Programing Project Group 18

Outline

The objective of this project was to work as a team to produce a programme which implemented a user interface which could accurately represent a large data set of "houses sold in the UK". We feel that we divided the work evenly between us, each choosing to work on a task that we felt was best suited to our individual skill sets. In doing so we believe we were able to produce an application that is enjoyable, easy to use and highly functional, all within the relatively short time period of six weeks.

We wanted our project to be fluid, with the user seamlessly being able to make query after query without the presence of loading screens. However, we also didn't want to overcrowd the screen with information so we made a slide out menu. Additionally, we implemented Google's material design since we display the menu, map and graphs all on the same screen.

Organisation and Work Division

Our team showed good organization and communication skills throughout the project. Almost as soon as the project teams were announced we created a Facebook chat and began discussing the project while still on reading week. Our first meeting was held on the first Tuesday and a lot of the decisions regarding the project were made. It was decided against having a leader and instead we all took on responsibilities and discussed decisions, agreeing between us on the best course of action.

We decided to use eclipse instead of processing as we believed this more developed integrated development environment would allow us to code easier and keep our program neat. It also allowed us to use a subversion extension for eclipse which meant we could commit and update the code from the repository easily. We worked hard to maintain a coding standard so that the whole team would be able to read and possibly edit each other's code.

The team met up twice a week, once in the lab and once on a Tuesday morning in the glass rooms. Decisions were primarily made at these meetings, but we also stayed in constant touch through our Facebook chat and a google drive folder where we put notes from the meetings.

From the start of the project we were eager to split up the workload evenly. Even for the first week we split up the tasks so that everyone would get to do some small part.

From the second week on we decided to split the project into four broad sections which we would each take charge of. Hieu oversaw the back end, he had the job of creating our database, writing the queries and improving the processing speed of our project. Philip was put in control of the menu, he had to create the widget classes and the text widgets that would be used to call the different queries. Conor built graphs to display the information. These included bar charts, trend graphs and pie charts. Sulla worked on the user interface and the map.

However, we didn't entirely keep to these rigid areas preferring to remain flexible and help each other on difficult sections. We also cooperated on areas that overlapped multiple sections.

Features Implemented

Menu

The menu was designed so that it would slide out when a button is pressed and be minimised when the user clicks outside it. In this way, we could avoid the screen becoming too cluttered. The x positions of the menu and the widgets inside it will be rapidly increased or decreased to give the appearance of it sliding in and out.

The menu contained a collection of widget buttons. We used our previous widget classes as a template. The widgets were simply a rectangle with rounded edges, that called an event when you click inside it. These widgets were sorted into four different arrays. The first array contained widgets representing all the years. It is this array that is displayed when you first open the menu. This allows you to select the year that you want to make your query too. There is also an option to compare the years. The second array contained all the different fields that you can make your queries to (town, district, price etc.). The third array was for the different year comparison queries. The final array simply pointed to one of the other arrays. In this way, we could avoid large amounts of duplication, when drawing the widgets. However, we still needed to make an int to show what section of the menu is currently open, when handling mouse clicks. There are also back buttons located in the different sections of the menu to allow the user to easily transition back to the previous one.

Some of the query widgets automatically called a query. Old/new and property type showed pie charts to display the information, while price brought up a bar chart. However, others needed the user to enter a specific search, and for this we used a text widget class. The text widget class extends the widget class. With the text widget, the user must first click on it to select it. A '|' indicator will blink on and off in the widget box to show that it has been selected. Then you can type keys and your message will come up in the text widget. One string will store the entire input message. However, due to size constraints, we couldn't show the entire message at once. Therefore, we have a separate string which take the last 14 characters of the other string and displays them in the box. When the user hits the enter key, a query will be called to the search field and year previously selected. If the input is valid then the relevant information will be displayed on the graph.

Back-End

The database is an embedded SQLite3 database which contains UK land registry data from year 2010 to year 2016, parsed from comma-separated value files. Data from the embedded SQLite3 database is fetched by Sql20, a small database query library for JDBC-compliant databases. Sql20 allows much easier querying than the standard Java JDBC method and its tedious exception handling of Connection and ResultSet manipulation. Sql20 contacts the SQLite3 database using a SQLite3 database driver.

The DBQuery, houses functions used to query and fetch data from the database. The functions can accommodate "typical" queries which are used often by the main program and custom queries which can be a String of SQL language. The class can return different types of data, such as Lists of Strings and Integers or just Integers.

The LandData objects hold the properties of a single line of data from the comma-separated values. LandData is encapsulated and contains getters and setters used by the main program Sql20.

There is concurrency/multi-threading in the program which allows the SQL queries to be run in a separate thread to the main program and removes the issue of the query blocking the program from being used until the data from the query has finished fetching.

Graphs

In this project, there are three types of graphs that can be used to display data. Pie chart, bar graph and trend graph. The bar chart and the trend graph are similar in that they are both drawn into the Graph class whereas the pie chart is its own class entirely. All three graphs accept an array of integers of n length and 1 dimension and reposition and resize their segments (bars, pie segment, trend points) appropriately.

The pie chart is given a set of data to represent, a radius for the pie chart and an array of segments (pieces of the pie chart). Using the set of integers which it is supposed to represent, the pie chart calculates a size in terms of 2 pi for each segment. It assigns this size each segment and proceeds to draw each segment sequentially, not starting the next segment until the previous one is drawn. Each segment also has a hover effect. The percentage represented by the segment is calculated by dividing the size of the segment by 2 pi. This is displayed underneath the pi chart when you hover over a segment.

The bar chart is given a set of data to represent and a height/width. It generates several bars representative of the length of the integer array it should display. The width of the bars is calculated using the number of bars to be represented on the graph and the total width of the graph. The height of the bars is determined by the integer value given to each bar. This value is divided by the max value represented on the graph and multiplied by the total height of the graph. In this way, a bar with the value of five in a graph with max value ten will have a height half the height of the total size of the graph.

The bars are also made to transition, when a new height is calculated, their current height will gradually change frame by frame until it has reached its new value.

The trend graph is a series of data points whose x/y position is calculated similarly to the height of the bars in the bar graph. These points are stored in an array of data points inside of the graph class. Each point is also given the x and y position of the data point in front of it and can in this way calculate the slope between itself and the next point. Using this information, the data point can draw a line between itself and the next data point. Like the pie chart it takes place sequentially where a data point will only start drawing a line to the next data point when the previous data point has finished doing so, giving the appearance of a smooth transition

Maps

A map was incorporated to the project as an additional visual and interactive feature. It was treated as the background or base layer of the program. The Unfolding Maps library was used for the functionality of adding markers, however, the map itself was an SVG shape file we created. Because the files exist locally, it allows us to run the project smoothly whatever the conditions are.

Markers were created by taking in query data from the other classes, creating a LandData Array List of 20 random properties, determining the longitude and latitude of each entry, converting this to x and y coordinates on the map and drawing the marker itself. In order to do this, we made a CSV of all the counties in England with their corresponding longitude and latitude.

A few issues were faced to make the previously explained functions happen. Unfolding Maps doesn't support Processing 3 which means it created various bugs that was almost impossible to fix. But this enabled us to develop our own interactive features ourselves by only using the Processing library.

When the mouse is hovering over a marker, a popup dialog box appears which contains the price, address and last date sold. To fix the box on the screen and keep it there even when the mouse is not on hover, the user can click the marker. To close it, the user can press the marker again and it toggles to fixed and unfixed states. These popup dialog boxes can also be dragged around. This is useful when comparing selected properties side by side. They can be minimised by clicking the marker again and the user can completely get rid of them by clicking the close icon at the top.

The map goes far beyond just making query inputs. This branch of the project makes queries, visualises the data, and allows input and output interaction with the user.

Graphical User Interface

The idea proposed for the UI was to follow the Material Design developed by Google. It works well in our project considering we are displaying the graph, map and menu all at once. The design provides structure and organisation that keeps the three separate modules neatly unified.

We treat every layer as if they were paper cards. Hence, the grey shadow is added to every project layer by default. We used red and various shades of black and white for the colour scheme. The font mainly used was Open Sans. The positioning of each piece of text was precisely measured.

Buttons were also coded so that they output something when hovered or pressed. The menu button (three bars on the top-left) shows a 'Menu' label every time it's hovered over. When a bar is hovered over, information stats prints on the main footer and are sometimes also displayed on the graph itself.

Problems Encountered

Using subversion presented some difficulties. Several times people committed code only to have it overwritten by someone else's commit a few minutes later. Additionally, sometimes there was difficulty trying to merge our code. We overcame these problems by maintaining good communication. Often a commit would be accompanied by a message of Facebook and if someone knew they would be making lots of changes to all the classes in the program then they would ask everyone to avoid working on the code for a few hours. We also made sure to make regular commits so that everyone would have access to the same code. When working together in labs these problems were never severe as we could coordinate much easier.

Throughout our project we were confronted by all sorts of bugs. These included problems with the map unfolding, the graphs not transitioning smoothly and the queries not displaying the right data. As well, as the standard protocols for fixing bugs, we often helped each other out providing a fresh perspective on how to fix the issues.