PARTH Final Report

Raj Bariah Otonye Bestman Parth Chandratreya
Joe Groocock Asvini Patel Luke Taher

Contents

List of Figures

1 Introduction

1.1 Algorithm Decisions

The algorithms chosen by our team break down into two sections, sorting and searching, within these we have selected another two:

- Searching
  - Depth First
  - Breath First
  - A* Search
  - Dijkstra's Algorithm
- Sorting
  - Bubble Sort
  - Quick Sort

The reason we have decided to delve into two areas of algorithms (searching and sorting) is to allow the user to gain a broader understanding that algorithms are used everywhere and for different reasons. Within the two areas the reasons for the selection of two was again to show the diversity of algorithms that can be used for a single problem. However within both sections a simple algorithm and a more complex algorithm have been selected this is to ensure that a young user can still use specific parts of our application as well as older users that are required to understand more complex algorithms. Bubble and Breadth/Depth first give a foundation understanding of what sorting and searching are providing the user with an easy to follow algorithm. On the flip side A*/Dijkstras and Quicksort use more advanced techniques; but with the aid of our visualisation, users can simply see what these algorithms are actually doing. Allowing for younger users to develop a deeper understanding of common computational algorithms in addition to older users being able to refresh their understanding and/or learn these algorithms easily for the first time.

The visualisation of the algorithms was very important to our group because we wanted to represent real world examples to enable users to fully understand how they work. We came up with a few ideas before settling for specific ones for sorting and searching. The idea was to incorporate as a colourful GUI as we found teaching applications can often look dull and boring we wanted to take time into making the whole interface smooth and modern looking.

1.2 Searching

For the searching algorithms the London tube map was unanimously agreed upon because it allowed us to incorporate a lot of colours into the GUI whilst giving a well-known real world example of where to demonstrate a searching algorithm. Similarly to the sorting section of this application, the searching also allows the user to toggle between two (with the potential to expand) searching algorithms. The way the interface works is by the user selecting one station as a starting station (node), and then select a finishing station (node). Once this is done a dotted line will start to appear along the tube lines showing the searching algorithm traverse through this graph. Similarly to the planets, this section will also have a tutorial section which will display all of the information necessary to the searching algorithms, i.e. starting node, goal node and the step by step description of the algorithm.

1.3 Sorting

For the sorting algorithms we decided upon using the planets of our solar system as it allows for maximum diversity, meaning that we can switch between sorting via size or distance from sun. The interface for the sorting algorithms consists of two drop down menus at the top, one allowing the user switch between (initially, as there is the ability to add more sorts) bubble sort and quicksort. Whilst the other will allow the user to switch the sorting between size order and distance from the sun (again this is expandable to mass of planet and spin speed). In addition to the visualisation of the algorithm we have implemented a tutorial running alongside the interface. This is the bottom quarter of the screen which has a written description explaining the current state of the algorithm. This will run continuously, however there is a button to allow the user to toggle between the visibilities of the text.

1.4 Tutorial

For the tutorial mode of our program, we decided we need to break down all the algorithms so they would be easy to explain. They also needed to link and reflect what the visulisation was showing, otherise it would be pointless. We decided to implemtned a step system for the animaitions so we could also implemtn a step systtem for the tutoaril mode. The main descion behind a step tutoiral system was that big blocks of text would not be clear in exaplining and the user might be put off from the program as a result.

2 Software Design

2.1 High-Level Structure Overview

PARTH was built as a container for visualisation modules built in the `.vism` format for the application. The program loads each module in upon load and presents it to the user in a list-view menu. Any of the modules can loaded and run as if they are a separate application. This modularity concept makes it very easy to deploy new modules and update existing ones.

There is a simple pipeline that every module follows which contains a series of steps, all of which can be handled by the module:

1. Application initialisation
2. Module load
3. Module resources load
4. *Main menu opens*
5. Module `onLoad()` is called

2.2 Module Format

The module format for this program consists of a vism archive compressed with zip compression and containing the file structure demonstrated in figure 1, right.

The most important file within the module is the `module.json` as it defines the location of the Controller class to load and manage the module and all the resources including the banner image, title and description all to be displayed on the menu screen.

The entry-point for each module is the Controller class which is loaded upon program initialisation. The controller is responsible for managing everything that the module itself requires including initialising the View, loading any resources required from the vism file and performing all the logic in the visualisation.

```
module.vism
├── module.json
├── res/
│   ├── banner.png
│   └── resource-file.json
└── org/
    └── company/
        ├── ModuleController.class
        └── ModuleView.class
```

Figure 1: Example `.vism` structure

A Controller should implement the `BaseController` interface to provide the required methods to the application for interfacing with the module internal workings. An abstract implementation of `BaseController` is also provided in the form of `JsonController` which simply handles some basic loading of the base resources from a JsonObject loaded from the `module.json` file of a module (title, description and banner image) required for the module to display in the menu; the `loadResources()` function is declared abstract so each module still has to handle that. This class is used over the `BaseController` internally when loading controllers from a module for increased simplicity when loading. Any module loaded from an external `.vism` file must use `JsonController` as the parent to the controller class.

2.3 Module Metadata File

Each module must have a `module.json` file in the root of the vism otherwise it will not be recognised or loaded as a valid visualisation module by the program. As a minimum, the file should provide at least every *required* field detailed by the example below.

```
{
    "name": "Module Name",
    "desc": "Module description",
    "banner": "banner-image.png",
    "load": {
        "package": "org.company.vism",
        "main": "ModuleController"
    },
    "res": {
        ...
    }
}
```

- The entries `name` and `desc` are strings used to display on the banner image in the main menu screen. **Required**
- `Banner` is the name of a file to use for the banner image, stored in the `res` directory, within the module vism file. **Required**
- The object `load` defines the `package` and class name (`main`) of the controller used to initialise the module. **Required**
- The `res` object defines resources required by the module. More information about this can be found in section 2.4

2.4 Loading Module Resources

After initialisation of the controller the program attempts to load any resource files specified in the `module.json` file, under the resources section. *Note:* This section is completely optional and can be omitted if no resources are required for the module.
Each resource should be defined as an object with values for the file name and the type of data it contains. The resource files defined in this section are loaded with the root of `res/` within the `vism` file. An example of the resource section from the `module.json` file as demonstrated below:

```
"res": {
    "an-image": {
        "file": "images/img1.png",
        "type": "Image"
    },
    "some-data": {
        "file": "datafile.json",
        "type": "JsonObject"
    }
}
```

A file of any type can be loaded and parsed manually but some, including but not limited to the ones below, have special cases and are parsed slightly differently upon loading for convenience, otherwise the `InputStreamReader` is passed.

- JsonObject
- JsonArray
- Image
- Zip
- Text/Plain

In the case of a different file type needed, the user can define their own by declaring the 'type' as whatever they require it to be and the InputStream to the file will be returned instead which the developer can handle themselves to load the resource any way they choose to.

The program will accumulate a list of loadable resources (throwing errors for each that cannot be loaded) and pass them to the controller via `loadResources()` in a HashMap, each accessible by the key used to define the resource in the `module.json` file.

2.5 Module View

In PARTH we enforce a *Model-View-Controller* style of programming. As the Controller is loaded upon program initialisation, it is then up to the controller to manage it's assets including the View. We provide the class `VisualisationView` as a basic GUI, split into 3 panels. The upper and lower panels are designed to house GUI elements like buttons, informative text etc and the centre to contain the main visualisation.

This implementation, whilst advised, is not enforced and therefore can be replaced with any view that implements the methods from `BaseView` given that the controller handles the entire visualisation, including the View.

The provided `VisualisationView` is more of a guideline example of what could be used, however the only requirement the visualisation must adhere to is to provide a `Scene` through the `getScene()` method from `BaseController`. The two supplied modules both use this view class as their implementation (See section 3.2 for more) but any other implementation could be used as an alternative.

2.6 Algorithm Streams

During the initial planning phase we decided the application should have a standard method of interacting with algorithms, allowing for simple *state-based interpretation* of the data produced by the algorithms. From this we devised `AlgoStream`- a standard interface we could use with any algorithm to get state information out.

With all the algorithms used in the application, they are designed from the ground up to accept generic arguments wherever possible. In our implementations, both the `SearchStream` and `SortStream` implementations are generified to as loose a bound as possible for more flexibility. Adding this slight extra overhead to the programming allows for a much wider versatility later if the search or sort functions need to be used for anything else then they should 'just work'.

In the final version of PARTH, with the included modules, there are two complete `AlgoStream` implementations: *SearchStream* (Section 2.6) and *Queck/BubbleSortStream* (Section 2.6) that are demonstrated by the included modules.

SearchStream

SearchStream was the first implementation of `AlgoStream` that we used in the project code. It is a highly generified class that will perform search across any graph and can be customised greatly with one of many factors. Although an almost complete implementation of a generic search algorithm, `SearchStream` is declared `abstract` as it is missing a couple of key elements to perform a complete search, namely a *frontier set* to store the list of 'to be explored' elements and functions to calulate 'cost' and 'heuristic' values for each of the explored nodes in the graph.

An important reason for the blatant omission of a frontier implementation is to allow for search variations that rely on different implementations for a frontier set.

As shown in figure 2, the 4 different search algorithms use a variety of data structures to represent the frontier set. Each search must also be initialised with 'cost' and 'heuristic' functions appropriate to the graph that is to be searched provided the algorithm calls for them (breadth-first and depth-first search don't have a heuristic).

Algorithm	Frontier Set
A⋆ Search	Priority Queue
Dijkstra's Algorithm	Priority Queue
Breadth-First Search	Queue
Depth-First Search	Stack

Figure 2: Various frontier types used

SearchStream can have a variety of states, represented by `SearchState` objects. This class contains the following elements:

– A list copy of the frontier set, ordered according to how it is stored in the SearchStream.
– A set of nodes already visited by the search, in no particular order (by definition of a set)
– A flag, set true if the search is complete and has found a path.
– A list of successive nodes that make up a path created by the `SearchStream`

At every stage of the search, as each node is expanded, a new state is created to show the change between the frontier set with the newer nodes added from the recently expanded frontier head node.

QuickSortStream & BubbleSortStream

Whilst the algorithms for search are simpler than those of search, the state representation is quite the opposite. We provide a base interface `SortState` that simply represents the list of items and a flag to determine the completion state of the sort. Built on top of the interface are a hierarchy of `SortState` subclasses that add extra information to the base state class.

– `ListSortState` is a basic 1-to-1 implementation of `SortState`
– `PartitionSortState` provides the same functionality as `ListSortState` along with list indicies for a subsection of the list as well as a pivot element.
– `CompareSortState` extends `PartitionSortState` and adds indicies for two elements being compared and a flag for whether they should be swapped.

Each `SortState` above `ListSortState` also have reference to the previous `ListSortState` produced by the stream and from every state can be accessible by `getList()`. The list provided is always a representation of the list state before the action being represented is performed.

Not all of these states are usable for every type of sort but they provide a wide range of representation that should be applicable to every commonly used sorting algorithm. `BubbleSortStream` doesn't require `PartitionSortState` as it is a non 'divide and conquer' algorithm whereas `QuickSortStream` does. The Planets visualisation (See section 2.7 for more) is flexible in that it can correctly animate any of the above mentioned states without knowing about the sorting algorithm and regardless of the source.

2.7 The 'Planets' Module

With this visualisation we wanted a heavy focus on visual appeal and interactability. In this module, the user is presented with an interactive solar system with realistic looking spinning planets. Any planet can be selected and zoomed towards showing more information and stats about the particular planet. Many small features were added to this visualisation to add just that bit extra appeal.

- Rotating planets around their true axis
- To-scale rotation speed, relative to the rest of the planets
- Accurate sizing of planet models, relatively scaled from their true size
- Twinkling stars in the background for an extra ambient touch
- Bump-map textures for planets to add an extra dimension of depth

All animations and tranisitions provided by JavaFX made it very simple and quick to nest and layer many complex transitions together to create a slick looking visualisation.

Planet data and textures were all loaded in via the resources system that the program provides and was used accordingly to both render the planets (and other solar system elements) on screen and also perform the sort. The attributes defined in the `planet-info.json` were used by the renderer to accurately depict the sizing, rotation angle and speed. The same data was also input into heuristic functions provided to the `SortStream`s through a series of Java 8 functional interfaces. This data and code re-use made it simple to represent the planets in code via a single `Planet` class containing all the relevant data that was used by both the `PlanetRenderer`s and the `SortStream`.

Demonstrating the search states in this demo was simple; all states are represented as a series of animations, moving the on-screen elements about. Comparisons are demonstrated by bringing the two planets to be compared forward, explaining to the user that the planets are being compared, by what and how, then returning them to their initial positions. Swapping is a similar story, performed directly after comparison, planets are moved along a semi-circular path swapping the objects.

All-in-all this was a relatively simple visualisation to implement, fitting for a simple algorithm too, using primarily animations and transitions to manipulate the solar system by moving planets and the camera to effectively demonstrate sorting algorithms.

2.8 The 'Tube Search' Module

Much of the complexity within this module spawns from the graph we chose to demonstrate search using. The choice of London Underground was *the* perfect choice of map to demonstrate seach but with a cost of a complex implementation of rendering the entire thing. Initially we planned to just use a static image to represent the map and it would have been siginficantly simpler and time-saving however this would not provide any flexibility in dynamic animation or even representation of the search states over the map. The only option we were left with was to render the entire map by hand, constructing the location data of all stations, lines and everything between all by hand. The `station-info.json` file is way over 6000 lines and was completely written by hand thanks to a dedicated couple of team members.

Positioning each station onto the screen is trivial: draw a circle at the location of the station. Connecting the stations together with the provided data that defines connections of stations via which line is similarly a simple task. When it comes to attempting to accurately replicate the distinct design of the actual London Underground tube map with every curve and connected line the trouble arises. We managed to alleviate much of the complication by specifying where each line bends and drawing in those bends into the lines. Many of the more intricate details never made it into our representation of the map due to time and complexity constraints.

We had the fortune of being in the same position with this visualisation as with 'Planets'; the data used to define the locations and the connections of each station can be used directly in the `SearchStream` as a constructed graph to perform the search algorithm on. Loading in the data from the massive json file, each station is constructed into a `TubeStation` object representing name, location and connectionse (including all line colour and curve data) about a station. This graph of objects is provided to the `TubeMap` renderer to produce a pretty and colourful diagram which is displayed to the end-user that is interactive, clickable, zoomable etc. It's also piped into the `SearchStream` instance which chosen by the user along with stations to generate a path between to generate a stream of `SearchState`s.

Thanks to the direct integration between the graph and renderable map, the stations referenced in the states can be used to find stations in the map and update the visual state directly without the need for complex mapping of objects.

2.9 Logging

One of the debugging and testing utilities that was added to the application was an advanced logging system. This is superior to printing to the stdout console for a couple of reasons:

- Log entries can be tagged with a severity level ranging from `FINEST` to `SEVERE`
- Logs can be filtered by a minimum level to strip the irrelevant messages and resolve the offending bug/issue quicker
- Logs are persistent and stored to file so can be shared for debugging assistance and are also useful as a debugging tool to look backwards at previous runs of the program
- Provides an easier method for users to submit bug reports by just providing the log that will contain a good amount of information about how and where the issuee occured in code

Log entries can be added from anywhere, called from the static instance located in the main PARTH application instance, including caught and uncaught exceptions from any thread within the application.

The choice to add this function to our application was invaluable and has proved a very useful debugging and in general a great development tool all around.

3 Visualisation Design and HCI

3.1 Initial thoughts

At the start of the project we knew what application we wanted to make, so we had a clear idea of what the design and visualisation requirements would be. We wanted the product to have enough colour and flare to attract younger users as well as being usable and interesting enough that older users could use the application without thinking it looked childish. This is where we came up with the ideas of our interface, which consists of the solar system for one section and the London Underground Tube Map for the other. These two ideas satisfied the initial requirements of being colourful, interesting and most importantly they are relatable real world examples that will help the user fully understand the algorithms that they are going to be demonstrating.

3.2 Reasoning for the design

Having a design where the program was split into two sections ensured that we could very clearly show how this application has two distinct areas (as of now). We wanted users to see that they could learn either sorting or searching algorithms by navigating into different areas of the program. This is why the menu was designed as purely two large buttons for the user to select between the two sections.

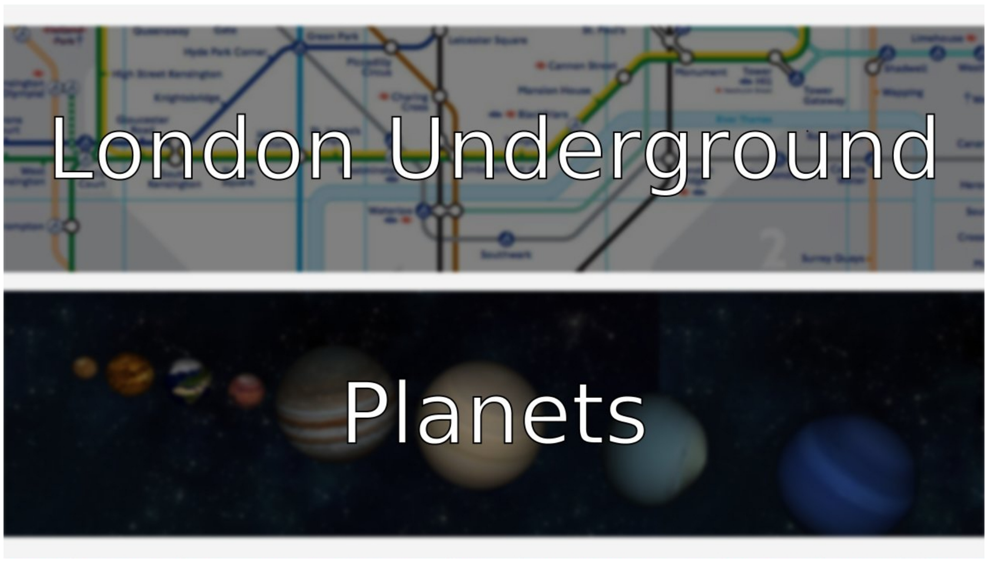
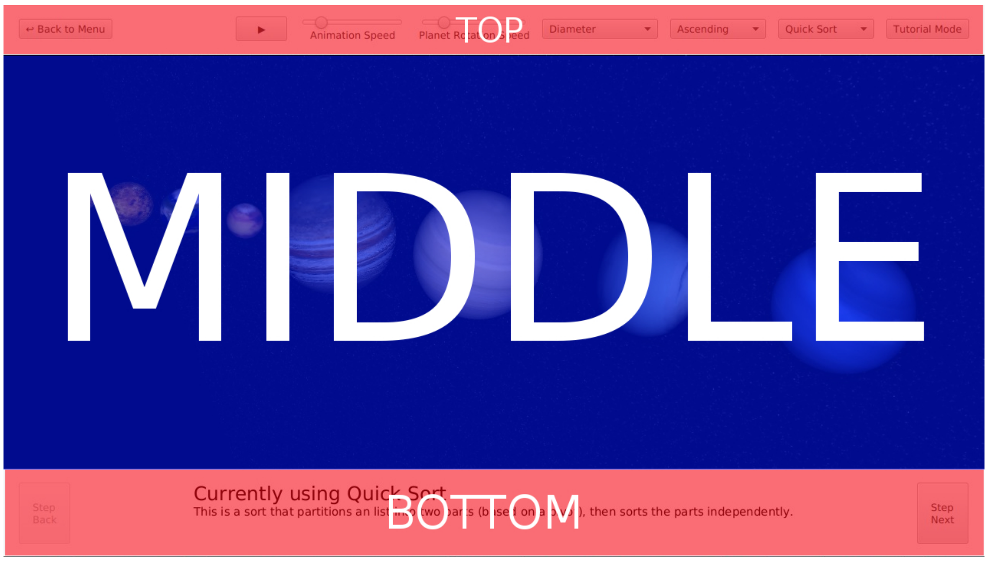


Figure 3: The main menu with the buttons (set on 'mouse hover' mode)

Once a selection was made, we wanted the application to look consistent, hence we made the layout a requirement which was split into three sections, an upper section, mid (main) section and a lower section.



Currently using Quick Sort
This is a sort that partitions an list into two parts (based on a pivot), then sorts the parts independently.

Figure 4: The Planet visualisation layout with an overlay explanation

The top section has the 'Back to main menu' button located in the top left hand corner, followed by the the settings and selections that you can add to alter the way the specific algorithm works e.g.

- being able to switch between the different searching or sorting algorithms
- being able to change speeds and to be able to completely run through the whole algorithm.

We felt it was important to keep all of this information at the top of the page so that is easily accessible but it doesnt intrude on where the visualisation is.

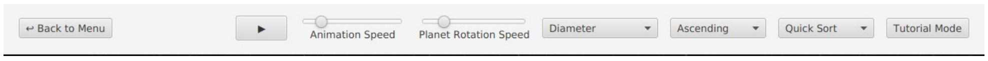
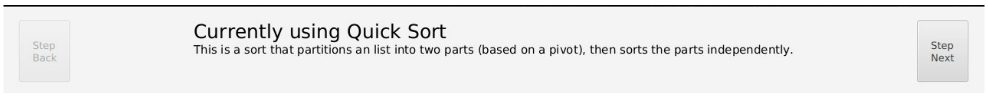


Figure 5: An example top section taken from the Planet visualisation

The middle, or main, section of the display is the interesting section, this is where the visualisation for the algorithms are displayed the tube map for the searching and the planets for the sorting. This was in the middle of the window and was approximately 60% of the screen. We felt this desicion was very important as this is the main part of the program hence having a larger display for it would allow our users to be able to view the algorithms in action with more ease.

The bottom section, similarly had a larger section as this is where we were running the tutorial section. The reasoning for the sizing of this section and location was so that people would be able to easily read the tutorial information, but when its turned off, the area isnt intrusive on the window. However, even when tutorial mode is off the stepping buttons which allow the user go through the algorithm step by step are still active and available for use, therefore having this larger area at the bottom is still necessary.



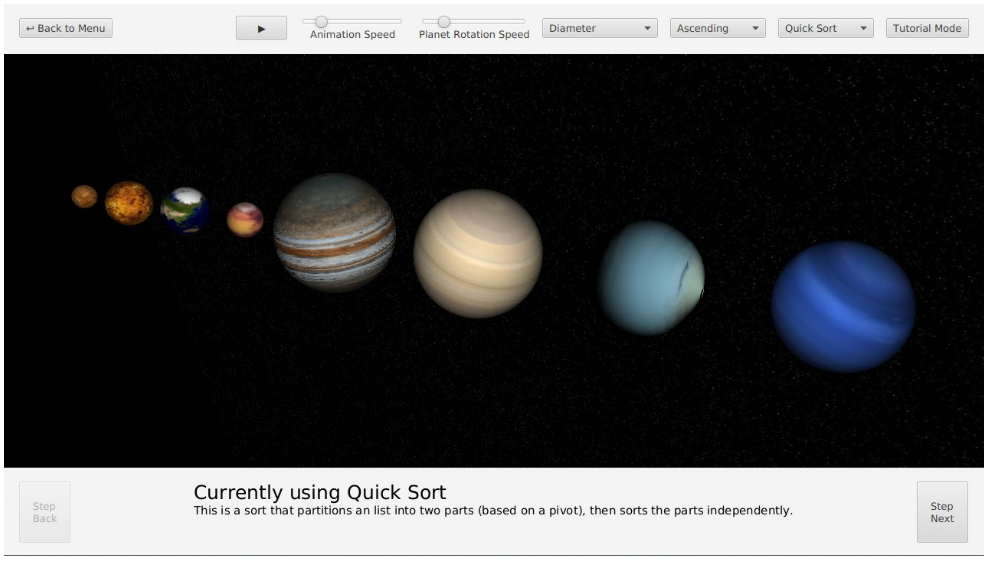
Currently using Quick Sort

This is a sort that partitions an list into two parts (based on a pivot), then sorts the parts independently.

Figure 6: An example bottom section taken from the Planet visualisation

3.3 Breakdown

- Main menu appears
  - Two large boxed are aligned centrally on the screen, one of the Planets and one of the London Underground.
- User can select either Planets or London Underground
  - When user hovers over either of the buttons the image blurs and text appears that reads either Planets or London Underground
- User can run through the sorting algorithms with the Planets visualisation.
  - The user can visually see the planets being sorted using different algorithms, but the user has the ability to do some customisation here with changing the sorting algorithm with a drop down menu, they can also toggle on and off the tutorial the tutorial appears in the section that allows the user to press previous or next in terms of the algorithm running. In addition to this there is a back button to return to the main menu.



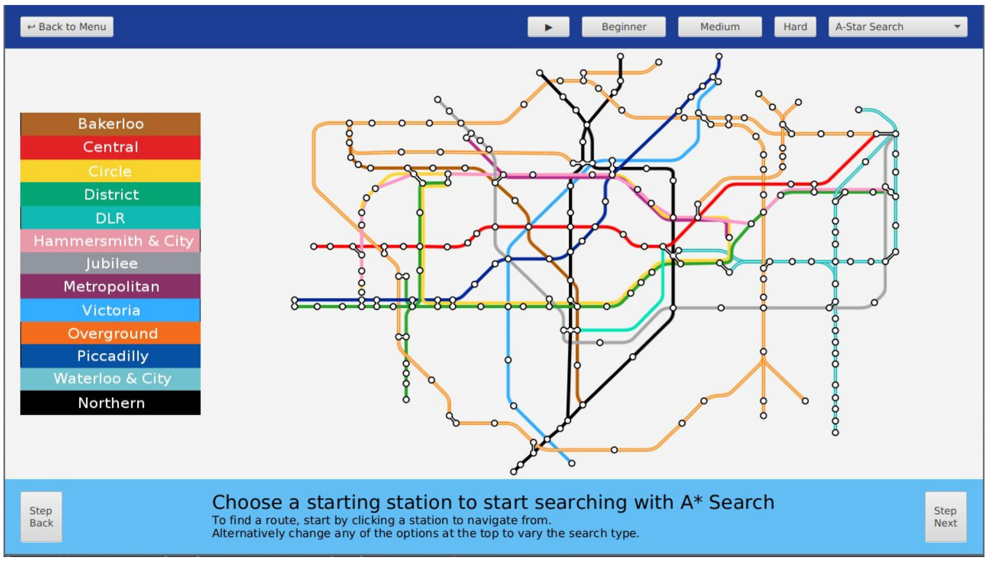
Currently using Quick Sort

This is a sort that partitions an list into two parts (based on a pivot), then sorts the parts independently.

Figure 7: Planet visualisation

- User presses Next
  - The planets will move to the nmext stage in the sorting algorithm and if the tutorial text is toggled to be on, the correct text will appear to explain this step to the user.
- User selects Previous
  - Similarly to previously the algorithm will go back a step, this will be reflected by the planets moving to the previous state in the search and the tutorial text explaining that state.

- Users can run through the searching algorithms with the London Underground visualisation
  - The user can visually see the searching algorithms working using the London Underground as the graph with the stations as the nodes. Similarly to the last one, the user can customise the searching in addition to there being a back button to return to the main menu



Choose a starting station to start searching with A* Search
To find a route, start by clicking a station to navigate from.
Alternatively change any of the options at the top to vary the search type.

Figure 8: London Underground visualisation

- User selects starting station
  - The user will have to select a station that they want to start the search from, this station colour will then change to green
- User selects ending station
  - The user will then select a station that is the end node for this search. The station colour will then change to red

We have specifically made the program run like this as it is intuitive and allows the user to navigate between different areas with fluidity. The other benefit of the program having a main menu means was so that in the future if we wished to we would be able to create a new button for another section of the program.

3.4 Human-Computer Interaction

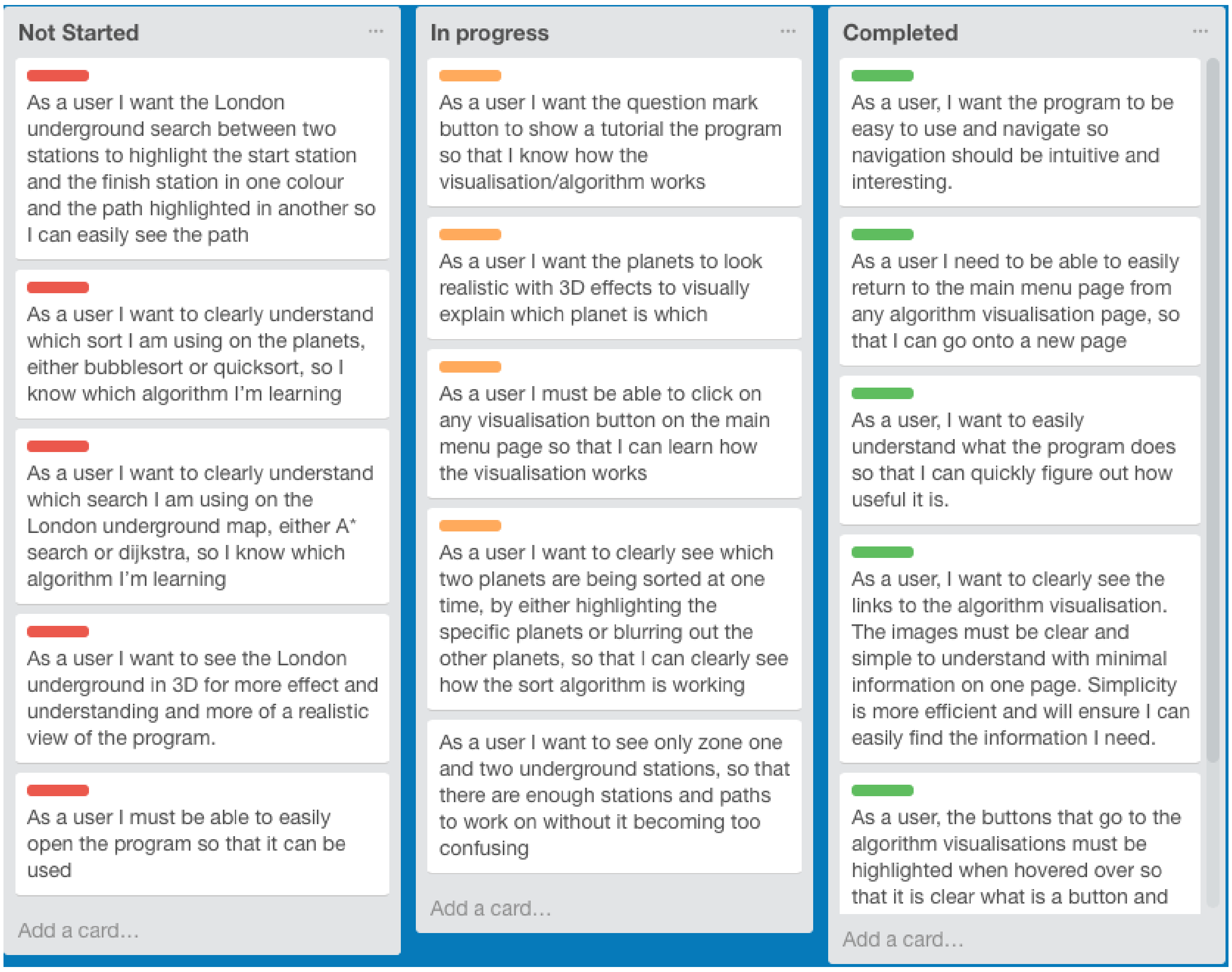
The user can interact with this program through the use of different peripherals. We have tested the product with the conventional Keyboard and Mouse as well as testing on Microsoft Surface using the touch screen. The reason that we have tested it with a touch screen is because in schools, where our target market is, this product may be used on interactive white boards. The functionality with the mouse/keyboard is identical to the touch screen, this is because the buttons are large enough that the user will have no trouble selecting anything and everything on the screen is clearly labelled. We created the GUI to work instantly, as everything worked through action listeners this ensured the program to work fast and ensured fluidity in the interactivity.

Seeing as the GUI was the most interesting part of the application, this is the area that we would see the most significant developments in week to week. Each iteration of the program, as a team, we would see advances in the usability and interactivity of the application. This is what motivated us to create such an intuitive and seamlessly integrated GUI to the backend of the application. The display shows this as we worked hard to ensure that the users would not see a cluttered image, but a simple system thats usability was simple to understand. This was a goal of ours that was set out from the start, we wanted to ensure the easiest interaction for the user, this is to mitigate the possibility of anyone getting confused using the system.

4 Software Engineering and Risk Management

4.1 Development Method

When deciding how to plan our project we considered all of the software development models. We soon realised that using a planned development model (i.e. waterfall) would not be practical as the process struggles with rapid software development and changing requirements. As we knew the requirements were most likely to change as we progressed through the project, we decided our software development model would need to allow us this flexibility in requirements. In the end we decided to use an agile software development approach, specifically scrum. This still ensures good quality, readable code, but with more informal communication within our team and extensive monitoring of development. We used the functional requirements that we wrote in the specification to create user stories. These user stories were created with the format of: As a user, I want so that this format ensured that user stories were simple to understand and were from the point of view of the user, which meant that we had to think about how the user would view and interpret the program. These user stories were placed on the Trello board as a product backlog so that they were easily accessible to all team members. The Trello board made the view of the user stories clear and we were able to colour code them based on whether they were completed, in progress or not started.



Not Started

- As a user I want the London underground search between two stations to highlight the start station and the finish station in one colour and the path highlighted in another so I can easily see the path
- As a user I want to clearly understand which sort I am using on the planets, either bubblesort or quicksort, so I know which algorithm I'm learning
- As a user I want to clearly understand which search I am using on the London underground map, either A* search or dijkstra, so I know which algorithm I'm learning
- As a user I want to see the London underground in 3D for more effect and understanding and more of a realistic view of the program.
- As a user I must be able to easily open the program so that it can be used

Add a card...

In progress

- As a user I want the question mark button to show a tutorial the program so that I know how the visualisation/algorithm works
- As a user I want the planets to look realistic with 3D effects to visually explain which planet is which
- As a user I must be able to click on any visualisation button on the main menu page so that I can learn how the visualisation works
- As a user I want to clearly see which two planets are being sorted at one time, by either highlighting the specific planets or blurring out the other planets, so that I can clearly see how the sort algorithm is working
- As a user I want to see only zone one and two underground stations, so that there are enough stations and paths to work on without it becoming too confusing

Add a card...

Completed

- As a user, I want the program to be easy to use and navigate so navigation should be intuitive and interesting.
- As a user I need to be able to easily return to the main menu page from any algorithm visualisation page, so that I can go onto a new page
- As a user, I want to easily understand what the program does so that I can quickly figure out how useful it is.
- As a user, I want to clearly see the links to the algorithm visualisation. The images must be clear and simple to understand with minimal information on one page. Simplicity is more efficient and will ensure I can easily find the information I need.
- As a user, the buttons that go to the algorithm visualisations must be highlighted when hovered over so that it is clear what is a button and

Add a card...

Figure 9: Using Trello to keep track of User Stories

An important part of the agile process is working simultaneously on different aspects of the program. This meant that daily scrum meetings were essential to ensure everyone was aware of what the others were working on, each member giving a quick description of their current working status. A scrum master is chosen to keep track of sprints and their deadlines. The sprints were given estimates of how long they would take to implement, the scrum master tried to make sure we stayed within these time frames. What also appealed to us about using scrum was the fact that prototypes were created with every iteration, this gave us an idea of how much we had done and what there was left to do or improve on. Scrum allowed us to decide which user stories we wanted to implement and if we didnt have time to implement extra user stories, we still had a working prototype of the program readily available. Our main priority was getting working software as quickly and efficiently as possible.

Within the first 2 weeks we focused on planning the design of the class structure of the project. We created an initial design of the class diagram to help us get an idea of how the classes would work together and the structure of the code. We used an MVC model for our class structure, which was shown in our class diagram. The number of classes has increased considerably since our initial design this showed us our progress, but the overall design of the classes has still remained intact. We were able to use intelliJ to create UML class diagrams from the classes we have made. This was great to see how our project has expanded and developed. For the classes that are part of the model of the MVC design we can see that the main classes have remained the same. We have kept AlgoStream as our main interface, which provides SortStream and SearchStream their methods. AStarSearchStream, DijkstraSearchStream, DepthFirstSearchStream and BreadthFirstSearchStream extend the super class SearchStream. We added a SearchCollection that is an interface for all of the data structures such as stack queue and priority queue.

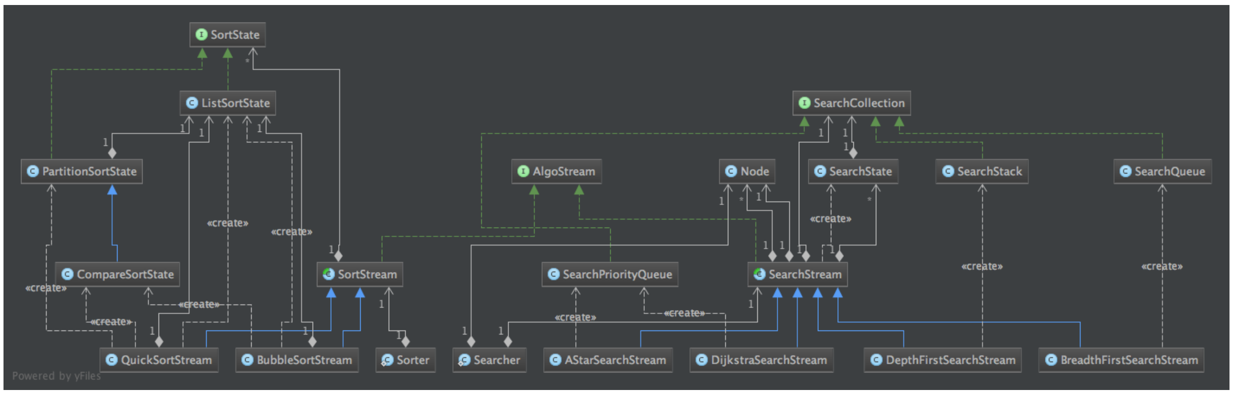


Figure 10: Model aspect of the MVC of our current set of classes in our program

For the classes that are part of the view and controller of the MVC design we can see that for the London Underground visualisation, the tube search view and controller have stayed, but now TubeSearchController loads the JSON file containing all the coordinates and connected stations, and the TubeStation class contains the data after it is loaded. All the original classes in the initial class diagram have remained in our project, and so our initial design was extremely useful in creating the structure of the whole program.

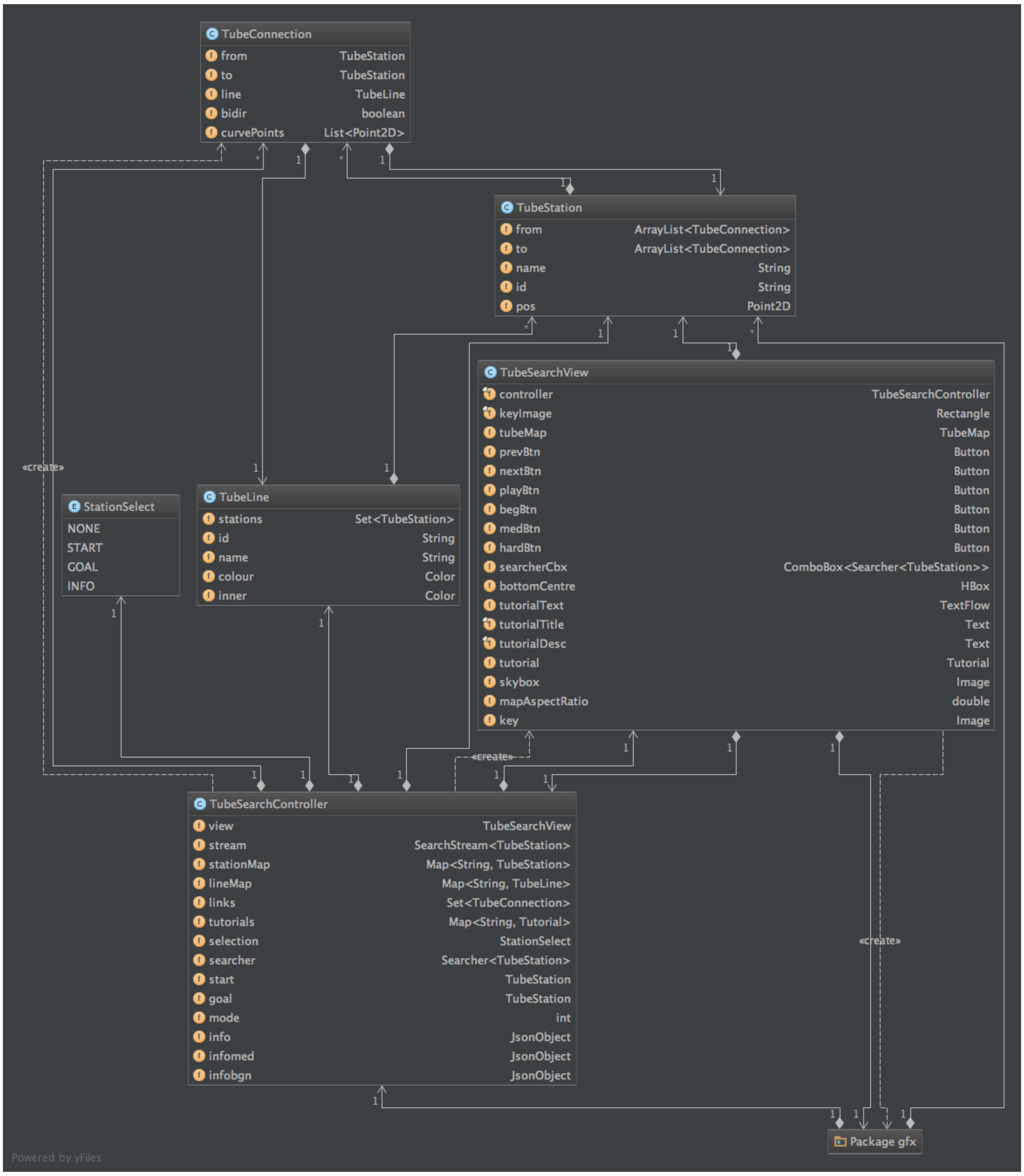
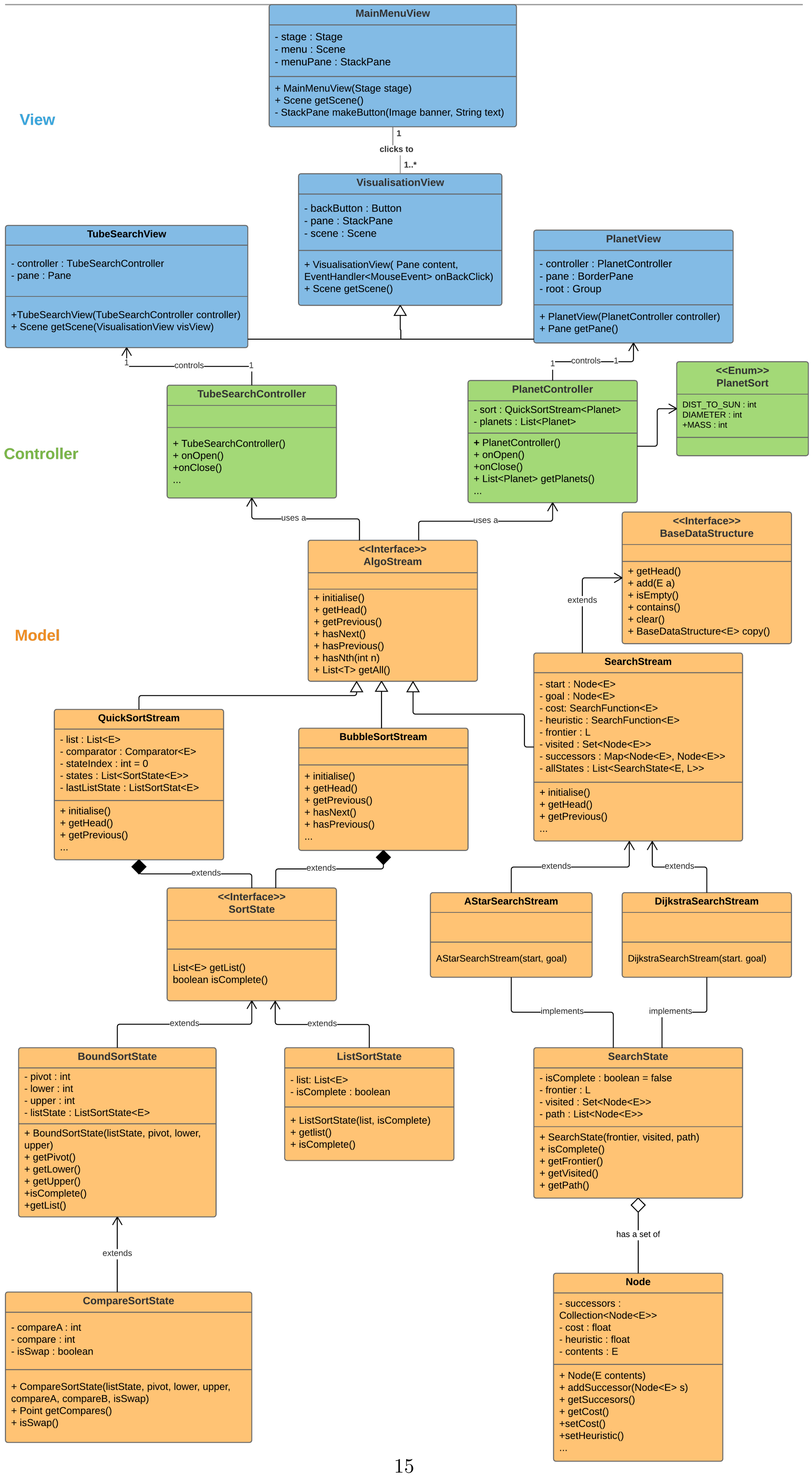


Figure 11: The view and controller of the london underground map visualisation for the current program classes


View
MainMenuView
- stage : Stage
- menu : Scene
- menuPane : StackPane
+ MainMenuView(Stage stage)
+ Scene getScene()
- StackPane makeButton(Image banner, String text)
1
clicks to
1..*
VisualisationView
- backButton : Button
- pane : StackPane
- scene : Scene
+ VisualisationView(Pane content, EventHandler<MouseEvent> onBackClick)
+ Scene getScene()
TubeSearchView
- controller : TubeSearchController
- pane : Pane
+TubeSearchView(TubeSearchController controller)
+ Scene getScene(VisualisationView visView)
PlanetView
- controller : PlanetController
- pane : BorderPane
- root : Group
+ PlanetView(PlanetController controller)
+ Pane getPane()
1
controls
1
1
controls
1
Controller
TubeSearchController
+ TubeSearchController()
+ onOpen()
+onClose()
...
PlanetController
- sort : QuickSortStream<Planet>
- planets : List<Planet>
+ PlanetController()
+ onOpen()
+onClose()
+ List<Planet> getPlanets()
...
<<Enum>>
PlanetSort
DIST_TO_SUN : int
DIAMETER : int
+MASS : int
uses a
uses a
Model
<<Interface>>
AlgoStream
+ initialise()
+ getHead()
+ getPrevious()
+ hasNext()
+ hasPrevious()
+ hasNth(int n)
+ List<T> getAll()
<<Interface>>
BaseDataStructure
+ getHead()
+ add(E a)
+ isEmpty()
+ contains()
+ clear()
+ BaseDataStructure<E> copy()
extends
SearchStream
- start : Node<E>
- goal : Node<E>
- cost: SearchFunction<E>
- heuristic : SearchFunction<E>
- frontier : L
- visited : Set<Node<E>>
- successors : Map<Node<E>, Node<E>>
- allStates : List<SearchState<E, L>>
+ initialise()
+ getHead()
+ getPrevious()
...
QuickSortStream
- list : List<E>
- comparator : Comparator<E>
- stateIndex : int = 0
- states : List<SortState<E>>
- lastListState : ListSortStat<E>
+ initialise()
+ getHead()
+ getPrevious()
...
BubbleSortStream
+ initialise()
+ getHead()
+ getPrevious()
+ hasNext()
+ hasPrevious()
...
extends
extends
extends
extends
<<Interface>>
SortState
List<E> getList()
boolean isComplete()
AStarSearchStream
AStarSearchStream(start, goal)
DijkstraSearchStream
DijkstraSearchStream(start. goal)
implements
implements
extends
extends
BoundSortState
- pivot : int
- lower : int
- upper : int
- listState : ListSortState<E>
+ BoundSortState(listState, pivot, lower, upper)
+ getPivot()
+ getLower()
+ getUpper()
+isComplete()
+getList()
ListSortState
- list: List<E>
- isComplete : boolean
+ ListSortState(list, isComplete)
+ getlist()
+ isComplete()
SearchState
- isComplete : boolean = false
- frontier : L
- visited : Set<Node<E>>
- path : List<Node<E>>
+ SearchState(frontier, visited, path)
+ isComplete()
+ getFrontier()
+ getVisited()
+ getPath()
extends
has a set of
CompareSortState
- compareA : int
- compare : int
- isSwap : boolean
+ CompareSortState(listState, pivot, lower, upper, compareA, compareB, isSwap)
+ Point getCompares()
+ isSwap()
Node
- successors : Collection<Node<E>>
- cost : float
- heuristic : float
- contents : E
+ Node(E contents)
+ addSuccessor(Node<E> s)
+ getSuccesors()
+ getCost()
+setCost()
+setHeuristic()
...

4.2 Testing

We created our tests whilst developing the software in stages. The main tests we did on our program were:

- Unit Tests
- End User Testing

We wrote unit tests for many of our classes and wrote tests for our algorithms such as bubble sort to ensure that these classes were still working correctly as the code was being updated and changes were made.

Our main focus was on user testing, this was where we asked our target users to try our program and provide us with feedback on the design and functionality of the program. We were able to discover bugs, functionality difficulties and design issues from this feedback that the earlier tests did not seem to detect. We were then able to use this feedback to produce a better, user-friendly program.

We managed to get two different prototypes tested by users at different stages of the project. Our 2nd Year peers tested the earlier iteration. Our tutor organised a feedback session with other groups to try each others programs. We created a Google form asking about what they thought about the program and how we could improve it. During this session we gave it out to our peers and this feedback helped with future prototypes, making it a better experience for users.

A few weeks later we had organised an end user testing session at a Secondary School, where the program was tested by year 11 and 12s studying computer science. Overall feedback was mostly positive. One useful improvement we got was to provide a beginner, intermediate and hard level for the graph search so that users that have never used a searching algorithm can work their way up, as the whole London underground map can look quite daunting.

By using the feedback from these tests we were able to adapt our requirements and make the appropriate changes. The scrum approach allowed us to get a working prototype at each iteration, before deciding the next features to implement. So once the first sprint was completed we could create a new sprint based on the next highest priority user stories.

4.3 Risk Management

In our specification, which can be found attached to this document, we outlined several risks which may arise during the project and hinder development. While we were obviously not able to predict all issues which may arise during development we felt that it was useful for us to pick out some likely risks at the start of the project so we knew what to look out for. During development we did encounter several risks to development which we recorded and dealt with appropriately and made our best attempts to minimise any impact they may have. Some of the risks we encountered and reduced include:

- Asvini unfortunately fell ill during development for a week. This was an issue as she was working on JSON data which we needed in order to render the underground map. We were able to minimise the impact of Asvini's ill fortune by having Luke produce enough of the JSON data so that we could test that the code to render the map worked. Asvini was able to finish the JSON data on her return and this affair ended up having minimal impact on overall development as we acted quickly and decisively to minimise risk.
- Another example of a risk we faced during the project was that, during our school visit, the program did not run on the school computers. To avoid this issue we sent an early version of the program ahead to the teacher so that he could check that it did in fact run on the machines and so it would be unlikely for any technical issues to arise during our visit to the school, which they did not.
- Initially some of the members of our team were not aware how to use SVN properly. This resulted in several issues towards the beginning of the project with merge commits. Once we realised this was an issue we met as a team and those of us who were more capable using SVN explained to the others in the group how to use it safely to prevent similar issues arising again and thus reducing the risk.
- During the project Otonye's laptop screen began to flicker and turn off with increasing frequency. This threatened Otonye's ability to continue his contributions to the project should it get worse. Otonye was able to get the screen on his laptop replaced and so this issue never became more than an annoyance as we tackled it early as soon as we identified the risk.

5 Evaluation

Overall we believe the application we produced turned out very well, and we are very pleased with what we have made. In the most part we stayed true to our original ideas and class structure. It was important to us when we designed our class structure that we kept the program extremely modular to allow us to make production of new visualisations in .vism files as easy as possible. It was therefore very important that we stuck to our MVC design pattern. Something which we achieved very well.

MVC structure also allowed us to work simultaneously on separate elements of the program at the same time to fully utilise all of the members of the group so that no one was left without work.

Overall we are very pleased with our HCI. We believe that our program is very accessible for our target audience and features like the graphics and the intractability such as zooming into planets adds an element of fun and engages its users. We were also pleased with the feedback we received from our user testing with ideas for features such as difficulty levels for the tube search which we went on to implement. We also used user stories to develop our features with a client oriented mind set. We think that these two approaches helped us refine the product so that it fit our target audience as well as it possibly could.

There were, however several things which we could have improved upon throughout our project. We would likely have chosen just one visualisation to do and perfect it rather than splitting our resources between two visualisations. While this demonstrated our modular design better it meant we did not have as much time as we would have liked on each visualisation and meant we had much more work for ourselves. We also would likely have taken a different approach when rendering the London underground map as we hugely underestimated the scale of the task when creating a json file to represent the graph. This took a lot of time.

6 Team Work

Figure 13: Picture of our group working together – Parth, Luke, Asvini, Raj, Otonye and Joe (from left to right)

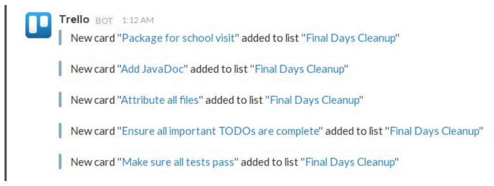
Throughout the time of the project the team split work up in many different ways and people have moved between different subgroups to complete and aid in different tasks. The versatility that each member of the team had was one of the aspects that allowed our group to progress through this project so well.

6.1 Communication

Our main source of communication and what allowed our group to excel in organisation is the fact that we integrated all of the services we used with a messaging application called Slack. This enabled all of us to be able to instant message each other directly to everyones computers, phones and in some cases, watches. This constant communication meant that no one in the group felt out of the loop and if work needed to be done, we could all work together. Slack allowed the integration of different services to enable each member of the group to see if and when anyone else was doing work, our integrations were:

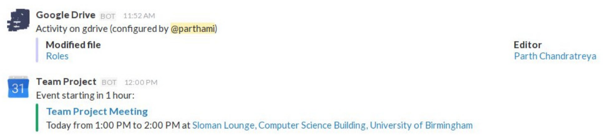
- Trello
  - As we were using the scrum methodology, Trello allowed us as a group to share user stories and keep on top of them. Trello gave us all access to the teams user stories and allowed us to sort them into three areas; completed, in progress and not started. The reason why we linked this to Slack was so that everyone would know what was happening with all of the user stories, and so that we knew if a new one was created so we could start work on it.
- Google Calender
  - We used Google calendar to plan our weekly meetings. We set four weekly meetings, three of which were just the group and the forth was our weekly tutor meeting. The reason that we integrated this in with Slack was because it meant an hour before any of our meetings an alert was sent to everyones phone specifying the times, location and the type of meeting.
  - In addition, we were able to make non-regular meetings in this calendar and it will notify everyone so that people required to turn up would know where to meet and at what time.
  - Similarly to the integration with Slack because the calendar was shared with everyones person calendars everyone would be able to see when meetings are by looking at their own personal Google calendar.
- Google Documents
  - Google documents was useful for making all of the documentation, not only did it give the ability for all of us to share our work, and collaborate the work. It allowed the updates to be pushed through to Slack meaning that people would know when certain parts of the documentation are completed, which were required to start the next bits. This Google Documents integration was hugely beneficial for efficiency with documentation as well as providing security for all of our documentation being backed-up to everyones computers as well as the Google servers.
- Git
  - As SVN doesnt have integration with Slack, so we have used a GIT bot that monitors all of the SVN traffic (pushes and pulls) and notifies within the Slack group. We did this because we thought it was important that everyone could view in real time what work was being done and who was doing it. Every commit would notify everyones phones with whom did it and exactly what they wrote as the description for that SVN commit.

This made our organisation a great deal easier because everyone was kept together within one application which was always accessible.



Trello BOT 1:12 AM
New card "Package for school visit" added to list "Final Days Cleanup"
New card "Add JavaDoc" added to list "Final Days Cleanup"
New card "Attribute all files" added to list "Final Days Cleanup"
New card "Ensure all important TODOs are complete" added to list "Final Days Cleanup"
New card "Make sure all tests pass" added to list "Final Days Cleanup"

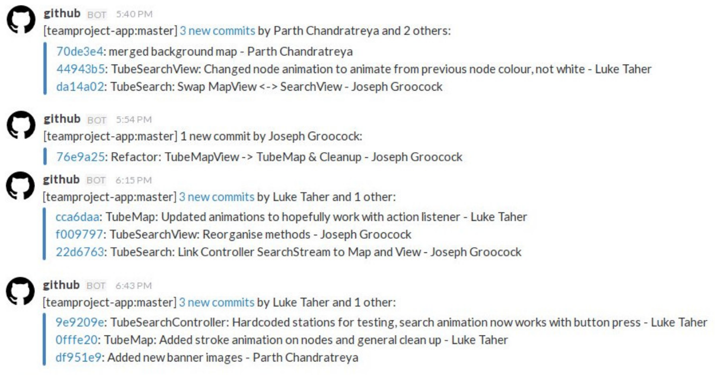
(a) Trello Integrations



Google Drive BOT 11:52 AM
Activity on gdrive (configured by @parthami)
Modified file: Roles
Editor: Parth Chandratreya

Team Project BOT 12:00 PM
Event starting in 1 hour:
Team Project Meeting
Today from 1:00 PM to 2:00 PM at Sloman Lounge, Computer Science Building, University of Birmingham

(b) Google Documents and Google Calender Intergration



```
github BOT 5:40 PM
[teamproject-app:master] 3 new commits by Parth Chandratreya and 2 others:
  70de3e4: merged background map - Parth Chandratreya
  44943b5: TubeSearchView: Changed node animation to animate from previous node colour, not white - Luke Taher
  da14a02: TubeSearch: Swap MapView <-> SearchView - Joseph Groocock

github BOT 5:54 PM
[teamproject-app:master] 1 new commit by Joseph Groocock:
  76e9a25: Refactor: TubeMapView -> TubeMap & Cleanup - Joseph Groocock

github BOT 6:15 PM
[teamproject-app:master] 3 new commits by Luke Taher and 1 other:
  cca6daa: TubeMap: Updated animations to hopefully work with action listener - Luke Taher
  f009797: TubeSearchView: Reorganise methods - Joseph Groocock
  22d6763: TubeSearch: Link Controller SearchStream to Map and View - Joseph Groocock

github BOT 6:43 PM
[teamproject-app:master] 3 new commits by Luke Taher and 1 other:
  9e9209e: TubeSearchController: Hardcoded stations for testing, search animation now works with button press - Luke Taher
  0fffe20: TubeMap: Added stroke animation on nodes and general clean up - Luke Taher
  df951e9: Added new banner images - Parth Chandratreya
```

(c) Git-Svn Integration

Figure 14: Diffrent intergrations in Slack

6.2 Organisation

Predominately the organisation of the programming was decided by a single member (Joe), this was agreed upon and wanted due to his extensive experience in Java and large application development. Having worked in Group projects previously like Hackathons, we knew that his leadership would be vital to our progression. From this the programming structure was decided allowing different people to work in groups and/or individually to complete the programming aspects of the project. This being said, all members of the groups worked together and worked hard to ensure that we completed the project and reached the communitive goals that we set out at the beginning. We fortunately had a very committed and motivated team, all of which were able to bring something to the table and had their own strengths.

Joe	Organised the layout of the programming and coding. Overlooked all the programming done
Parth	Worked closely on the GUI, specifically with the Planets and Tutorials, in addition to teaching everyone in the group JavaFX so that they could work on the GUI too
Raj Asvini	Helped with the Tutorial, worked hard on the JSON files and the documentation Worked hard on the Tube-station JSON file, the software engineering and documentation
Luke	Worked with the algorithms and closely on the tube station GUI
Otonye	Worked on the algorithms as well as the testing

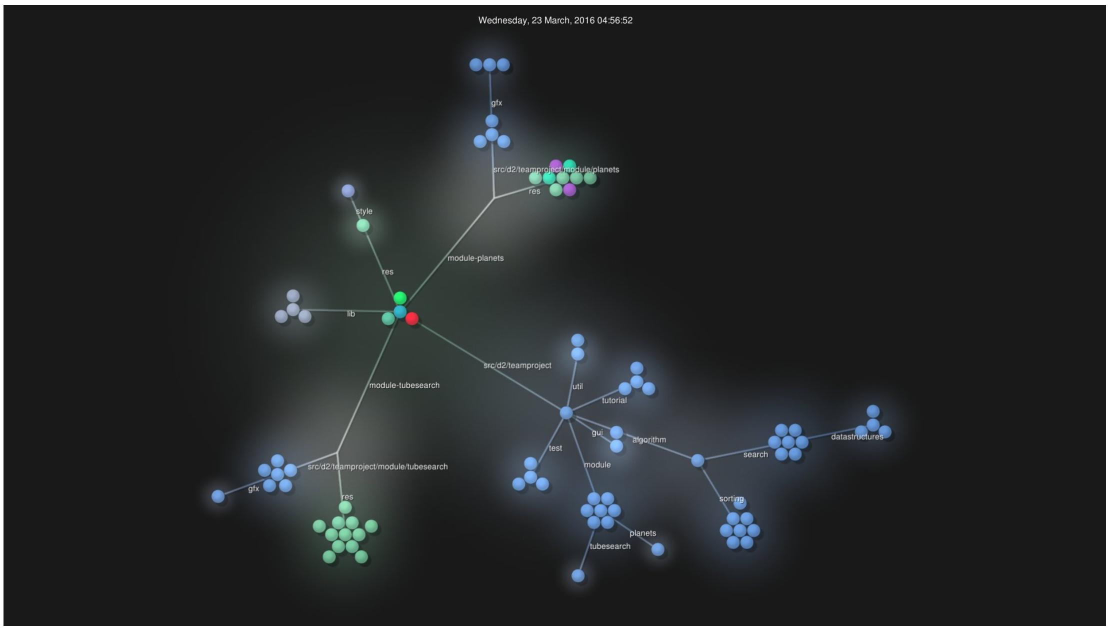


Figure 15: Our SVN tree

7 Summary

Working on PARTH has been something which we have all enjoyed and a great experience from which we have learnt much. We are proud of the product we have produced and how far it has come through development. While we faced problems during development we were able to overcome them and learn from them as a team. The experience was a very rewarding one for all of us, particularly in gaining vastly positive user feedback from students which really made us feel like we had made something worthwhile and useful. We would like to see our program expanded in the future through .vism files by teachers and students alike and for it to grow into a powerful learning tool which is used in schools.

8 Personal Reports

Raj Bariah

This module has given me an incredible insight into working in a group and production of a larger scale application from start to finish. I found the work very enjoyable and felt my role in the group, although not leading, was necessary and required. Though I didnt take a leading role, I helped a lot in the planning and was very involved in the early stages at the group meetings, which were mainly about the application ideas and the planned organisation of the group. Over the course of the project I took less of a programming role, not being a part of the algorithms at all, but taking a keen interest into the GUI. Learning JavaFX from another team member was a highlight of this for myself as it was interesting and showed exactly how much of a team we were able to work as. Parth used our slack group to arrange a meeting between all team members that hadnt used JavaFX yet and taught us it. This was insightful and allowed us not only to understand how the current GUI features worked at the time, but allowed us to get involved in this aspect at a later time. This is what I ended up doing as I got heavily involved with the Tutorial aspect of the program. This was the section that displayed the text explaining what the algorithm was doing at a certain time. Me and Parth worked very closely together over this section, we both worked on the Java aspect, where I learnt a lot from Parth as he has more experience on this section. Then I worked on the JSON files ensuring that the tutorial text was in the correct format, it was split into the correct steps to display and that it was coherent. In addition to these JSON files I worked closely with Ash to complete the JSON files for the tube stations, where I helped enter in the lines and station information.

Similarly to the application side of the module, we ensured that every member was involved in the documentation side. Here I took more of a lead role taking it upon myself to lead the Prototype presentation in addition to starting the final presentation and final report documents. With this side of the project I feel everyone worked very hard together and everyone was always willing to read other peoples work to give immediate feedback.

Over the entirety of this project I have been extremely happy with my group and with the progress we have made. Not only have we managed to achieve our goals for the application we have developed but we have also managed to enjoy working together and become friends as well as just peers/colleagues. Fortunately I have benefitted from this experience not only academically but also socially as within the group we started little traditions (e.g. ordering dominos pizza every Friday whilst we worked together as a group). However in hindsight if we were to redo this project and start from the beginning I think that ensuring all members had a larger role in the programming aspects of this module would be a factor Id change. This is because the code was created in an extremely high level and abstract way that most of the group failed to fully understand. This demoralised the group as certain points and made it almost impossible for some people to alter and amend different parts of the code. The way I would have changed this would have been to split the program into three main areas and then encouraged pair programming across all three areas. This way everyone would have had direct help with and from their partner, as well as everyone getting more hands on with the code.

Otonye Besman

For the team project, I had a hand in the implementation of the algorithms used to run the sorting of the planets. To start, I wrote a basic bubblesort algorithm used to sort Integers. I used the program I wrote to sort an ArrayList of integers; this was used to test my program as I was working. Afterwards I made the bubblesort class a stream class, so that it can be connected to the GUI and sort the planets. I also change the type from integers to an Immutable type. The BubblesortStream class I created takes a list of immutable types E and a comparator object which will be used to compare the items with their neighbouring items. This BubbleSortStream implements Sort stream. Sort stream is a classed used to return the next move and get the previous move when the sorting algorithm runs.

I also worked on the Quicksort algorithm, like the bubble sort algorithm, I ran my program on an ArrayList of integers first, to make sure that a sorted ArrayList of integers is returned. After confirming that the program works, I made the QuickSortStream class; the class also extend the sort stream class and also returns the next and previous moves for the program. Like the BubbleSortStream class, this class took a list of immutable types and also a comparator used to compare chunks of the list. I wrote the Json file for the Astar search tutorial mode. This json file was used in the tutorial mode for the London Underground tube search. I was also involved with writing the JUnit test for the sorting algorithm. I started writing the JUnit test for the QuickSortStream and passed. Also I wrote the JUnit test for the BubbleSortStream and it passed as well.

I enjoyed working on this project as a team. I had some challenges when it came to actually implementing the algorithm for the sorting, because I had to learn how to stream my bubble sort, so that the Graphical User interface could use the information passed to it by the algorithm. As the time went by I was able to understand how to stream the program and was able to complete the bubbleSortStream and the QuickSortStream.

If I had the opportunity to do redo the project, I would have approached it differently. I would have taken more time in writing pseudo code and planning before the actual implementation of the code. I would have been writing test classes once I am done with every algorithm stream. All in all I manage to get the work done, so I am happy with the work.

Parth Chandratreya

My main focus for this Team Project was on the GUI-side of our visualisation. I found my role a very fulling and enjoyable role, even though I had to learn the JavaFX library (a modern replacement for Swing) for this project. I was also introduced to SVN for the first time, so my programming knowledge has increased throughout this module. Working extensively with a team larger than usual has helped me with my ability to work with others and bettered my knowledge of software engineering. I have also learned a great deal from my team members in coding styles and problem solving.

Joe and I worked closely initially as we both were new to the JavaFX library and wanted to create an interesting and interactive interface for the program. This aspect of the program took a major chunk of my time over this term. I also held a meeting to teach the rest of my team how to use this library. We had a lot of work to do to meet the prototype deadline, which I think came out great. My favourite part of the project was the 3D implementation of our local solar system, along with textures and rotations. Alongside this, I also helped implement the animations.

Following the presentation, I transitioned to working on the Tutorial Mode for the project with Raj. We implemented a system that would work across the visualisation and help the users understand the algorithms. Later on , we also change the actual text of each step to reflect in real time the alogorithm. This was another exciting aspect of the visualisation and we worked well together to implement this feature. As the final deadline approached, I switched roles again as we all started cleaning up our code and writing documentation, adding our final touches.

Overall, I am personally happy with the way this project turned out and the progress we made. I have made some great friends and I am thankful for the advice and knowledge that has been imparted on me. I feel that every aspects of my progression as Computer Scientist has improved, from software development to peer interactions. I am even surprised to say I enjoyed our very frequent meetings as they helped us stay on top of the work for a majority of the term.

In retrospect, there are two main points that I feel that could have gone better. Firstly, I wish I personally had been more efficient in my time. I could have wasted less hours at times had I have been more efficient as time management can be an issue for me. I do think have improved on this issue this over the weeks.

Secondly, I also wish we had been slightly less ambitious and perhaps more realistic with our goals. There were some weeks where I was not fully confident we could complete this project in time. We could have also focused more on the software engineering aspect. This whole experience has reinforced the fact that all facets of this project, not just writing raw code, must be held to a high standard.

Pair programming might have helped all of us understand the codebase better and involved everyone. Nevertheless, I thoroughly enjoyed this team project and I am looking forward to the next opportunity I have to work within a team.

Joe Groocock

Team Project for me has been exactly what I expected it not to be: hard work with many team complications in between. I came into this module expecting to have a good time, write some code, make some friends and produce something cool. While I did make some new friends andI wrote *a lot* of code, it definitely wasn't enjoyable throughout and moveover the 'teamwork' aspect was much more involved than I had initially planned. There were many hurdles like disagreements and miscommunication that slowed progress initially. In my view this module, whilst not being as expected, has exceeded my expectations in how teams form, grow and work together through their differences- it's been a valuable experience for me.

My contributions to the project consisted of mostly overlooking the code, maintaining a sound and solid codebase for everyone to work with. I've always been available to stash my changes and help the other team members resolve their issue, merge their code and explain the workings of the existing code. My focus has moved from area to area within the project, following the rest of the team, assisting with general code and cleaning up, standardising as we went along. I've had a big hand in most of the code written for this project, helping the team to understand what is going on and generally managing which tasks are being performed and by whom.

While it may not have been a popular initiative amongst all the team members, I enforced a good policy on commits to keep the repository clean and have standardised, relevant and concise commit messages. As a result the repository is very well kept, almost all commits compile to a working standard and not a single file that shouldn't be in the repository was commited (`.classpath`, `.idea/`, `bin/jar/.class` or other meta file)- the same cannot be said for many team project groups.

Right back at the start of the project I jumped straight in to building up the `ModuleLoader` system to dynamically load code into the application. This is something I've never attempted before but wanted to try simply because of the sheer coolness-factor. I can happily say that after a lot of tweaking and error handling I managed to construct a solid system for dynamically loading code from external `.vism` files. This feat in itself taught me a lot about how the underlying ClassLoader system works in Java helping to further expand my knowledge of this horribly flawed language.

A decision that we made as a group right from the start was to use JavaFX as our front-end toolkit over the admittedly aging Java Swing as a new perspective on what the language can provide and as a bit of a challenge. While I still completely stand by this decision, it was brilliant, it didn't come without a price. Many hours were spent, not just by myself but by everyone, reading, testing, re-reading and changing out the design of the program as we learned more about how JavaFX itself works and how we should use it to pull our program together. As a library and a tool, JavaFX is brilliant, especially the 3D animation system. The animation subsystem that we took complete advantage off really added a *wow* factor to both the Planets and Tube Search modules in exchange for very little code and effort. It provided a quick and intuitive platform to build exactly what we required for this project.

Given the choice to do this project again, I would not have picked to make an algorithm visualisation as it doesn't allow me to demonstrate my skillset fully and also it's not an exciting concept; it has constantly reminded me of writing boring data visualisations for big companies which honestly nobody wants to do. Not given the choice, I'd have chosen JavaScript and HTML as my goto framework for building a visualisation as it has much more flexibility and is a much faster platform to develop for.

My ideal project instead of the assigned task would be to create a fully-featured service including network communications, file handling and data management as I feel it would have allowed me to present my skillset much more broadly.

One achievement I am particularly proud of is during this project I managed to increase my GitHub stream to 24 days, many times what it was before, all thanks to commiting to this project.

Asvini Patel

This project has been a thoroughly rewarding and valuable experience. From the first meeting with my team I felt we were all on the same page about what our expectations were for this project, and we were all motivated to produce a great program. I was wary at the beginning of the project, as my programming skills were not as strong as the rest of the group. So I felt my skills would be more suited to the software engineering and report work. I was very keen to put in as much work as possible in all other ways for the project to make up for my lack of java knowledge, I made the group aware of this during the first meeting and they were very understanding. In our first meeting we started collaborating our ideas for our program; by the third meeting we had decided on using my idea of creating real world examples as visualisations such as sorting planets and searching the London underground map. Everyone was very happy with this choice.

I focused on the software engineering side of the project for the first few weeks. I was appointed scrum master, where I was in charge of organising scrum meetings and ensuring that team members stuck to the deadlines of their sprints. I created majority of the user stories and created a product backlog on Trello. By being scrum master it meant I had to have a good understanding of the whole project, and so by drawing out an initial class diagram I was able to get a more in depth analysis of how the program was going to work. I then focused much of my time on the London underground map visualisation. I am mostly proud of the many hours I put into collecting raw data and manually inputting this into a JSON file for the London underground map visualisation. I felt the magnitude of the file was a great achievement for me, and it is something I will always remember as part of my contribution to the team. One of the main problems I overcame was when we realised the detail at which the JSON file had to be in, such as finding the coordinates to not only the stations but to each individual curve on the underground map. This was a very time consuming piece of work that was one of my main responsibilities. For the evaluation of the project I assisted Luke by planning the presentation and questionnaires that he was going to take to the school for our user testing.

Having never worked on a project of this scale before I had many learning opportunities, which has really developed my interpersonal skills as well as my computing skills.

This project has taught me how to manage my time effectively to ensure I kept up to date with all the work for my other modules as well as organising and keeping track of everyones timeframes for the project. I learnt how important communication is when it comes to large projects such as this one. I was also grateful that my teammates were happy to explain their code to the rest of the group, for example Parth was very happy to explain to everyone how JavaFX works. The technical skills I learnt included how SVN works and basic JavaFX.

I enjoyed how well the team worked for the majority of the project; we all worked consistently and efficiently. However, towards the end of the project the stress started to get to us and we realised that differences in coding practices caused some problems later on when it came to teammates working on each others code. In the end I feel our motivation had decreased, as the level of commitment was not equal to the start of the project.

If I were to go back and do this project again, I would ensure everyones programming differences were tackled straightaway to prevent conflicts in the future; pair programming may have helped this situation. I would also try to make sure our group had realistic and achievable goals, as I think in this project we stretched ourselves rather thinly creating two visualisations when we should have focused our attention on creating one clear, detailed visualisation of an algorithm.

I can say that overall it was an enjoyable experience and I am very proud to have contributed towards creating a successful program. I have made great friends with my teammates and I am glad to have worked with them.

Luke Taher

Overall I feel our project was successful in our aims and I am proud of the product which we have produced. During the project I have learnt new skills and I have embraced the problems and stress we faced as a team throughout the development. We came up with a strong idea and stuck by it through the weeks, adding new features and improving while staying true to our original plans.

During the project I worked on various aspects of design and development. Initially my task was, along with Ash and Joe, to produce a Class diagram and overall MVC model for our software. During this period, I came up with the concept of Algorithm Streams which became an integral part of our model and allowed us to integrate the model and the view very easily. After we had completed work on the program structure we split into two teams and began development. I was part of the team working on the model. In this role I developed the tutorial mode model which was responsible for processing and storing the written tutorials for the program.

Once this was complete the bulk of my work effort went towards the London Underground graph traversal visualisation. I wrote code to parse the JSON data file of stations as well as to store the stations and lines in objects as part of the model. I then went on to write the code to visualise the underground map in JavaFX using the model. I wrote the code to drop the lines and the nodes on the graph which was later evaluated by Joe. I then worked on the animation of the algorithms on the graph and I also added buttons and code to change the difficult of the traversal by reducing the number of nodes in the graph.

Another role I had towards the end of development was to fix small bugs and to finish any small remaining development tasks. Examples of this include re-implementing the bubble sort model as it had originally been implemented incorrectly, writing edge case code to prevent the same node to be selected as both the start and end node of the graph traversal and to aid Asvini in developing a technique to position Bzier curves in a way which allowed us to have parallel lines on the underground map.

I was also responsible for User Testing. In this role I went to a High School and carried out a questionnaire and a hands on demonstration of our program with a GCSE and an A level Computer Science class and collated the results.

During the project I learnt and refined a wide range of skills in software engineering, software development and soft skills. I improved my communication and presentation skills through discussing ideas and problems with my team mates and presenting the program in the presentations as well as during the user testing. My knowledge and understanding of the Java programming language was improved through this module as I learnt much, specifically to do with JavaFX through research and from my team mates as this was a tool I had never used before. I also gained valuable experience working as part of a team on a software project, something which will likely be invaluable to me in my career. Asvini also taught me a lot about software engineering as she has taken the software engineering modules this year. I was able to learn and refine a lot of software engineering techniques and build my knowledge in this area through working on the initial design with her.

In hindsight there are a few decisions I think I would have made differently throughout the project. Firstly, I would have suggested we spent more time on development in the initial weeks as I believe we spent too long working on how we would go about the specifics of our project which left us having to spend a lot of hours on development in the later weeks. I would also suggest that before we started programming we set down some design pattern rules for our code. This is something I feel strongly about as we had some team members who had more experience in Java than others and wrote very abstract and sometimes quite unnecessarily complex code which was uncommented until the end of the project. This made it difficult for those in the team who were weaker at Java to contribute evenly to the code base and even for the abler of us it took a long time for us to be able to process and understand the code which slowed down development heavily.

In summary I believe the project to be a success, while we faced some problems during the project they were all things we learnt from. This project has given me invaluable team work experience which will stay with me throughout my career. Im proud of the product we produced and of my team for the hard work they put in.

Team Project D2
Product Specification and Outline

Raj Bariah Otonye Bestman Parth Chandratreya
Joe Groocock Asvini Patel Luke Taher

March 24, 2016

Contents

1 Introduction

1.1 Purpose

The idea of the application and program is to teach people of all ages, in different educational stages in life the concept of a multitude of computational algorithms (e.g. sorting algorithms like bubble sort and search algorithms like breadth first.). The application will be developed so different preloaded algorithms will be listed and be available to view and learn from.

We feel like that applications currently available to teach these things can be very verbose in the explanations of different algorithms and do not help people understand the underlying concepts with simple language. Hence this will allow people to understand this and further their knowledge.

1.2 Document Conventions

Within this document we are ensuring the consistency within the type by using the default formatting rules within LaTeX. The sizing of text will purely depend on context and what it represents, as for plain text within the document, it will be kept to a constant standard, whereas headings will have a consistently larger font size, effectively using heading texts and using comments for the plain text.

In conjunction with our conventions we will only specify tasks that we wish to complete at the time of writing this document. However any idea, additional concepts or functionality that we may want to achieve in the future will be marked with a lower priority. These have been included because they are parts of the system which would improve usability, however they are not vital to what we want to achieve in this project.

1.3 Intended Audience and Reading Suggestions

This document will be created for several reasons, for everyone involved with the project. As developers, we require this documentation to fully comprehend the time frame in which the project needs to be completed. In addition to this, this document will also provide a critical source of ideas and plans to refer back to, for the developers to see the specific requirements of each different task.

During the testing phase in the production of the project, this document will be referred back to as a template in order to evaluate the development progress and if the implementation has been completed to the standard required. This document will provide information on what the final product should provide for functionality and therefore testing will be done against the requirements defined within this document. This ties in with whomever will evaluate the project as it provides the strict guidelines this project has, as well as providing goals that the team should reach with the finished project – meaning the product can be reasonably compared directly against these requirements.

1.4 Product Scope

The product being created will be primarily a desktop application to teach a multitude of people across different ages about many computer algorithms for search, sorting and so on. This application will be built using Java as per the project requirements and consequently will run on many platforms, those that provide a Java implementation, including (but not limited to) Windows, Mac OSX and Linux with the potential for providing mobile and web interfaces for the platform in the future – both of which are considered a low priority for this project.

The objectives we have for this project are to show users how different algorithms work and to provide interactive sections of the application for the user to be able to demonstrate their understanding from what they have learnt.

For example, a user may have to perform sort using data provided by the application *or* see a search algorithm running on real life data and examples. Another goal is to ensure that we only use language appropriate for our users, so we will ensure that different explanations of each complexity level for all age groups is available. *Age information will be requested as needed by the program upon starting.*

1.5 References

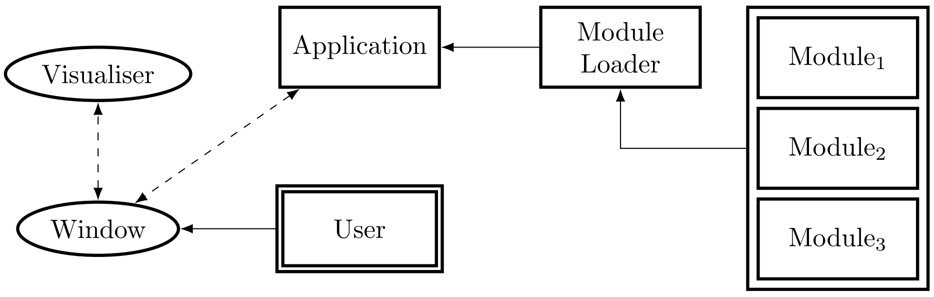
No references are required for this project.

2 Overall Description

2.1 Product Perspective

Our software is designed to be a self-contained software package that runs as a standalone suite in a desktop application, web applet or mobile app. The software itself is a modular system in which various teaching modules can be inserted and interacted with within the application by means that the program provides.

Figure 1: An abstract model of the system layout.



There will be a standard interface to load and execute the individual visualisation modules as modeled far right (Figure 1). Each module is loaded on application start or refresh and displayed to the user as content to be displayed. Interaction for each module is handled by the main programs and passed to the code of the visualiser module to be executed; all of the logic for each visualiser is handled within the module itself then pushed to the program via a standard interface to update the appropriate contexts.

2.2 Product Functions

The following core fuctions of the application are all necessary for full operation of the software system. A more detailed overview can be found in sections 4 and 5

1. A selection menu for choosing between the loaded visualisations in the application (Section 4.2)
2. Graph traversal algorithm visualisation with various different algorithms (Section 4.3)
3. Sorting algorithms visualised with several algorithms of varied complexities (Section 4.4)
4. The system should allow for modular visualisations that can be dynamically inserted into the program and loaded at runtime (Section 4.5)
5. A tutorial mode must be available for demonstrating each algorithm to the user with helpful and descriptive prompts (Section 4.6)

Each visualisation module has potential for considerable user interaction depending on the simulation and the target user. Most of the default demonstrations included will have varying degrees of interaction within the visualisations to showcase the different potentials that the application can provide.

2.3 User Classes and Characteristics

This product is aimed at an educational market with ages ranging from young high-school to adults. It will be adaptable to anyone in that age range, configurable by choice of the user as they load a module to use. The software will adapt how each visualisation is displayed and explained through the various levels of user comprehension and staged complexity stored within each module. A module will specify a suitable user age range and have several levels of increasingly complex descriptions and examples to show to the user based on their age. We expect the majority of the user-base for this application to roughly be ages 14-24, from high school to university. This subset of users covers the majority scale of difficulty levels ranging right from the beginner stages to the complex in-depth algorithm analysis required for university level comprehension. Such a wide span of users should all be satisfied by the grade of information that is conveyed to them in accordance with their knowledge level.

2.4 Operating Environment

As the program is written purely in Java, it should have no conflicts with any other software due to the self-contained nature of Java applications that run in a JVM. All hardware requirements for the software

are again abstracted away by Java therefore the only true potential limiting factor is a lack of a JVM implementation on a given platform with sufficient display and graphics hardware support.

Almost every modern platform that this application may potentially be run on has ample support for the requirements of this application.

Depending on the prerequisites of a given module, the program may require hardware graphics acceleration for rendering however most desktops, laptops, tablets and even mobile phones provide a sufficient hardware accelerated graphical experience plentiful for this software.

2.5 Design and Implementation Constraints

We are required to only use Java to complete this project - this is a large design constraint as there are other languages that will allow us to create a GUI (graphical user interface) with more ease.

However as we knew from the early stages that we had to create this system in Java only we catered for this and ensured that we would not decide to implement anything not available within the confines of the Java language. As a commercial product, this project doesn't supply enough time nor developers between the team to implement, test and maintain a full system with every feature we originally planned to implement. Due to this constraint, many of the initial plans have been condensed or scrapped entirely to allow for a sufficient implementation with the resources we have in order to still deliver a complete and stable product.

Maintenance of the software including updating and fixing as well as creating newer, relevant and performant visualisation modules for the software would be an ongoing job and this limited time frame simply does not allow for such a service therefore only a few example tools will be supplied with the shipped application as a demonstration of what it could potentially provide.

2.6 User Documentation

We will be producing a readme.txt that will be downloaded with the application for the users to be able to understand this program. The reason for no instruction manual is because this will be a very intuitive program that will have help and tips how to user it built into the program. Each extra visualiser module can optionally be supplied with documentation but that is up to the developers of said plugin. Some of the more intricate standard tools will be supplied with basic documentation explaining how they can be operated.

As a primary principle for this software "*to teach users how algorithms work through the medium of interactive visualisations*", it should be somewhat assumed by the user that the software package is self-explanatory to an extent given that it is a core premise for the existance of this tool in the first place."

2.7 Assumptions and Dependencies

We will assume that the user has a computer capable of running simple applications with a screen and appropriate input devices for interaction. Here are some suggested device specs for running the program. They are strictly guidlines as actual performance will vary depending on hardware and operating circumstance:

- 1GB RAM recommended minimum
- Display output with resolution at least 1280×720
- Keyboard & Mouse *or* Touchscreen
- *Optional:* An active internet connection for updating and downloading new visualisation modules for the program.
- 3D Accelerated graphics processor
    - ***N.B*** *There is potential for some visualisations to be considerably more graphically intensive than others and may require more performant hardware to run at full speed.*

3 External Interface Requirements

3.1 User Interfaces

Throughout the software package there will be many standard controls such as help, navigation icons and standard images that should become familliar to the user. Some of these standards are highlighted (Figures 2, 3 and 4)

Figure 2: Main Menu

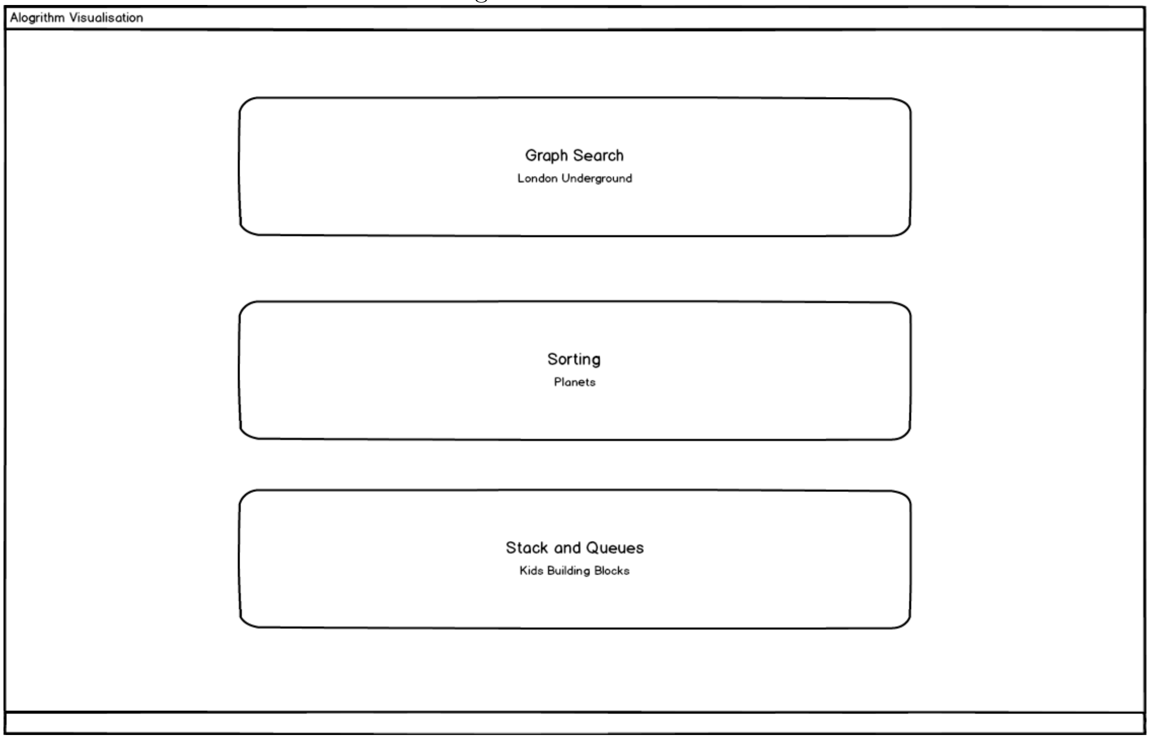


Figure 3: Example visualisation page

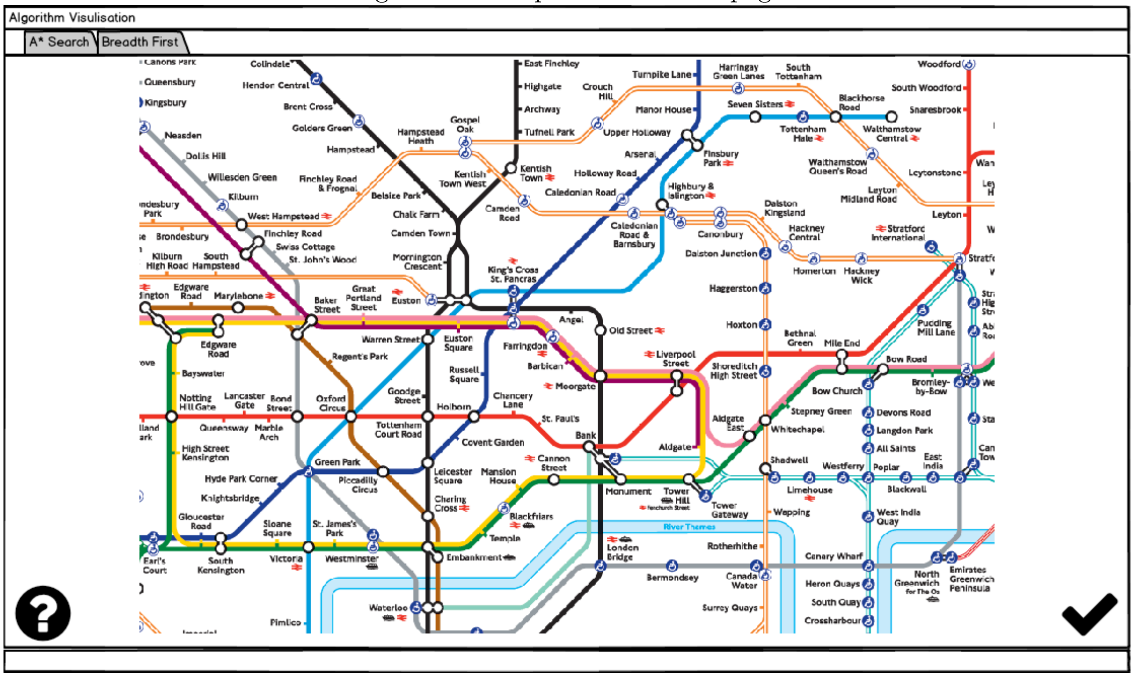
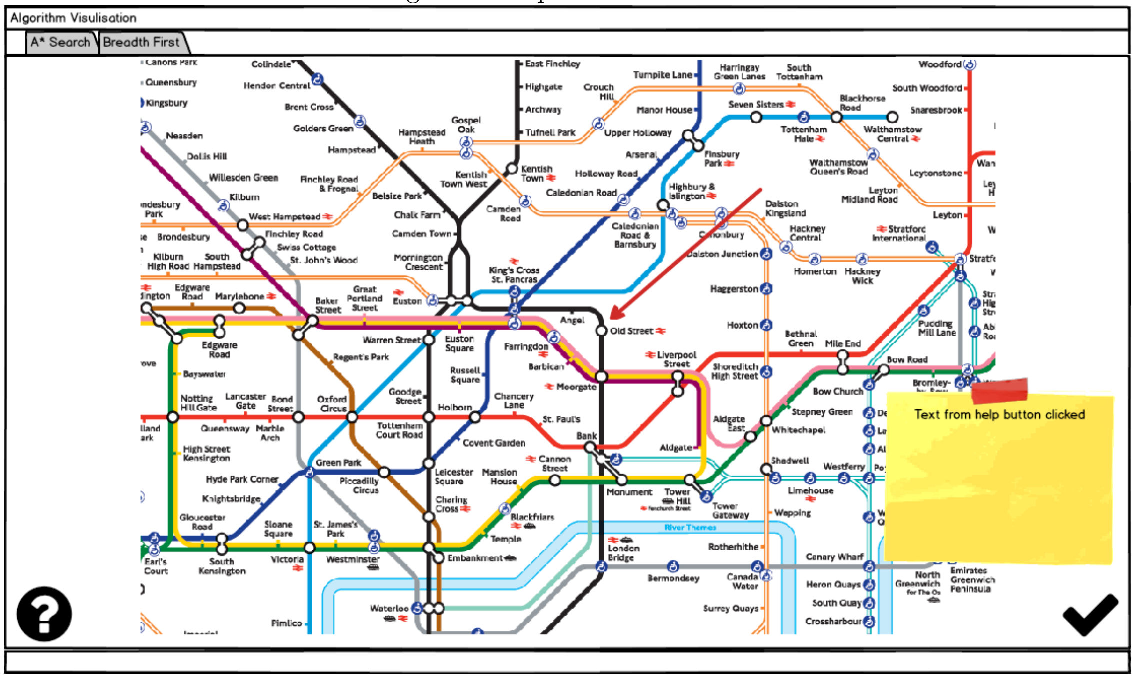


Figure 4: Help button activated



We have two GUI implementations. Both will use a basic mouse and keyboard input system. Using this way of input enables an expansion to touch input alongside a virtual keyboard. A help button on each visualisation will help explain each implementation to the user. Each tab or button will take you to the visualisation selected. Each visualisation could have different inputs such as selection by the user or text input.

3.2 Hardware Interfaces

The hardware used to control the software will in most case be a keyboard and mouse for inputs as this will be an application to be used on desktops. No extra hardware is required to use this application in a kiosk mode. The inputs for this mode would be touch input. As this software is an algorithm visualiser it requires no interaction with physical hardware other than that required for input and output to control and display the software.

3.3 Software Interfaces

This application will have no external communication. The application will run on WIndows, Linux and Mac (amongst other systems, see Section 1.4) with potential for expansion to web applets and mobile applications. As a contained system, there is an interaction between the visualisations run within the software and the portable modules that supply them including assets and executable code. There is no sharing of information in or out of the program.

4 System Features

4.1 Application startup and shutdown

Priority: High
The program will be an window based executable which will be used for all interactions between the user and the software.

Functional Requirements

1. Application startup: The System will be opened locally from an executable.
2. Startup menu: Once the system has started up it must allow the user to select an algorithm.
3. Application Shutdown: The System must allow the user to close it from the menu.

4. Application Abort: The system must allow the user to close it during use but prompt the user for confirmation first.

4.2 Application menu

Priority: High
The program will allow the user to see all available algorithms and select an algorithm. The user must be able to initiate an algorithm session from this page.

Functional Requirements

1. Algorithm description: The system must provide a description of each algorithm.
2. Algorithm selection: The system must allow the user to select an algorithm and navigate to the algorithms page.

4.3 Graph Traversal Algorithms

Priority: High
The program must allow the user to view a visualisation of graph traversal algorithms based on the london underground map.

Functional Requirements

1. Algorithm parameters: The user must be able to select the traversal algorithm by clicking between buttons
2. Select start and destination: The user must be able to click stations to select the the locations to traverse between
3. Select route: The user must be able to click through stations to create a path
4. Verify route: The user must be able to check the route they have selected is correct

4.4 Sorting Algorithms

Priority: High
The program must allow the user to view a visualisation of sorting algorithms based on the planets in the solar system.

Functional Requirements

1. Algorithm parameters: The user must be able to select the sorting algorithm by clicking between buttons
2. Choose representation: The user must be able to select which of the visual representations they wish to see by clicking between buttons
3. Interact with elements: The user must be able to interact with the elements which are being sorted so that they can be ordered.

4.5 Modular Plugin System

Priority: Medium
The program should take each individual visualisation tool as a self-contained package that can be imported and loaded into the main application. Each modular container can be updated and distributed independently of the primary visualiser tool.

Functional Requirements

1. Modules should be loadable at any time into the application for instant use.
2. Any module running in the software must be self-contained and have no external dependancies other than what the base application requires.

4.6 Tutorial Mode

Priority: High
The program must show the user a tutorial on how to carry out the given algorithm.

Functional Requirements

1. Launch tutorial mode: The tutorial should be launchable from a button on the algorithm page.
2. Algorithm description: The tutorial should display in text a description of each step.
3. Tutorial navigation: The tutorial must be navigable using buttons to go through the steps.
4. Update Graphical representation: As the tutorial progresses the graphical display of the algorithm should be updated to match the stage in the algorithm the tutorial is at.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

1. The system must be available 24/7
   Rationale: To allow constant access.
2. The download time of the visualisation may vary depending on Internet connection speeds
   Rationale: Each user will be affected differently
3. The system response time when loading data must be instantaneous for the user
   Rationale: To ensure the user is not waiting long for data to load

5.2 Safety Requirement

There are no safety requirements.

5.3 Security Requirements

No security requirements are necessary. Product is available to users of all ages. Application does not save any personal data about the user such as their name or age so there will be no privacy issues with regards to use of the product.

- Product is available to users of all ages.
- Application does not save any personal data about the user such as their name or age so there will be no privacy issues with regards to use of the product. This allows easy access to the site.

5.4 Software Quality Attributes

5.4.1 Reliability

- The system must be available 99% of the time.
  Rationale: Users need to be able to access the application 24/7.
- The system must respond within < 10 seconds to ensure that the customer does not believe the system has crashed.
  Rationale: Users will only wait a short amount of time before they believe the application has crashed
- The application must be scalable to accommodate the growth of the product.
  Rationale: To allow the system to accommodate the increased number of users.

5.4.2 Adaptability

- Must be easy to implement new visualisations of new algorithms, and improve on the existing algorithms based on feedback.
  Rationale: To ensure that users do not become bored of product.
- The software must allow for loading of new and updated visualisations.
  Rationale: As new algorithms for completing tasks are discovered and created, the program needs to have a way to stay current and relevant to provide a useful service to its users.

5.4.3 Availability

The system will be fully implemented by March 2016.
The program must be able to run at any time and place, depending on nothing including external services to work.

5.4.4 Maintainability

- The system should be coded in a modular way to obey to high coding standards.
  Rationale: To produce maintainable code with the ease of resolving any coding issues and for future modifications.
- The system code must have javadoc clearly explaining how a function works in detail and how to use & implement it.
  Rationale: To ensure code is easily accessible and understandable for future work.

5.4.5 Portability

- Users will be able to use the application on a mobile browser, the website will scale, as required, to the screen size.
  Rationale: Users will be able to access the website as long as they have their phone with them.

5.4.6 Testability

- Be able to test each individual algorithm with unit tests and integration tests.
  Rationale: Each separate algorithm, algorithm part and module need to be tested individually before integration into the main system as it siginficantly reduce the chance of causing issues with the rest of the program.

5.4.7 Usability

The application will display a menu screen (Figure 2) showing links to our 4 primary algorithm visualisations:

- Graph Search (London Underground Map)
- Sorting Algorithm (Planets)
- Other potential loaded modules in the program.

- The system must be easy to use and navigate for all types of users.The system must be consistent and take <4 minutes for a new user to understand navigation.
  Rationale: To ensure navigation is intuitive.
- The system must be easy to use and navigate for all types of users regardless of input and age. The system must be consistent and take about <4 minutes for a new user to understand navigation.
  Rationale: To ensure navigation is intuitive and the user experience is siginficant enough for the user to want to use the program again.

- The system must take <3 minutes for users to understand the layout of the menu screen and integrated help options throughout the entire system.
  Rationale: To ensure homepage is intuitive.

6 Risk Analysis

Risk	*Effects*	*Possible Solution*
Code might get corrupted	This will make our code redundant and not usable	You can backtrack to an earlier version of the application on SVN and work with that
Someone getting ill	This will slow down the production process because someone that was meant to do work is not available	We would have to share the task that was meant to be done by the ill person between ourselves
Forgetting to pull recent code before pushing	This will cause a conflict between the old and the new code	Everyone should take their time and fix the conflict or just backtrack to an earlier version
Disagreement of decisions between team members	This will prolong the time of the production of the application	A vote will be used to determine what happens in cases like this.
Application not working during the time of presentation	This will be a bad presentation	Series of testing will be done before the main presentation to avoid such a scenario from happening
People finding their task difficult to handle	This will slow the team down	In this case, we can allocation a lot of easier roles to the people having this problem, and give harder but less roles to the people that can handle it.

7 Use Cases

7.1 Actors

Students and users who wish to understand the workings of algorithms

7.2 Precondition

There is a diagram of the underground and the stops

Basic Flow of Events

- The use case begins when the user goes on to the site
- Use Case: Validate User is performed
- The system then provides options of algorithms to learn
- The user chooses an algorithm
- A real world example of the algorithm is presented to them
- The system prompts them to run the algorithm manually
- The user runs the algorithm manually
- The system checks if it is correct
- The user is congratulated by the system
- The use case ends successfully

Alternative Flows

Invalid user: Step 2 of the basic flow of the use case: Validate User doesn't complete the successfully

- The use case ends with a failure condition

Wrong algorithm: Step 8 of the basic flow of the use case: Wrong answer after check:

- The user is informed that he failed

Quit: If at any point the user wants to quit the system:

- The use case jumps to step 10 (The final step)

7.3 Key Scenarios

7.3.1 Scenario 1

Alex decides to use our system today. He first starts it up, and then picks out an algorithm he would like to be educated about. Alex decides that he would like to learn a graph search algorithm. He is showing how the algorithm works using a train line. Alex is then asked to run the algorithm manually. Alex successfully runs the algorithm manually. The system Congratulates Alex.

7.3.2 Scenario 2

Aurelia decides to use our system today. He first starts it up, and then picks out an algorithm he would like to be educated about. Aurelia decides that he would like to learn a graph search algorithm. He is showing how the algorithm works using a train line. Aurelia is then asked to run the algorithm manually. Aurelia fails running the algorithm manually. The system informs Aurelia that he failed.

7.3.3 Scenario 3

Tim decides to use the system today. Tim doesn't like how the system looks. Tim quits the system.

7.3.4 Post-conditions

Successful Completion:
The user is congratulated for successfully running the algorithm manually *Failure Condition:*
The use case informs the user that he/she has failed and skips to step 10