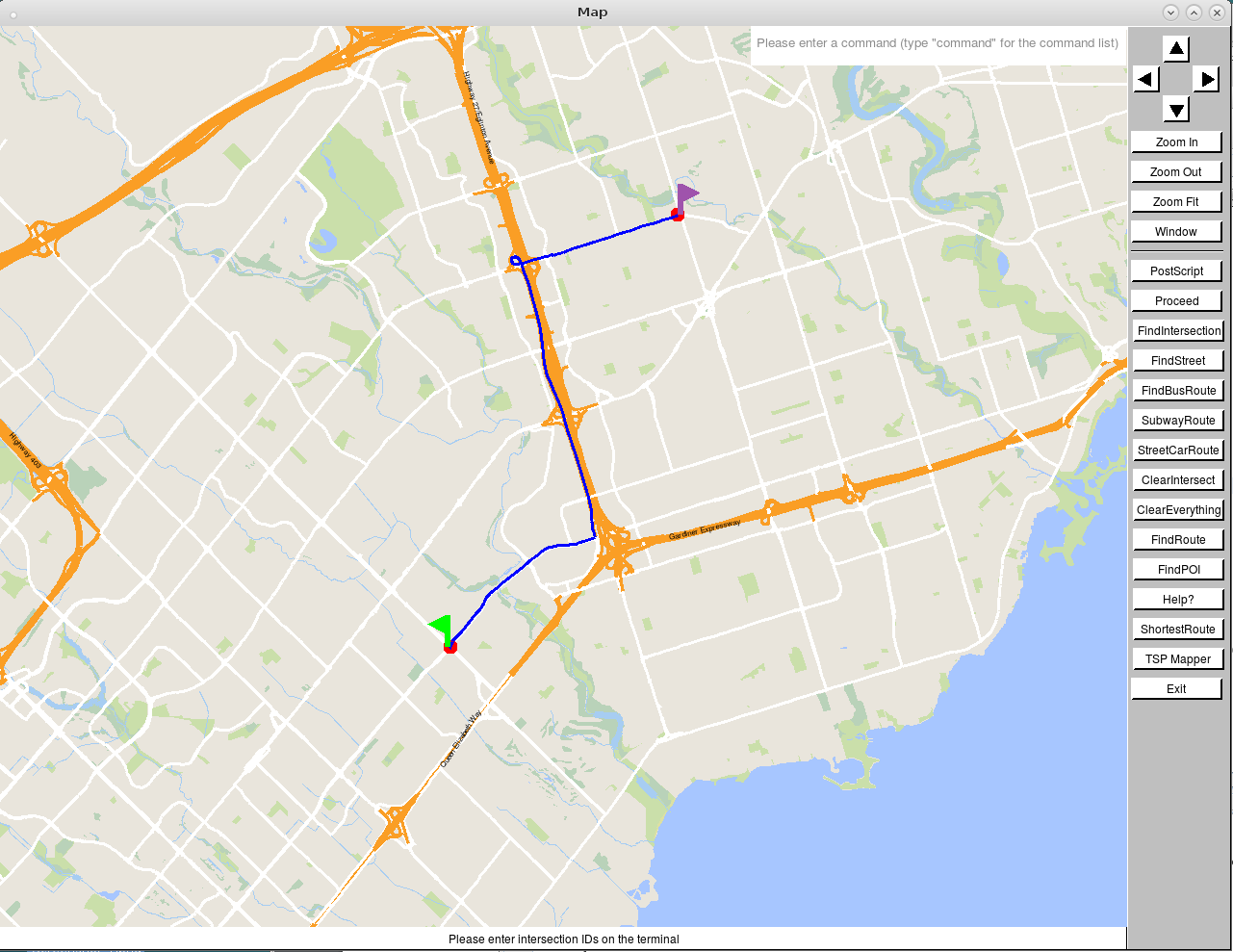


Figure 10: The display of the shortest path from an intersection to another intersection.

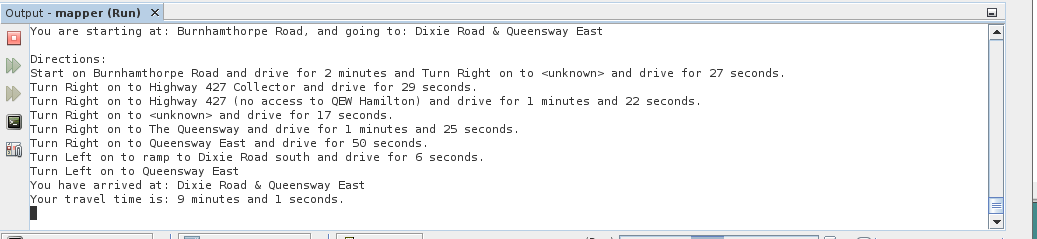


Figure 11: The detailed direction of the shortest path displayed on terminal.

### 

### 

### 2.3 Special Features

Due to the rapid growth in public awareness and demand for sustainable modes of transit, we decided to implement a public transportation mapper as our special feature [12][13]. Our public transportation mapper is able to display bus routes, streetcar routes, and even subway lines for major cities.

At the click of a button, transportation routes are displayed on the map, as seen on Figure 12, and the related information is displayed on the terminal. Different colours are used to distinguish the transportation routes from the other roads and map features since similar colours may confuse users and lead to misinterpretations of the geography.

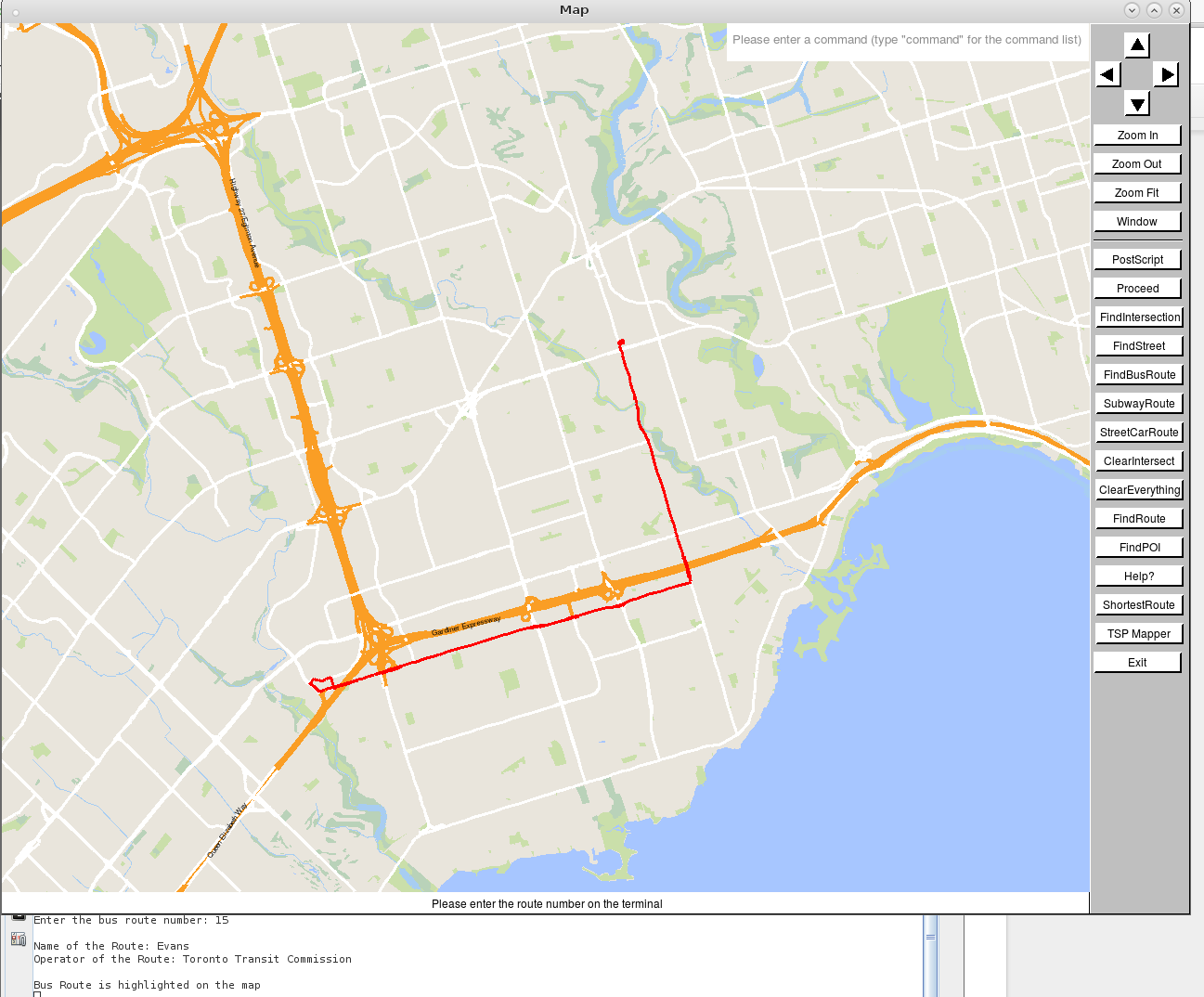


Figure 12: Screenshot showing a bus route on the map screen. Users can choose which transportation route to be displayed by clicking a button on the side of the map or typing a command.

### 2.4 Response Time

Our mapping software is designed to maximize performance instead of overloading the map with extra features that would slow the graphics down. We used the ECE297 exerciser and unit tests to define the standards for a fast mapping software. Our map passed every performance unit test with time to spare, and these results can be found in Table B in the Appendix. Hence, we concluded that our mapping software would be fast enough to provide a seamless user experience.

In addition, our program’s graphical interface can draw one frame of the map in 0.1 seconds. Considering that an average human blinks for 0.40 seconds, our mapping software can draw four frames at the blink of an eye [14]. Likewise, our function can search paths five times faster than the suggested benchmark, in under 0.02s [15]. Runtimes of other important functions can be found in Table C in the Appendix.

Our program finds courier paths by loading travel times between deliveries and iteratively improving the first legal solution that it has found. Loading 260 deliveries locations for Toronto took about 6 seconds, and each subsequent path it searches takes less than 1 millisecond. Effectively, our program can find courier routes with a Quality of Result (QoR), where a lower score means a better route, of 1490 after 30 seconds of runtime. As seen in Figure 13, there is also a tradeoff between the solution quality and the runtime. Hence, our program will be feasible to use for personal and commercial use for finding courier routes.

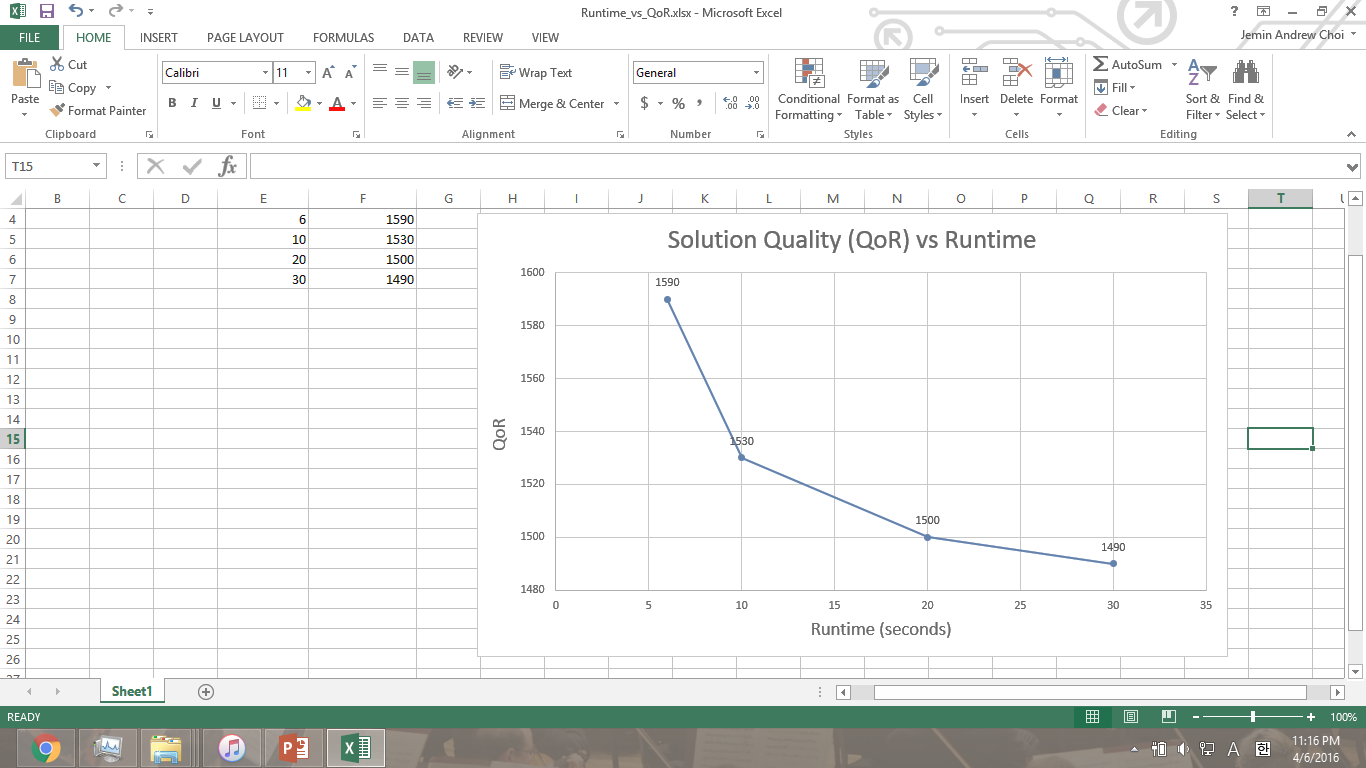


Figure 13: Graph showing how the Quality of Result (lower scores means better routes) vary depending on the runtime of the program. A longer runtime yields better results because more path computations and perturbations can be made, and as a result, a better route can be found.

## 3.0 Technical Overview

To achieve the core of the project, our team used algorithms and data structures that are fast to access and search data. This section will present a higher level view of the code and explain the algorithms and data structures that our team used to efficiently make calculations and draw the map.

### 3.1 Higher Level View of Code

Our software is organized in two main folders: libstreetmap, libstreetsdatabase. The libstreetmap folder contains m1, m2, m3 and m4 files as well as the other class header and definition files. The libstreetsdatabase folder contains files that house OSM data. Figure 14 shows a block diagram of all the files used in the software.

The functions within the m1 file are used to allow coders to easily access map data. The main function of the m1 file is the load\_map function. This function loads the OSM data as well as the main data structures that are used by the other functions and files in our software. The m1 file also uses various data structures such as vectors and a k-d tree to access elements at low complexities.

The m2 file contains a collection of helper functions, called by draw\_map, that are used to draw the map on the screen.

The m3 file contains the functions that calculate the shortest travel path between two intersections or between an intersection and a point of interest. It utilizes Dijkstra’s algorithm and a graph data structure. The definition of the graph can be found in GraphVertex and GraphEdge classes.

The m4 file contains functions that are necessary to find an optimal courier route. This file also uses a 2-dimensional unordered map to store travel times between deliveries.

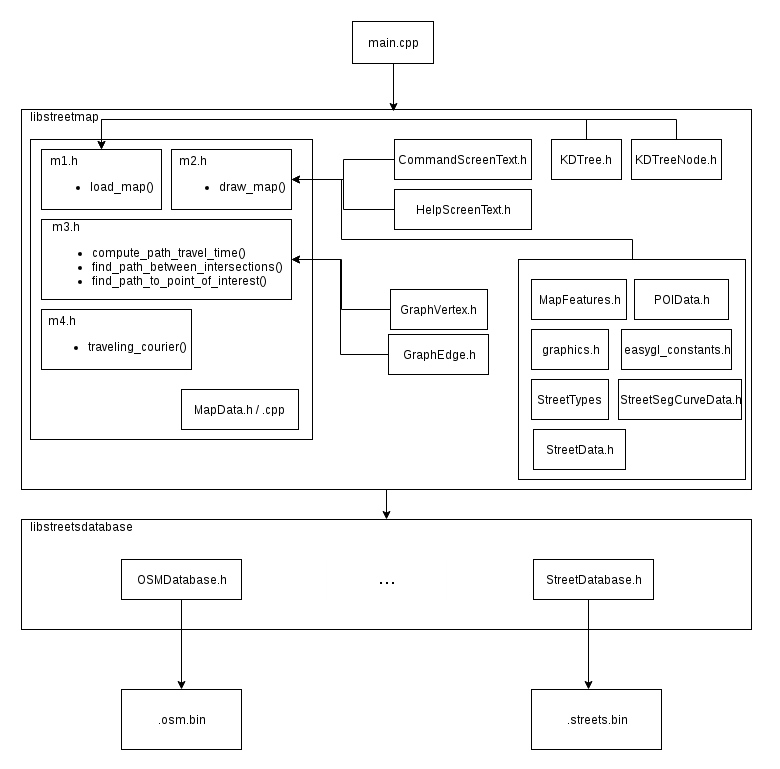


Figure 14: Block Diagram showing the organization of the code, classes and files of the mapping software [16].

### 

### 

### 3.2 Algorithm Design

Our mapping program uses a variety of algorithms to convey accurate geographical information and adhere to cartographic standards [17][18]. As seen on Figure 1, we thought it would be ineffective to display all the streets and street names at the lowest zoom level. Our software decides when to display specific streets and names when the ratio of the area displayed by the map and the city exceed a certain threshold. These thresholds are shown in Table A and Table D in the Appendix.

To reduce screen clutter and to accurately display street names, street names are only displayed if the name could be positioned and angled between the starting and end points of the street segment. In addition, for different zoom levels, we displayed every 8th, 6th, 4th, etc. street name in order to prevent the names over-crowding the geography of the map.

For curved streets, it was visually displeasing to display a name. Hence, for street segments that curved more than 10 degrees, we did not display street names on that segment (Figure 15).

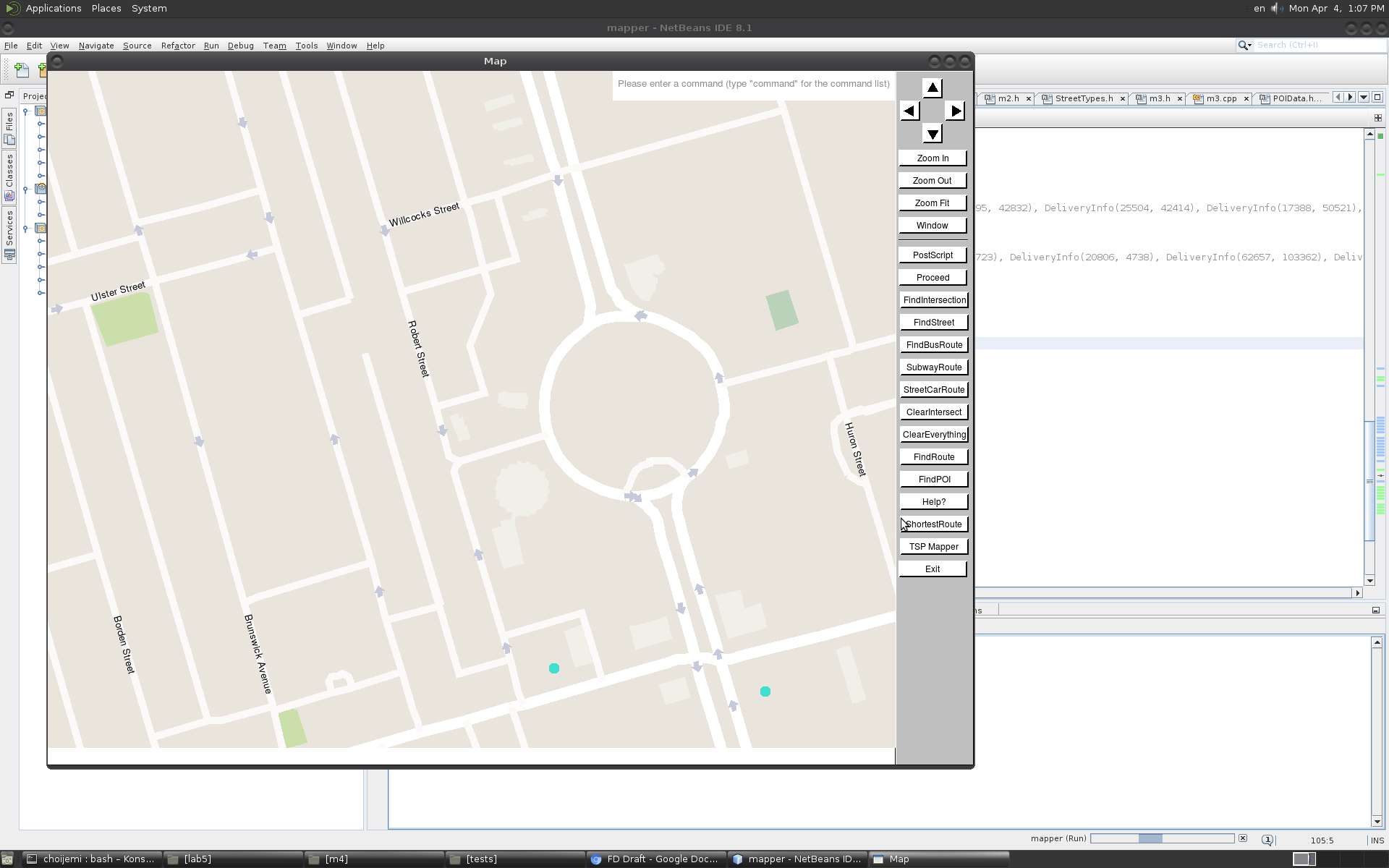


Figure 15: A screenshot from our mapping software showing street names absent on Spadina Circle.

For path-finding, our team implemented Dijkstra’s algorithm because it was well-documented, and effective for this single-sourced graph problem [19]. We initially started to search for paths using the Breadth First Search (BFS). However, the crucial limitation of the BFS was that it only found a path with the lowest number of travelled street segments. To mitigate this, we used Dijkstra’s algorithm. Despite both algorithms being equal in complexity, O(nlogn), Dijkstra’s algorithm yielded the shortest paths in terms of travel times rather than street segments [20]. Although we considered implementing heuristics to find paths more quickly, we determined that it was unnecessary as our path-finding function could pass the very hard performance unit tests on the exerciser by a factor of 2, as seen in Table B in the Appendix.

Our program uses a greedy algorithm and simulated annealing to find optimal courier paths. Our program initially calculates travel times to every delivery and stores them in an unordered map. To quicken this process, we used eight threads to compute multi-directional Dijkstra searches. Then, the program uses a greedy search to construct the first route based on shortest travel times. We used a greedy search to find the first route because it was quickest method to find a legal solution, yielded good routes (QoR was high 1500s), and only had a complexity of O(n) [21]. Moreover, a greedy search using travel times also yielded better QoR compared to a greedy search using Euclidean distances by a factor of 3. Figure 16 shows the improvement of our algorithm with respect to QoR.

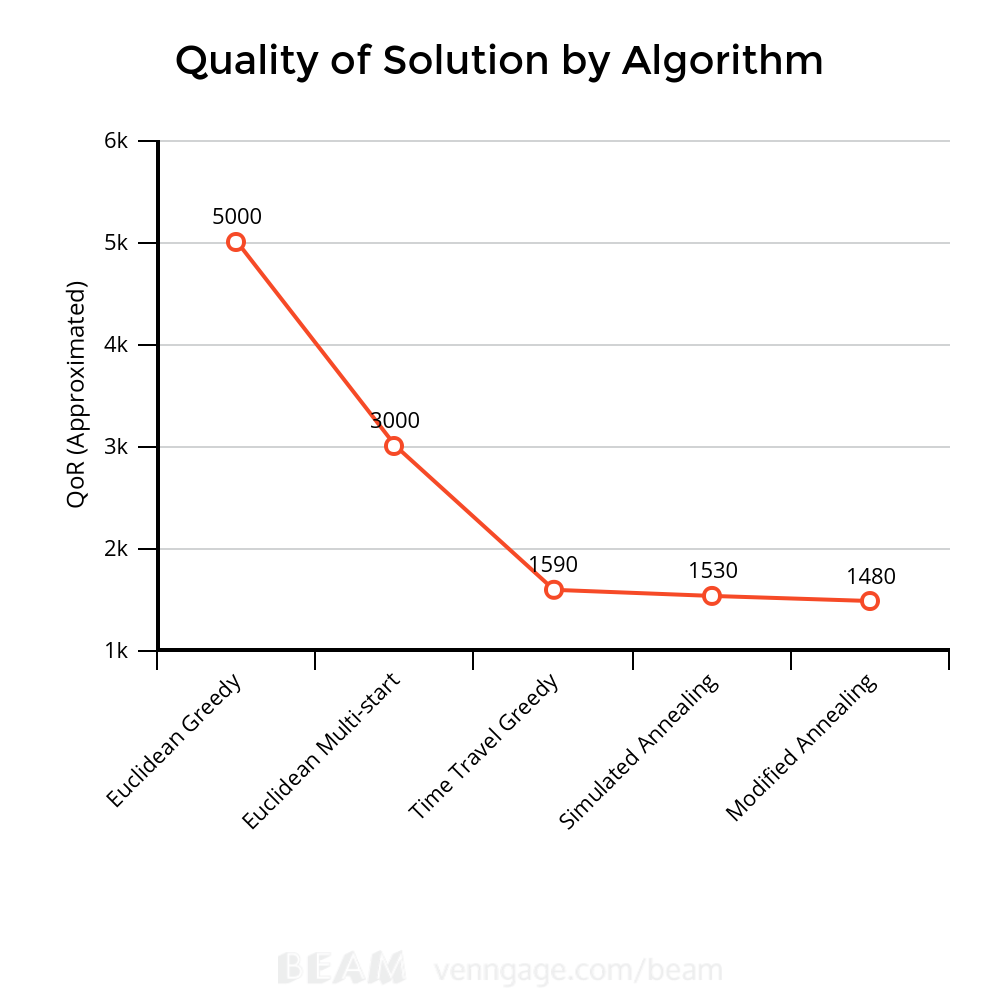


Figure 16: Graph comparing the progressive improvement of our courier route algorithm.

After the first path is found, the program uses simulated annealing to iteratively improve the solution by swapping two legal deliveries until the program runs out of computational time. For initial paths, the annealing searches a wider range of solutions by accepting suboptimal routes with random probability [22]. As better routes are found, the annealing process only accepts better routes and rejects suboptimal ones [22]. However, we found that increasing the time of initial annealing and accepting more suboptimal routes created better and more consistent solutions, as the solution was less prone to be stuck at local minima. By modifying the annealing process, our final solution had an average of 1487.61 QoR, as seen on Figure 17.

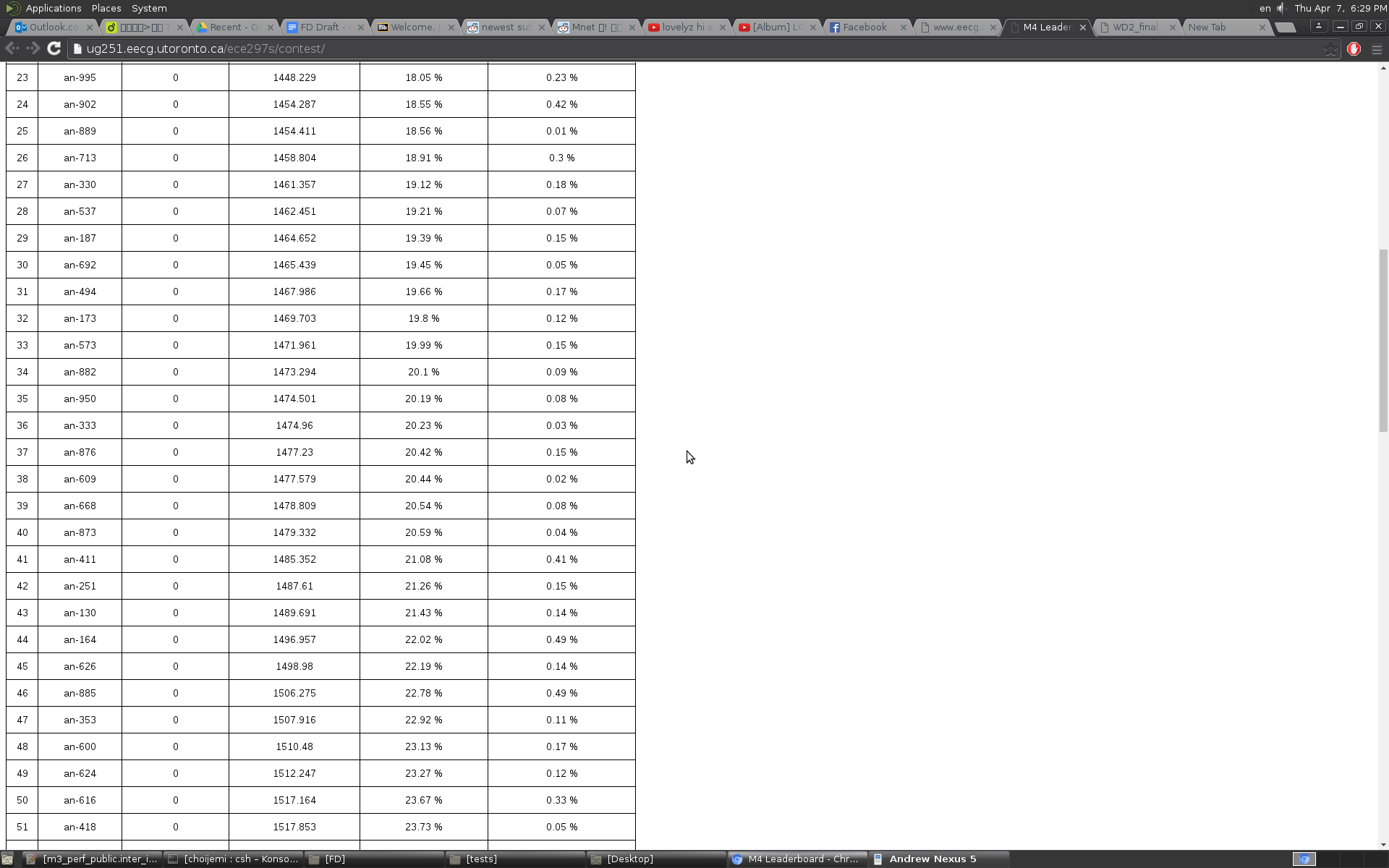


Figure 17: Partial screenshot of Milestone 4 Leaderboards. Our program found courier routes with an average QoR of 1487.61, and it was ranked 42nd out of 116 submitted programs.

### 

### 3.3 Data Structures

Our software uses efficient data structures that resulted in better average complexities, and thus, resulted in faster runtimes.

We chose vectors to store intersections, street segments and street information because its linear structure made it easier to code and understand. Additionally, accessing elements in a vector was very fast with a complexity of O(1) [23]. An unordered map also had an average complexity of O(1) when searching for an element, however, a vector outperformed an unordered map in this case as the average time for accessing for a vector was lower than searching an element in an unordered map. We tested with two data structures by accessing an element in a vector and searching an element in an unordered map one million times. The result is in Table 1.

Table 1: Comparison of accessing in vector and searching in unordered\_map

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Worst Time (seconds)** | **Average Time (seconds)** | **Total Time (seconds)** |
| Vector | 5.70x10-5 | 2.32x10-7 | 0.232 |
| Unordered map | 6.50x10-5 | 2.90x10-7 | 0.290 |

As it is evident from the table above, vector is a better choice as it outperformed unordered\_map by 20%.

Using vectors can be efficient if there is a direct relationship between the ID and index of a vector. However, if searching is needed to find the information, other data structures such as an unordered map can be more efficient [24]. As shown in table 1, searching for an element in an unordered map only takes 20% longer than accessing an element in a vector. This means that we can search for an element almost as fast as accessing an element in an unordered map. Our software used an unordered map of an unordered map to store the travel times between delivery points. We can use this data structure effectively when finding courier path as we can calculate travel time between 520 intersections in only seconds, which is much shorter than computing travel time for each intersection and adding them together, which takes 0.8 seconds.

However, an unordered map can be less efficient than other data structures when it is required to find the smallest or largest element in the data structure. In these cases, binary search trees or priority queues are more effective than an unordered map because these data structure are sorted in a way that allows easy access for smallest or largest elements [25]. Similarly, we used a priority queue to store wave elements when finding the shortest path between two intersections. Although a priority queue has similar complexity with binary search tree, when combined with searching and deleting, its “absolute runtime” is better than a binary search tree [26].

Additionally, we used a k-d tree to find a nearest intersection to a given position. If we used a vector instead of a k-d tree, the average complexity would have been O(n), instead of O(log n) [24][27]. The actual runtime for a k-d tree was 7.0x10-7 seconds per call, while searching a vector took 0.00211 seconds per call.

## 

## 

## 4.0 Lessons learned and Future Development

The biggest issue for our team was that we often could not meet internal deadlines on time. Consequently, this problem led to the lack of time to test the code and delayed sequential tasks. We believe that this issue arose due to miscommunication and differences in individual work habits.

For example, when we were working on Milestone 4, we decided to research different algorithms for solving the traveling salesman problem and write the code for it. However, this method was not effective as some members had trouble coming up with a feasible solution before the team’s internal deadline, and thus, hindered the team’s progress. At the beginning of the milestone, our group thought that this way was the best way to approach this problem, because we thought that it would have opened up more possibilities for a potential solution. However, due to differences in work habits, it was not the best approach for some group members. Some members worked better when the team continually exchanged ideas and pair coded, while some members worked better when working alone.

From this experience, we learned that it is important to respect other people’s work habits. In addition, we found it crucial to spend some time early in the project to analyze and identify these individual work habits. As the project progressed, we found that members who finished their part early got impatient at other members who were planning to work on their part at the last minute. Consequently, our team had to spend time away from the project to solve internal and emotional conflicts.

Moreover, to help prevent similar problems in the future, we could define and establish a concrete team structure and clearly state each team member’s expectation and roles within the team [28]. For instance, we could have assigned a specific leader for this project, who would be responsible for setting the team’s vision for Milestone 4. In addition, we could have assigned a facilitator to smoothen the communication within the team and help discuss realistic goals for each member.

While our team is satisfied with our final design, we would have liked to add an output window on the graphic interface. If implemented, this window will display information regarding points of interest and print route directions to enhance the usability of the design [29]. This output window will be placed under the mapping interface and eliminate the user’s need to switch back and forth between the terminal and map. Hence, we think that this feature will improve the mapping experience by helping users solely focus on the graphic interface.

## 5.0 Conclusion

This document analyzed how our mapping software makes it effortless for users to identify key features on the map. To achieve this, our map only shows crucial information depending on the zoom level and implements several features, such as the auto-completed search bar, that bridge the graphic interface with the terminal. Moreover, our map smartly uses a combination of data structures and algorithms to make complex calculations efficiently and to draw the map in a seamless and fluid manner. By designing our map around human factors, we have created a mapping software that is widely accessible and offers users a responsive and intuitive mapping experience.

### Appendix

Table A: Information that is visible to users at certain zoom levels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Zoom Level 1 | Zoom Level 2 | Zoom Level 3 | Zoom Level 4 | Zoom Level 5 |
| Highways | Motorways | x | x | x | x | x |
| Trunks | x | x | x | x | x |
| Primary Roads | x | x | x | x | x |
| Secondary Roads | x | x | x | x | x |
| Tertiary Roads |  | x | x | x | x |
| Motorway Links |  |  | x | x | x |
| Residential Roads |  |  |  | x | x |
| Unknown Streets |  |  |  |  | x |
| Features | Lakes | x | x | x | x | x |
| Islands | x | x | x | x | x |
| Greenspaces | x | x | x | x | x |
| River |  | x | x | x | x |
| Beaches | x | x | x | x | x |
| Parks | x | x | x | x | x |
| Streams |  |  |  | x | x |
| Buildings |  |  | x | x | x |

Additional Notes: the symbol x means that the corresponding feature is displayed on the map at the zoom level

Table B: A comparison between our mapping software’s performance and the speed benchmarks that the program was expected to operate at.

|  |  |  |  |
| --- | --- | --- | --- |
| **Performance Unit Tests** | **Execution time for Our Program** | **Time Constraint** | **Percentage Decrease from Time Constraint** |
| Intersections Toronto (Hard) | 12.34 seconds per 475 calls | 16.23 seconds per 475 calls | 24.0% |
| POI Toronto (Hard) | 3.03 seconds per 200 calls | 17.59 seconds per 200 calls | 82.8% |
| POI Toronto (Very Hard) | 5.91 seconds per 400 calls | 15.61 seconds per 400 calls | 62.1% |
| Intersections London (Hard) | 13.65 seconds per 100 calls | 16.43 seconds per 100 calls | 16.9% |
| POI London (Hard) | 3.28 seconds per 20 calls | 19.04 seconds per 20 calls | 82.8% |
| POI London (Very Hard) | 6.99 seconds per 70 calls | 14.89 seconds per 70 calls | 53.1% |

Note: Since our program operated much faster than these benchmarks, the percentage decrease from the benchmark was shown in the third column.

Table C: The table summarizes the average, minimum, and maximum durations to execute one function call on a map of Toronto (known as runtime), rounded to two significant digits.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Average Runtime** | **Minimum Runtime** | **Maximum Runtime** |
| Loading the map | 1.9 seconds | 1.9 seconds | 1.9 seconds |
| Drawing one frame of the map at the highest zoom level (most detail) | 0.091 seconds | 0.040 seconds | 0.101 seconds |
| Finding a path from one intersection to another | 0.014 seconds | 0.0068 seconds | 0.021 seconds |
| Finding a path from one intersection to a popular Point of Interest | 0.0024 seconds | 0.00095 seconds  (Esso) | 0.0063 seconds (Starbucks) |
| Printing out directions to a path | 8.8 microseconds | 8.9 microseconds | 8.6 microseconds |

Additional Notes: The runtimes of each function were calculated by running five sets of 200 function calls (excluding the load map function) for a total of 1000 function calls. The minimum and maximum runtimes were found by taking the lowest and highest durations of a set of 200 function calls.

Table D: Street Name Display Depends on the Zoom Level

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Zoom Level 1 | Zoom Level 2 | Zoom Level 3 | Zoom Level 4 | Zoom Level 5 |
| Display of Street Names | Motorways |  |  | x | x | x |
| Trunks |  |  |  | x | x |
| Primary roads |  |  |  | x | x |
| Secondary roads |  |  |  | x | x |
| Tertiary roads |  |  |  | x | x |
| Residential roads |  |  |  |  | x |

Additional Notes: the symbol x means that the corresponding feature is displayed on the map at the zoom level

### 

### 

### 

### Attribution Table

|  |  |  |  |
| --- | --- | --- | --- |
|  | Youngchul Han | Youngjo Kim | Jemin Andrew Choi |
| **1.0 Introduction** | P | RD, R, P, E | RD, FC, E, P |
| **2.0 Design Overview** | P | RD, R, P, E | E, P |
| **2.1 Visualization** | R, RD, E | P | FC, E, P |
| **2.2 User Interface** | E, P | RD, R, P, E | FC, E, P |
| **2.3 Special Features** | E,P | RD, R, P, E | FC, E, P |
| **2.4 Response Time** | E, P | P | R, RD, FC, E |
| **3.0 Technical Overview** | E, P | RD, R, P, E | FC, E, P |
| **3.1 Higher Level View** | E, P | RD, R, P, E | FC, E, P |
| **3.2 Algorithm** | E, P | P | R, RD, FC, E |
| **3.3 Data Structure** | R, RD, E, FC | P | E, P |
| **4.0 Lessons Learned** | R, RD, P | P | RD, FC, E |
| **5.0 Conclusion** | P | P | RD, FC, E |

R = Research

RD = Rough Draft

FC = Final Copy

E = Edit

P = ProofRead

### 

### References

[1] J. Nielson. (2010, June, 21). *Website Response Times* [Online]. Available: <https://www.nngroup.com/articles/website-response-times/>

[2] S. Smith and J. Mosier, Guidelines for designing user interface software. Bedford, Mass.: Mitre Corp., 1986

[3] J. Crampton. “Interactivity Types in Geographic Visualization”, *Cartography and Geographic Information Science*, vol. 29:2 pp. 85-98, 2002

[4] C. D. Morais. (2011). *Ten Things to Consider When Making a Map* [Online] Available: <http://www.gislounge.com/ten-things-to-consider-when-making-a-map/>

[5] A. Blair-Early and M. Zender, "User Interface Design Principles for Interaction Design", Design Issues, vol. 24, no. 3, pp. 85-107, 2008.

[6] D. Frigioni and L. Tarantino, "Multiple zooming in geographic maps", Data & Knowledge Engineering, vol. 47, no. 2, pp. 207-236, 2003.

[7] E. Wegman and Y. Said, "Color theory and design", WIREs Comp Stat, vol. 3, no. 2, pp. 104-117, 2011.

[8] M. Harrower and B. Sheesley, "Designing Better Map Interfaces: A Framework for Panning and Zooming", Transactions in GIS, vol. 9, no. 2, pp. 77-89, 2005.

[9] American Foundation for the Blind. (2016). *Creating Accessible Computer Applications* [Online] Available: <http://www.afb.org/info/accessibility/creating--accessible-computer-applications/25>

[10] Google. Google Maps [Online]. Available: <https://www.google.ca/maps/>

[11]I. Ceaparu, J. Lazar, K. Bessiere, J. Robinson and B. Shneiderman, "Determining Causes and Severity of End-User Frustration", International Journal of Human-Computer Interaction, vol. 17, no. 3, pp. 333-356, 2004.

[12] American Public Transportation Association. (2014). *Public Transportation Use is Growing - Here Are the Facts* [Online]. Available: <http://www.apta.com/mediacenter/ptbenefits/Pages/Public-Transportation-Use-is-Growing-.aspx>

[13] American Public Transportation Association. (2007). *Public Transportation: Benefits for the 21st Century* [Online]. Available:

<http://www.apta.com/resources/reportsandpublications/documents/twenty_first_century.pdf>

[14] K. Kwon, R. Shipley, M. Edirisinghe, D. Ezra, G. Rose, S. Best and R. Cameron, "High-speed camera characterization of voluntary eye blinking kinematics", Journal of The Royal Society Interface, vol. 10, no. 85, pp. 20130227-20130227, 2013.

[15]: ECE 297, Lecture #19, "Traveling Courier Part 3", 2016.

[16] ECE297. (2016). *Milestone 2 Drawing Maps* [Online]. Available:

<http://www.eecg.utoronto.ca/~vaughn/ece297/ECE297/assignments/a2/a2.pdf>

[17] D. P. Lanter, “User-Centered Graphical User Interface Design for GIS”, National Center for Geographic Information and Analysis Report 91-6, 1991. Available: <http://www.ncgia.ucsb.edu/technical-reports/PDF/91-6.pdf>

[18] A. Bach and S. Freelan. (2007). *Cartographic Rules* [Online].

Available: <http://www.wwu.edu/huxley/spatial/tut/cart/cart_rules.htm>

[19] K. Murota and A. Shioura, "Dijkstra’s algorithm and L-concave function maximization",Mathematical Programming, vol. 145, no. 1-2, pp. 163-177, 2013.

[20] Lecture Slides (Lec 16)

[21] Wolters, M. (2012). “A Greedy Algorithm for Unimodal Kernel Density Estimation by Data Sharpening”, Journal of Statistical Software, 47(6), 1 - 26. Available: <http://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1002&context=statspub>

[22] Wolfram MathWorld (2016). *Simulated Annealing* [Online]*.*  Available: <http://mathworld.wolfram.com/SimulatedAnnealing.html>

[23] cplusplus.com. (2016). *std::vector::operator[]* [Online]. Available:http://www.cplusplus.com/reference/vector/vector/operator[]/

[24] N. Josuttis, “The C++ standard library”. Reading, Mass.: Addison-Wesley, 2012.

[25] B. Sutherland, “C++ Recipes A Problem-Solution Approach”. APress, 2015.

[26] ECE 297, Lecture #16, "Faster Graph Search and Milestone 3 UI", 2016.

[27] Stanford University. (2014). *Assignment 3: KD Tree* [Online]. Avilable: <http://web.stanford.edu/class/cs106l/handouts/assignment-3-kdtree.pdf>

[28] J. Brown and G. Dobbie, "Supporting and evaluating team dynamics in group projects",SIGCSE Bull., vol. 31, no. 1, pp. 281-285, 1999.

[29] Nivala, S. Brewster and T. Sarjakoski, "Usability Evaluation of Web Mapping Sites", The Cartographic Journal, vol. 45, no. 2, pp. 129-138, 2008.