CS1013 Programming Project
Dublin Bus Data Visualisation
Group 8

Bergman, Edward Cosgrove, Conal Leung, Seng McCann, Samuel

April 2016

1 Structure

1.1 Architecture

The project followed the Model-View-Controller (MVC) architecture which separates the program into three interconnected sections. The model consists of data analysis, the view comprises of the user interface and the controller communicates with these two areas.

Isolating the project into distinct sections allows for individuals to work in their specified area and to better organise the work flow. Classes of the same area are managed into packages and sub-packages. Objects are designed with full modularity in consideration. Parameters and return variables are explicitly stated to allow for easy communication between classes.

The model was initially intended to have objects representing different stops and buses but the power of a well designed database allowed much faster retrieval data than Java could compete with.

1.2 Integrated Delevopment Environment (IDE)

The use of Processing's proprietary IDE is severely limited in terms of features and management. Small sketch projects are acceptable but for large-scale software development, Processing's IDE is a liability. Therefore, the decision was made to switch to professional Java IDE, namely IntelliJ IDEA.

1.3 Version Control System (VCS)

The utilisation of Apache Subversion (SVN) enabled project files to reside in a central repository. The project could be worked on simultaneously by group members by downloading and uploading files. A log is retained to record the changes each time a person uploads, edits or deletes a file.

1.4 Javadoc

Members followed official Java commenting convention to properly identify code. Javadoc tags consists of information relating to the author, date, version, parameters, returns and description. Correct documentation promotes good coding standard and better understanding of code.

2 Features

2.1 Database

2.1.1 Structured Query Language (SQL)

The original uncompressed data comprises of approximately 4 GB of .csv text files for one month. Utilising conventional methods to read small text files was not sufficient enough, given the large quantity of data.

Bergman researched about the use of SQL for the large amount of data. Performing a database configuration enables the load times to shorten exponentially when compared to basic input streams. The use of SQLite is optimal due to its operation of reading local database files rather than from a server. The text data is converted into .db files, allowing for fast access by the SQLite client.

The Data if further subdivided into tables within this database that contextualize the data, for example, a table that lists all the buses and the stops they visit. This allowed us to rapidly retrieve information needed without sifting through all 44 million lines of data. Further use was made of indexes that could be generated by SQLite to preprocess the data and how it is ordered to change what could be a 8-9 second query be reduced to 0.1 seconds on many occasion.

The database towards the end of the production cycle of the program began to supersede the initial design of the Model. The initial Model held objects that retrieved and stored their data. As this data grew, the objects became larger and larger, not only leading to be expensive in memory but also longer searching times. The database with its optimized search patterns and effecting preprocessing of data offered a much cleaner solution to what was becoming a time expensive issue for each query.

The final database file amounts to 5.5 GB of bus information covering the span of January. The total amount information contained totalled to 44 million lines. The load and query times were relatively fast, resulting in a fraction of a second, despite the immense quantity of data.

2.2 Graphical User Interface (GUI)

2.2.1 GUI Architecture

Samuel oversaw the back-end and overall aesthetic of most of the GUI. The GUI is built up of 2 main components and a controller. The Screen dealt with all the major GUI components such as the displaying the data from graphs and other objects created by Eddie, Conal and Seng. The Navigation Side Bar as we call it was the second dominating component to the GUI, consisting of multiple menu items that neatly stored all the different query options presented to the user. This is also where most of the GUI was controlled by the user.

There were two main obstacles that we wanted to overcome with the GUI. Firstly, we wanted to provide the user with options, lots of options. Yet the program still had to be somewhat accessible and user friendly. Samuel implemented Menu Items into the navigation side bar that hides the more complex queries and clutter of buttons away from the general user that might just wish to view some quick data through the route explorer, yet the more advanced user that might be using this program for a bit more has the option to open up into the menu items to gain more customisable data viewing.

Another important aspect of the GUI is that we wanted the GUI to remain responsive and slick throughout the user running the program. This meant we had to overcome the obstacle of making queries without our programming freezing. The solution we came to was multi-threading. Samuel taught himself the basics of multi-threading in order to allow different components of the GUI operate in parallel. By keeping all drawing and rendering in the main draw thread, moving all logic and number crunching for animations into a logic thread, and moving all queries into a query thread. No matter what the task in the program is, the GUI always remained responsive and would never freeze or slow down.

Due to the multi-threading of the program, we we're allowed to do some more complex animations and a lot of number crunching, while maintaining a very respectable frame-rate of 60 on the more powerful machines we use. Even for the weaker machines like Samuel's laptop, 30 FPS was still the expected value during high pressure moments in the program with lot's of things going on.

2.2.2 Widget

Leung was responsible for the creation of widgets, which consisted of buttons, single sliders, dual sliders, drop-down boxes, checkboxes and text input fields. The decision was selected to manually code all widgets to enable maximum compatibility and customisability. Each widget in general is constructed with maximum modularity. Basic parameters are required to construct the object and if necessary optional parameters can be passed, promoting the concept of polymorphism.

Small details are required for the widgets to enhance user experience. Subtle animations are considered, for example, the blinking caret of the text input field or small colour changes when the mouse hovers over. The role of mathematics is highly involved in the creation of certain widgets, as to enable a full modular design. As an example, for the drop down menu, each item to be added is arbitrary, hence the necessity of an expandable algorithm.

2.2.3 Animation

The bulk of the animations present in the program were done by Samuel. By implementing animation interfaces written by us, telling everything to animate itself was extremely straightforward and was an iterative process to just copy, paste the core animation structure of each object, and then write the different equations into each class for its animations.

When deciding the animation equation of movement to use, we were adamant to stay away from linear movement, and always implement some sort of acceleration and deceleration into our program. Sinusoidal and logarithmic equations we're the most commonly used.

2.3 Data Visualisation

2.3.1 Route Explorer

Bergman implemented the use of the Route Explorer into the program. While the GUI options allowed for very fluid navigation between the different Data Displays and Screens, lack of connectivity between the different Data Displays and the data they showed was apparent. The Route Explorer features easy-to-use navigation buttons to navigate general info for a bus and stop such as its route and what buses visit a stop.

For displaying more stop relative data, a Delay Clock was implemented to easily distinguish ideal times and worst times to get a bus based on its delay at a certain stop. Finally, with the use of some extra image buttons, automatically generated queries are made for the user based and what they currently have selected in the Route Explorer to further view data using the other Data Displays developed by the team.

2.3.2 Table

Leung worked on the display of tabular data. Data is shown in a column and row format with headers, providing basic textual information of specific queried data.

The table information consisted of arrays of strings in distinct columns. There is the ability to expand the information by scrolling, which was a considerable feature. The scroll feature is highly mathematical as it depended on the length of data being passed. The scroll feature adjusts in response to the data being input, by minimising or expanding the scroll bar scale factor Hence, it was necessary to work in ratios, formulae and proportions.

2.3.3 Graph

Cosgrove created the presentation of graphical data. The x and y axis numbering are displayed, along with the different position of points.

Additional features included the expansion of information when the mouse hovered over the points. The distinct x and y coordinates of the data is presented to the user.

2.3.4 Map

Leung managed the display of map data. The position of a bus or bus stop at a particular coordinate is displayed as a point. Clicking on the point reveals an information window, containing data related to bus ID, stop ID, longitude and latitude.

A map of Dublin is obtained from Google Maps as a .png image which acts as a background for the overlay information. The image is exactly the size of the screen (1280x720). The corner points of the image are calibrated to the exact coordinates of the map. Thus, the image acted as a small cartesian plane. Using proportional mathematics, data coordinates can be inputted and points are displayed at the correct position on the map.

3 Problems

3.1 Flawed Data

The dataset contains data which is informatively vague. Fields are not explained with descriptions from the data download source. Fields, such as Congestion and Journey ID are up to the user to interpret.

The dataset contains intermittent null values for some fields. The invalid results causes visualisation display errors with graphs and charts. The quantity of null values in the delay field showed zero delays, which is unlikely. Flaws also included journey ID's changing before the bus reaches a terminal and a bus not providing a data point when it had reached a stop.

3.2 Libraries

Compatibility issues existed with external Processing libraries. The GUI library *ControlP5* conflicted with the use of a PGraphics window. Drawing GUI elements using library is not possible using a PGraphics buffer, which causes issues with animations and drawing precedence. Coding GUI aspects manually enables maximum compatibility and control.

The map library *Unfolding Maps* is not compatible with the latest version of Processing 3.02. Its latest version is only functions correctly with Processing 2.00. The library was modified to work with Processing 3 but serious bugs and instabilities existed. The map did not enable other objects to be drawn over it and the map rendering mode reduced frames-per-second to non-functional levels.

A consideration was made to downgrade to Processing 2, but this conflicted with the use of a PApplet, a major propagating parameter throughout the program, in which Processing 2 did not use. If a downgrade occurred, major code restructuring was required, which was not worth a single library. Therefore, the decision was made to manually code the map, albeit, missing the useful functionalities of *Unfolding Maps*.

We decided as a team within the first week of the brief that we didn't really want to just write an application that uses multiple libraries written from other people. We were determined to write everything from scratch, and so in the end, the only libraries utilised in our project are the official processing libraries and SQLite libraries used in our DB.

3.3 Architecture Implementation

We adopted the MVC approach to designing our program which in turn allowed the team members to each individually focus on there modules for the project. Minor 'glue' was used here and there in the interest of time but perhaps our biggest Architectural design issue was in effectively utilizing the controller to act as a link between the Model and the View. While quick solutions were explored by using interfaces or global variables, each in turn had their downsides and a team decision was reached to forego the controller beyond telling each Data Display to retrieve the data they needed.