# Analysis

## About King’s

The King's School is a British co-educational independent school for both day and boarding pupils in the English city of Canterbury in Kent. It is a member of the Headmasters' and Headmistresses' Conference and the Eton Group. It is held to be the oldest continuously operating school in the world, having been founded in 597 AD.

In 2011, the school was subject to its latest regular, independent inspection. In summary, the inspection team praised the pastoral care system, the high academic achievements of all its pupils — irrespective of their age, aptitude or ability — and the happiness of the pupils. According to the Good Schools Guide the school is "Highly successful, producing excellent results." The Guide also stated that "You need to be creative, academically able and hard-working, as everything moves fast here

## About My client:

Dr Johnson is a teacher in the King’s school Canterbury, he received a PhD in maths. He is current an important member for the school as he is responsible for Oxbridge applications. His main fields of study in university was (Sorry, Mr. Wooldridge I need found out in my follow up interview as I cannot find any information about Dr. Johnson)

## Identify the Problem:

Many students and teachers find it very hard to locate the classroom due to the lack of a logical naming system of classrooms departments, and faculties. For instance, the main entrance of the school was named ‘Mint Yard gate’, this is extremely confusing especially for new pupils which do not have the knowledge about the history of the school. Also, on the time table most of the class room locations are represented by codes. These are useless for new students to know the position of the classroom. For example, the 5th geography classroom is categorised as J5 where the classrooms used to be a junior school. However, this is extremely counter intuitive as ‘J’ not related to geography department. For many teachers, especially new teachers, will find very difficult to found the classroom that they may need to be for a cover lesson. This is emphasised by my client Dr.Johnson.

## Interview with my client (Dr.Johnson)

Question 1: What do you want the navigation system to do? Features of the navigation system (navigation to specific Rooms? general direction? )

I want it to be an app that ideally navigate me from any given points of the school to another room.

Question 2: What are the desired input methods for the address?

For input methods, I want it to be optimised for single hand operations (one finger). As the current system has too many options that are not optimised. And I wish to use standard King’s short code as the once existed on the time table. Such as J5 for geography 5.

Question 3: What are the desired navigation methods for the address?

I have not specific request on the map/navigation, therefore it replies on the developer, however I do want it to be designed for new member of staff or navigating to an unfamiliar location.

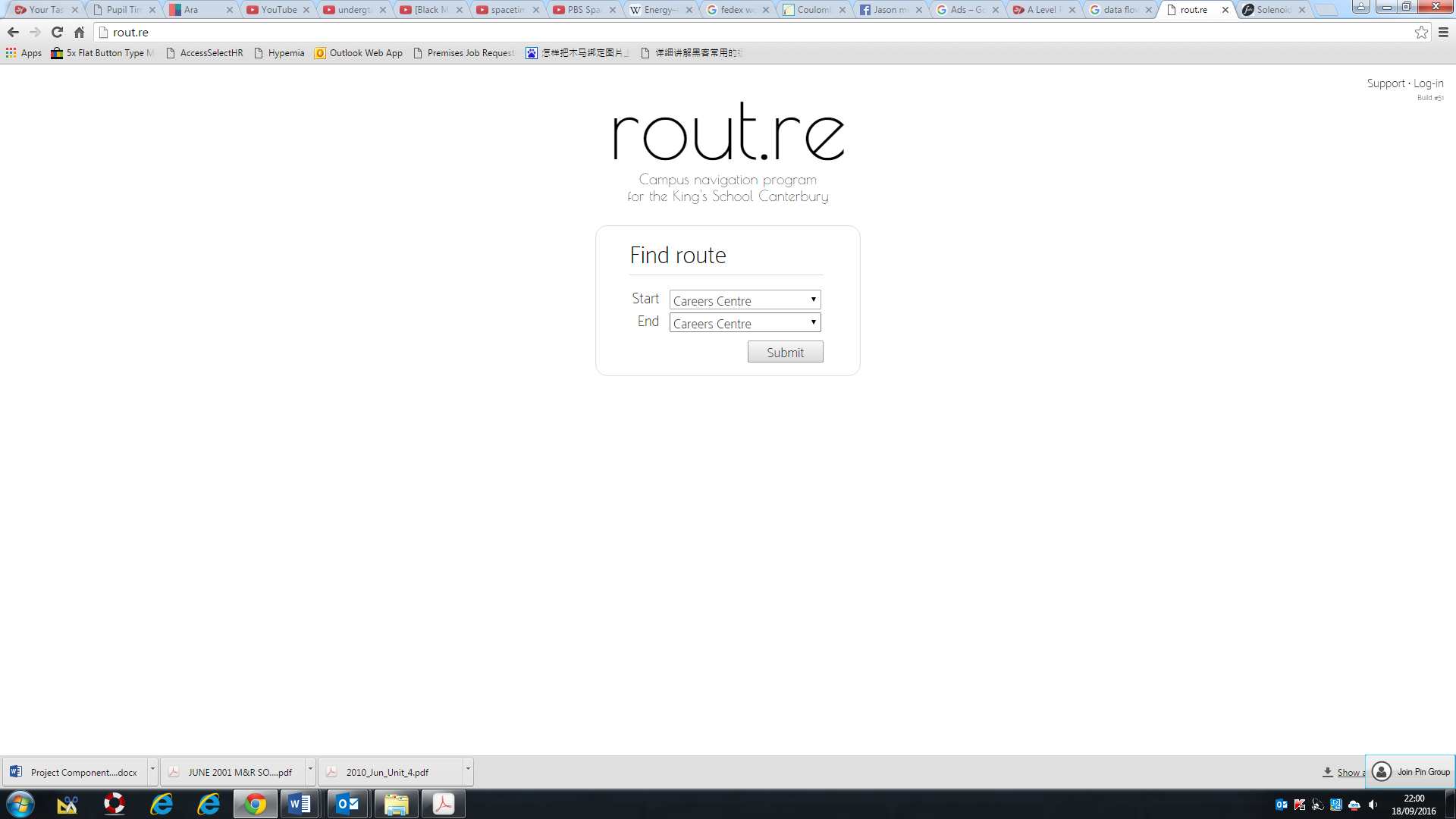
Question 4: Do you want a login system, which is connected to your school account, to store ‘my favourite location’

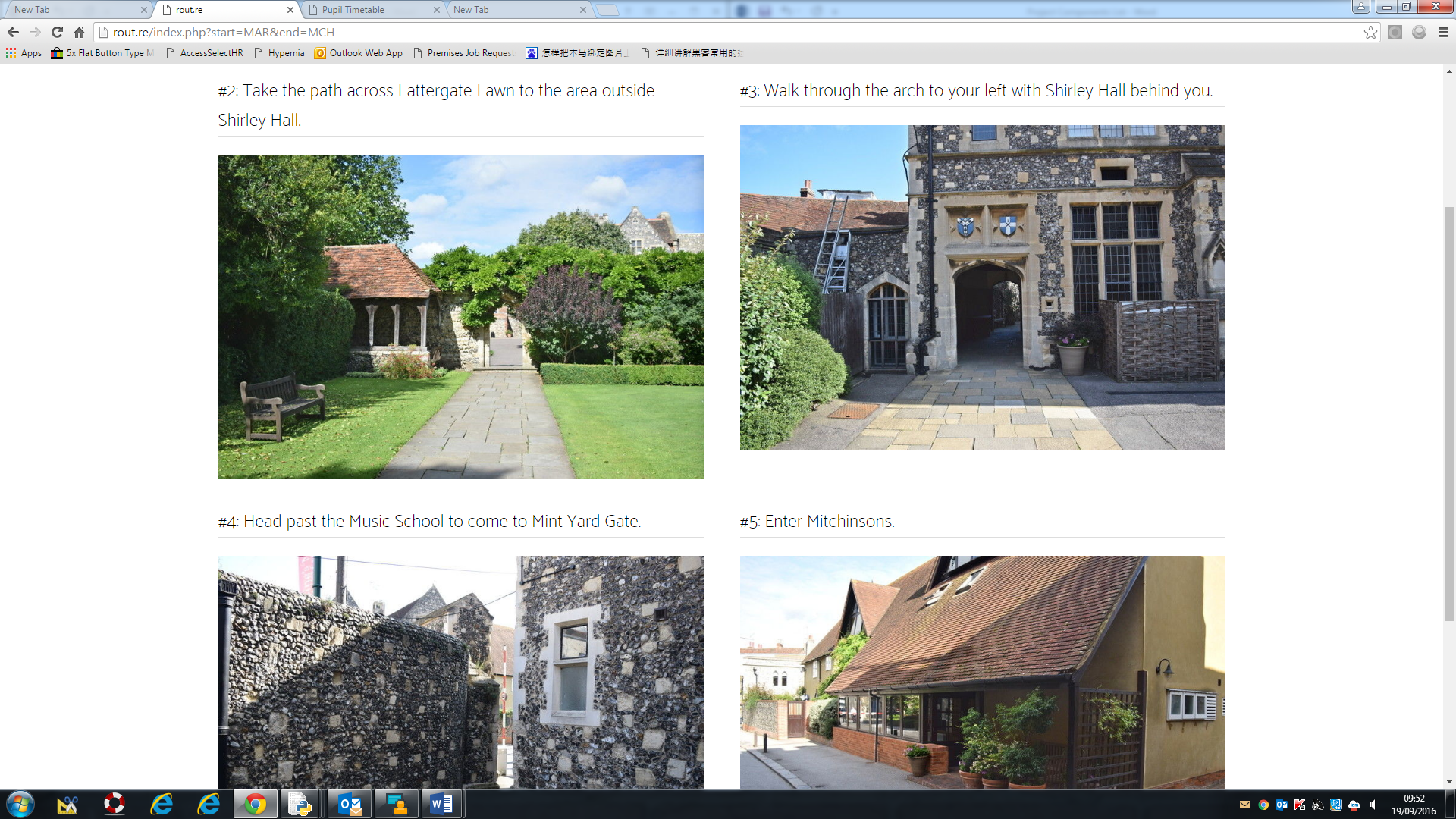
No as I don’t found them useful, as the system should be mainly targeting for cover lesson for teachers and new lesson.

## Current System.

(<http://rout.re>)

The current existed system consist of 2 interface. The first one consists a page where the user are requested to input the starting location and the destination.



After this interface, the next page will includes the pictured guide: 

The advantages of this guide are:

1. The current existing system has straight forward photos, which is helpful for new pupils to visualise the place.
2. A simple system to implement, and the system is relatively easy to use, as the images can be a very intuitive.

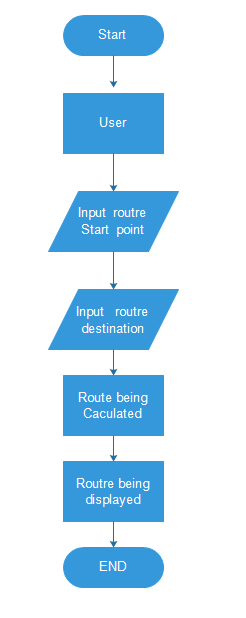
The Disadvantages of the current system are:

1. It is still dependent on text navigation where it might need to use the name of some places (such as Shirley hall ) to navigate the client.
2. The system only navigate user to the door step of the building, but not the rooms, this is extremely confusing for some complicated builds such as the palace block, where it has 4 entrance leading to different classrooms.
3. Some of the instructions are extremely unclear for new pupils such as fig.1
4. The system is not optinmised for portable device, as the webpage can be too big or too small depending on the devices

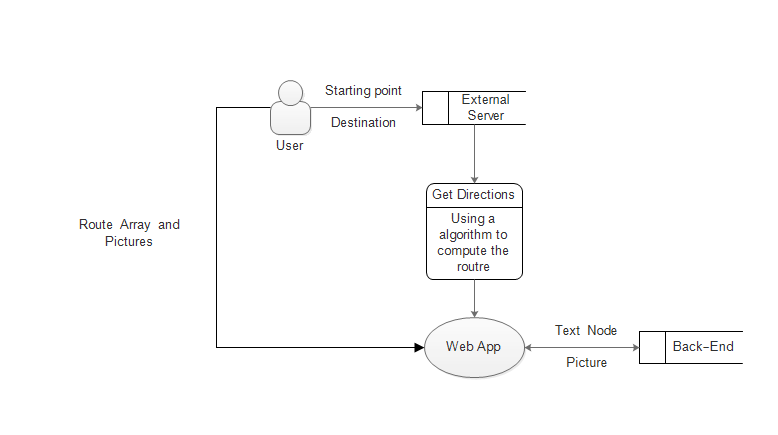


Figure

## **Current system flowchart**



## Data Flow of the current system



## Research On similar systems

One of similar navigation systems that has been publicly used are the google map, it is a commercial system developed by Google Inc. where it targets general public rather than specific school users.

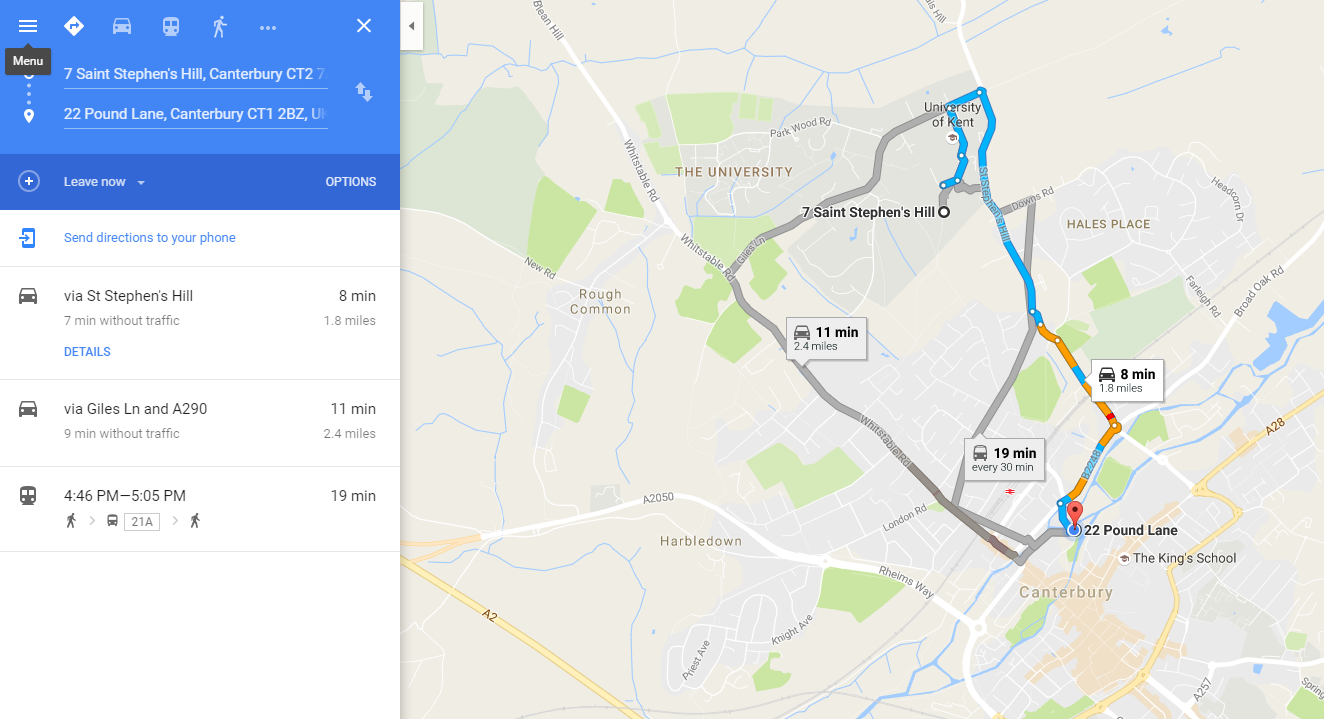
 Google map is an interesting example, as it contains many different methods and platform to navigate people. For example, it can be displayed on multiple platforms, webpage, IOS, Android and even windows phone.

Figure Google Map webpage

This solution is very complicated as it consists of many different languages and a global GPS navigation system. The interface is optimised according to using conditions, such as single hand operation on mobile platform (mobile phones specifically).

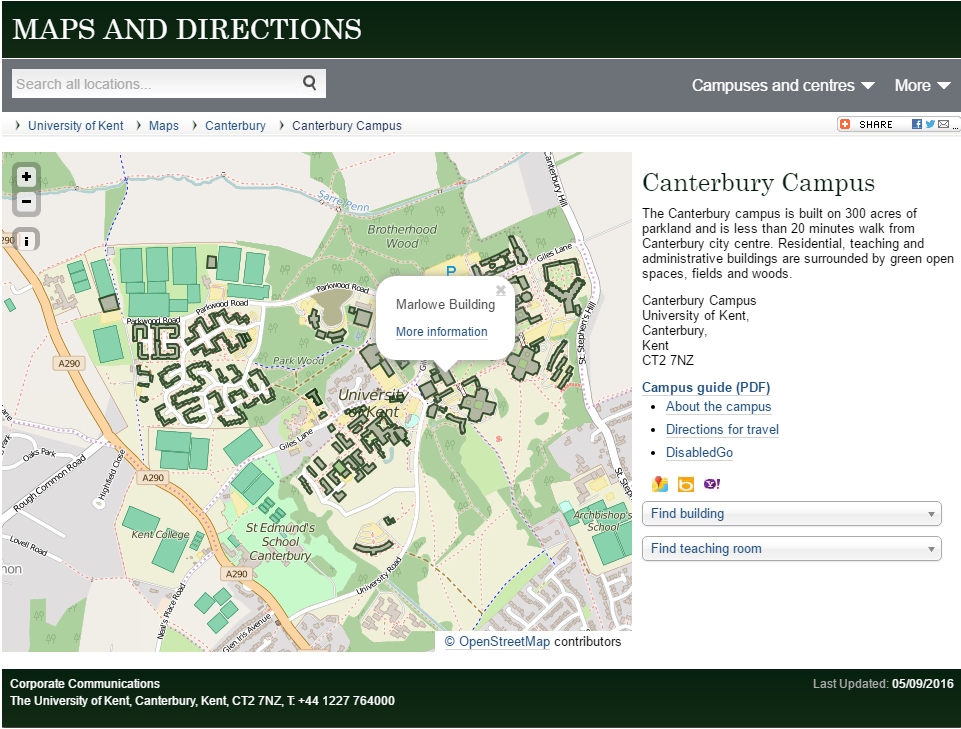
Another good example will be the Map for the Kent university. It is an map that targets university pupils and new visitors. 

Figure interface of the Kent U. Map

This map uses OpenStreetMap API therefore it relies on third parry. Also this solution does not have the navigation system like the google map as it is not intended for mobile use, therefore it does not support GPS signal input and calculation of a suitable path. The application is also not optimised for single hand usage, as this is not its intentional using condition. However, this type of method has serval type of benefits. Firstly, it requires a lot less space on both the client side and the server side, as no extra space is needed for storing information such as the path finding algorithms. In addition, this map is relatively simple to use due to only small amount of user input is needed.

## Potential solutions:

There are many potential solutions to the issues:

Potential Solution 1: Improve upon the current system including resign the back-end of the program so it will provide clearer instructions, reprogram some of the front-end GUIs to optimise the mobile using ability. However, this solution is not the best solution due to serval reasons. Firstly, this solution requires me to look into another people’s program, which might be an unwise option as changing an un-standardised code might cause confusion, and potentially produce more inefficient ‘spaghetti code’.

Potential Solution 2: Improve off the shelf navigation system such as google map, by submitting extra information to third party apps such as Google Maps or Apple Maps. This is not a good solution as this will rely on other companies to improve the service. Also due to security reason it might be unwise to share the information about the classrooms onto the internet for public access.

Potential Solution 3: Programme an application that are optimised for mobile usage which are simple to use and accessible by members of the school. This option is fine as the developer can optimise the program to his desire without providing information to third parties, and can prevent the problem of non-efficient spaghetti code due to the differ in styles of coding.

## Chosen solution and justification

I have chosen Solution 3 as my final solution as it allows me to optimise the program to his desire without providing information to third parties, and can prevent the problem of non-efficient spaghetti code due to the differ in styles of coding. Also, this will also allow me to add further modification when it is needed rather than relying on a pre-opened API. Also Solution 3 can be run in one of the school servers which will decrease the response time, which could be important for a teacher going to a cover lesson or a student desperately looking for his classroom.

## Prospective users

This Project should be mostly being used by the member of the Kings school Canterbury. As it is a school project. Most users are new students that are unfamiliar with the school facility and needs a method to navigate them from classrooms to classrooms. In some situation, some teachers that need to go to a new classroom might use this application.

## Questionnaire completed by users about layout

Question 1: Which types of GUI would you prefer to use when inputting the data into the application (Drop down menu/Clicking Icons/scroll menu/manually had surveyed 60 people of the Linacre house of the Kings school Canterbury to acquire information.typing).

As shown in the diagram, most people perfer manual input as they allow the use of standard Kings code for classrooms. However a significant proportion of pupils also perfer drop down menus, as it is simpler to use and easier to control.

Question 2: What is the preferred style of GUI (simple/futuristic/ industrial)

It is rather clear that out of the 60 person I have serveyed 33 of them perfer a clean GUI as it is not only easier to use, but also it is more aestheticly pleasing in daily usage. Also Clean interface prevents confusion that might have when more comlicated GUI is used, as this application is targeted at amature users that do not have a high level of understanding on how program works, clean GUI is a good design to go.

Question 3: Style of the map ( satellite view/ Map view)

As the pie chart has showed, most of the Linacrians prefer the street view of the map as they think a satellite view will not provide good enough benefit for school navigation.

## Acceptable limitations:

|  |  |
| --- | --- |
| It cannot navigate user to individual classrooms | The application does not provide information about complete layout of a complicated building Therefore, it is impossible to navigate user to different classroom in a complex building |

## Data Volumes

I predict that the size of the application will not be very big on the client side, as most of the data is stored on an external server. The most likely size for an Android Application package(.apk) will be less than 20MB. However due to the amount of the pictures and/or other information needed for the navigation the server side of the application will be bigger, occupying less than 200 MB of storage. Within the android app the application will mainly contain the script for the application to acquire information from the API of the server. This allows me to reduce the app-size as most mobile devices do not have as many storage space as a desktop or laptop. On the server side where most information will be storage, it mainly contains pictures, nodes for path finding and path finding algorithm.

## SMART OBJECTIVES

1. The application must navigate the user to specific locations within the school bounds.
2. The application must produce clear text instructions for user to understand.
3. The application must contain GUI with a map.
4. The application must indicate the position of the user on a map.
5. The application must show a visible path on a map GUI.
6. The application must show a realistic path on the map GUI. (For example, NOT APPLE MAP)
7. The application must be suitable for single hand operation.
8. The application must accept standard King’s short code for classrooms as a valid method of input.
9. The application must have the option to use text bar as a method of input.
10. The application can use dropdown menus for destination input.
11. The application must be simple to use, without user needed to input to much complicated data.
12. The application must be able to recognise different entrance of the same building.

## Research Documents

|  |
| --- |
| Bird site view of the Kings School Canterbury |
| Code list for all the classrooms |
| Position of all the entrance of the school |
| Position of the entrance of the palace Block |

## Path Finding Algorithms that can be used

During the Investigation I have come across several well-known algorithms that can be helpful for this particular project.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Advantages | Disadvantages |
| A\* algorithm | A\* algorithm is a best-first algorithm where it will solve problems by searching all possible paths. It works by comparing 2 different value of any node. Its g cost (Cost of path from the start node to end node), h cost (heuristic that estimate the cost of the cheapest path from current node to goal.) and F cost where is the combination of G cost and F cost. By following the nodes with the lowest F cost it is guaranteed to find the path with the smallest distance. | * Efficient * Guarantee shortest path * Allow duo direction graph traversal * The algorithm takes as worst case performance. | * Unnecessarily complicated for a small graph of a school map. * The raw algorithm is not adapted for node graph traversal, but more of a grid graph traversal, which means more time and resources must be spend on adapting the algorithm. |
| Dijkstra’s Algorithm | Dijkstra’s algorithm is a greedy algorithm that is guaranteed to find the shortest path from point A of a graph to another point on B. This algorithm explores the nodes one by one until find a path that connects the start to the end. | * Dijkstra’s algorithm is designed for node base traversal. * Dijkstra’s algorithm is relatively fast for small graphs. * Dijkstra’s algorithm is relatively simple to implement. * Dijkstra’s algorithm is guaranteed to find the shortest path. | * Dijkstra’s Algorithm requires a lot of system resources. * It is extremely inefficient requiring an time for worst case. |
|  |  |  |  |

# Design

## Input Output Process chart

### Initial Page

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Outputs |
| User clicked the ‘start navigate button’ | System get the value for the target destination and end destination. Store the value into a storage space, where a function can access in the next page | The Sever, Local stack | Close the current page and opens a new page where the map will be displayed. |
| User input the target destination | Validation process  Also upload the value to once start navigation button is pressed | The server | Output to Local stack, where a value will be stored for the next page |
| User Input invalid address for destination | Validation process | The server | Error message, noticing the user and requesting |
| User input Starting Location | Validation process and upload the value to local stack | The server | Output to Local stack, where a value will be stored for the next page |

### Map Page

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Outputs |
| Page Opened | Read value of the Starting location and the Target location.  Instance Object <canvas> where be map will be drawn on. | The server, and local stack | Store Values on the local JavaScript variable |
| <Canvas> instanced | Load up the base image for the map with desired dimensions. | On client’s device as it need to be rendered. |  |
| Starting Location and Destination is store into variable from local storage | Pass the variable into the Dijkstra’s algorithm to find a suitable path. | The server, variables | N/A |
| Location Variable is passed into the Pathfinding function | By using Dijkstra algorithm, and the node data base to compute a path for the user and store the path as list of string | Client’s device | Calculated Path as a list of node names where it will be used for further rendering. |
| Path being passed in | To a canvas drawing program, where the track will be draw onto the canvas. | The Server, Local Browser renderer. | Image being rendered, and path being rendered |
|  |  |  |  |

## Designed System Flowchart (Overall)

## Data Flow diagram (Proposed System)

## Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data type | Data content type | Length | Description |
| Start location (From menu) | String | N/A | 2 | The standard code used in the program for name of the classrooms. Select by user as the starting location |
| End Location  (From menu) | String | N/A | 2 | Standard code used in the programme for name of the classrooms. Select by user as the destination |
|  |  |  |  |  |

## Top Down Diagram

## OOP Class Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Parameter pass In when Instanced | Attributes | Functions/Subroutines | Description |
| Nodes | name;  connect;  Coordinate;  Type;(Optional) | name : name;  connect : connect;  distance : 1;  temp : Infinity;  perm : null;  ord : null;  retracevisited : false;  p1 : Coordinate;  type : type ; |  | This is the base object for the Nodes or way points for the path navigation and rendering the path onto canvas. |
| User | N/A | Start  End |  | User’s selection will be handled as User.Start and User.End |
| canvas | N/A | Img : new Image() | Img.onload = Load image the moment the Canvas is instanced. |  |

The OOP design will also be constantly updated during programing.

## Pseudo Code for Key Functions

### Dijkstra’s Path Finding algorithm

Function Dijkstra(Start,End):

Start.Permenant\_label = 0

Start.Order\_Label = 1

Push Start to Visited List

Read List Start.Connected\_Node

For all Start.Connected\_Node

Start.Connect\_Node.temp = Distance of Start.Connect\_Node From Start

End For

Select Start.Connected\_Node with Smallest temp value Call it CurrentNode

Loop Until End.Order\_label is not null:

CurrentNode.Permentant\_Label = Distance + Last\_Visited\_Node.Permenant Label

CurrentNode.Order = n where n is last Node’s Order\_label +1

Push CurrentNode to VisitedList

For all CurrentNode.Connected\_Node

CurrentNode.Connected\_Node = Permentant label of CurrentNode + distance from CurrentNode to Connected Node

Select CurrentNode.Connected\_Node with Smallest temp value Call it CurrentNode

End Loop

Push End into Track\_List

Loop Until Current Node is Start Node

Find Node Where Node.Perm = End.Perm – Distance to Node

Push Node to Track

End Loop

Reverse Track

## Security and Integrity of data

Most data used in this project is not extremely sensitive, but it can still cause some safety concern as

the data will be tracked, and recorded, which means it is possible to work out exactly where the user

of the app has gone within the day. This will be alarming as it can be considered as an privacy

hazard. Most the data transmitted between server and client should be encrypted to prevent un-

Faithful usage

## HCI Designs

## Storage Medium

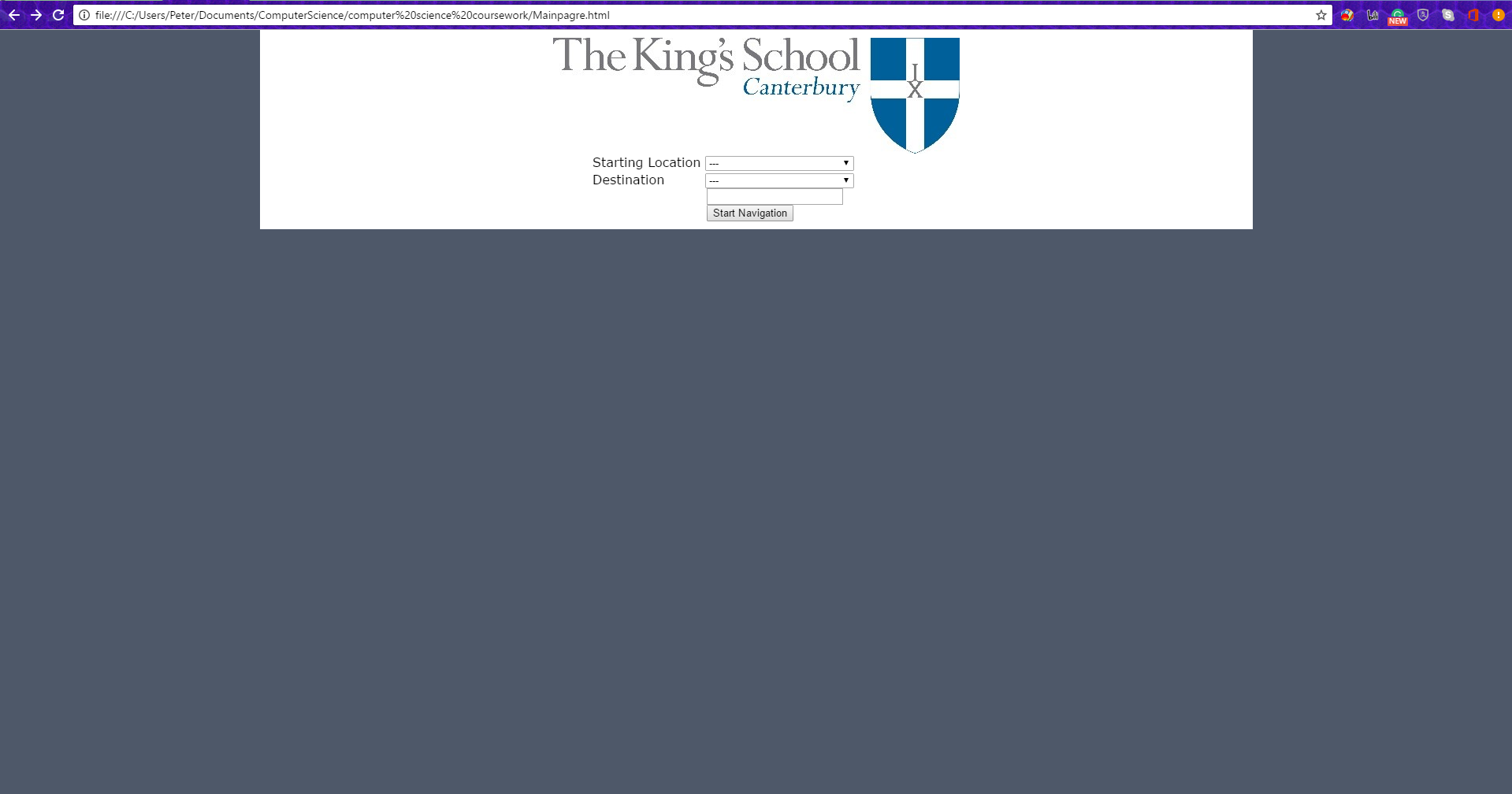
## Test Plan

|  |  |  |
| --- | --- | --- |
| Test | Methods |  |
| IF the main menu can be successfully displayed | Start the app, and wait for the main menu |  |
| If the start navigation button is functional | Press the start navigational button, and test if it leads the user to the navigation page. |  |
| If the GPS map function properly | By programing a beta version and distribute among selected people, it is possible to gather if any of the path seems to be dysfunctional |  |
|  |  |  |

# Technical Solutions

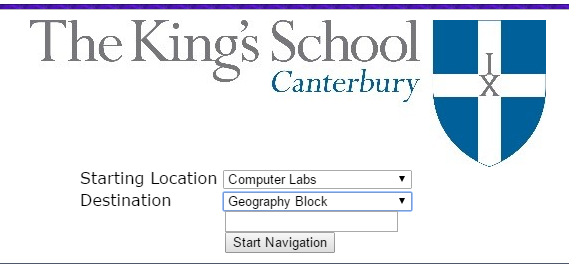
## Screen shot of running program

### Idle Main page



Picture : Initial page

### Information inputted



①

④

②

③

Picture : User Input Data

1. King’s school Canterbury Logo
2. Starting location box
3. Destination Box
4. Button to submit data

### Map Produced



Picture : Map

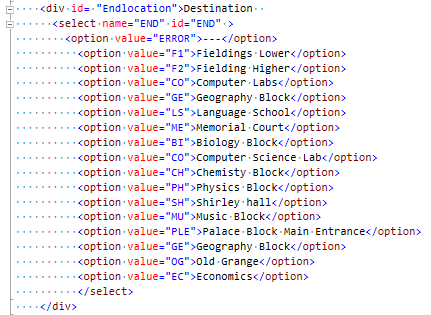
Green Dot: Starting Location

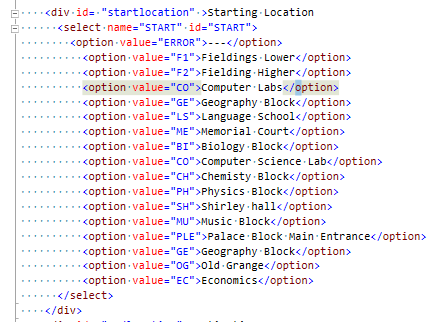
Red Dot: Destination

Blue Path: Recommended Path

## Technique Used

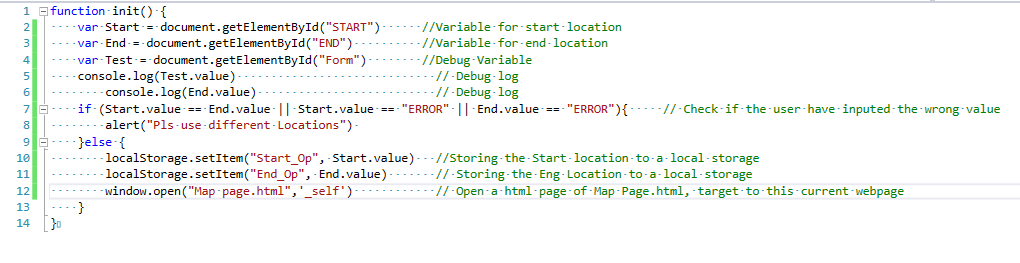
### Picture 1 & 2

For the HTML page, I have used many HTML elements to acquire information regards to the user’s choice.



Code : Starting location Menu, End Location Menu, the button and the .JS being used

In this page I have used two <div> each with ID of ‘startlocation’ and ‘Endlocation’ Both containing a select menu with id “START” and “END”. The options are same. Once the user made the selection and press the “Start Navigation Buttom, it will call function ‘init()’ in ini.js where the follow action will be done.



### Map Page (Picture 3)

Once the new Map page.html is opened this is what will happen.



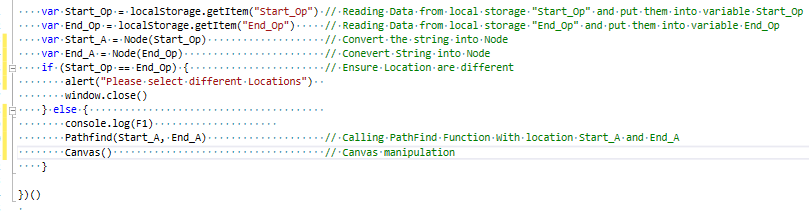
Code : Canvas Creation

The Webpage will first create an canvas which is where the main map will be stored. This has an 1054x917 sizes as this is the native resolution of the over view map provided by the school. I manual restricted the size to sure maxium resolution for diffferent device.



Code : Java Script load up

After the canvas is created, it will load up a JavaScript document. Where it contains a subroutine that will be called the moment the Script is loaded.



This subroutine will call in a function called Pathfind() where it is where the Dijkstra’s algorithm is.

function Pathfind(Start, End) {

console.log('---1. Assign Temp to Start---')

Start.temp = 0

console.log("---2 Assign Perm to Node---")

AssignPerm(Start) // Assign permanent value to the current node

visited.push(Start) // Push the current Node to the visited list, so that the algorithm will not visit it again

console.log("---3 Creating Connection array---")

PushConToTemp(Start) //Reading the connection array and push it to a temporary node list, where in this list, every node is has an temporary value

console.log(TempNodes)

console.log("---4 : Assign Temp value---")

for (i = 0 ; i < TempNodes.length ; i++) {

Assigntemp(Start, TempNodes[i]) // Assigning temporary value to all connected Node

}

console.log(TempNodes)

console.log("---5: Bubble Sort TempNodes---")

bubbleSort(TempNodes) // Find the node with smallest Node using bubble sort

console.log(TempNodes)

console.log(End)

while (End.ord == null) { // looping this algorithm until the Destination has an permanent value

console.log("---6: Current node -> smallest temp---")

var Current = TempNodes.pop() // Go to the next node

visited.push(Current) // Push it to visited node

console.log(TempNodes)

console.log("Current Node:" + Current.name)

console.log("---7: Assign Permentant value for the current node---")

AssignPerm(Current) // Assigning permanent value to the node

console.log("Next Order: " + Ord)

console.log("Current Order" + Current.ord)

console.log("---8: Push the new connected to the Templist---")

console.log(Current)

PushConToTemp(Current) // Push the connected node to the temp list

console.log(TempNodes)

console.log("---9: Assign Temp Values to the new Node---")

AssignConnectedtemp(Current) // Assign new temp value to those nodes

console.log("---10: Bubble Sort TempNodes---")

bubbleSort(TempNodes) //Sort the List again

console.log("Current Node:" + Current.name)

console.log(TempNodes)

console.log(visited)

}

//RETRACT FROM HERE

track.push(Current.name)

retraceLastNode(Current, Start) // Retrace From last node

track.reverse() // Reverse the list so it starts from the Starting Point

for (i = 0; i < track.length ; i++) {

track\_Node[i] = Node(track[i]) // As PathFind() only return an String version of the path, I must convert time to node and store them in a separate variable call track\_Node

}

return track // Return the Path

}

The code above is my implementation to the Dijkstra Algorithm; this algorithm also has the following subroutines.

function PushConToTemp(NodeName) { // Pushing all the connected node of the current node to a temperary Node list

for (i = 0 ; i < NodeName.connect.length ; i++) {

if (TempNodes.indexOf(Node(NodeName.connect[i])) == -1) {

if (visited.indexOf(Node(NodeName.connect[i])) == -1) {

TempNodes.push(Node(NodeName.connect[i]))

}

}

}

}

function Assigntemp(LastPerm,NodeName) { // Assigning temporary value to all the nodes in the temp list

if (NodeName.temp >= (LastPerm.perm + NodeName.distance)) {

NodeName.temp = LastPerm.perm + NodeName.distance

}

function AssignConnectedtemp(CurrentPermNode) { // Assigning temperay value to all connect node that exist in the connect list.

for (i = 0; i < CurrentPermNode.connect.length ; i++) {

var ConnectedNode = Node(CurrentPermNode.connect[i])

if (ConnectedNode.temp >= (CurrentPermNode.perm + ConnectedNode.distance)) {

ConnectedNode.temp = CurrentPermNode.perm + ConnectedNode.distance

}

}

}

function bubbleSort(NodeList) { // Simple bubblesort algorithm

for (i = 0 ; i < NodeList.length - 1 ; i++) {

for (j = 0 ; j < NodeList.length -1 ; j++) {

if (NodeList[j].temp <= NodeList[j+1].temp) {

var a = NodeList[j]

NodeList[j] = NodeList[j + 1]

NodeList[j+1]= a

}

}

}

}

The reason I use bubble sort to sort my ‘temp NodeList’ is that this the list will contain only 31 items in the theoretical worst case, which is highly unlikely due to some of the nodes only connects to a single entrance/exit which makes them worthless for this algorithm. Although it is possible to use other sorting algorithms such as quick sort or binary sort, there will be no noticeable difference in performance, and system resources to carry out the sorting action on a list with the size less than 30, therefore I have decided to use bubble sort, as it is the quickest algorithm to implement.

function retraceLastNode(node,Start) { // This subroutine retrace the using temp values and perm order to trace the shortest path for the node connection

console.log('Current Node:' + node.name)

console.log('This is the ' + n + ' Recursion')

n = n +1

for (i = 0 ; i < node.connect.length; i++) { // Looping for all the node that is connected to the current node

var ConnectedNode = Node(node.connect[i])

if (ConnectedNode.retracevisited == false) {) // Check if this node has being retraces

console.log("d1")

if (ConnectedNode.perm == node.perm - ConnectedNode.distance) { // Check if the perm number of the node is equals to last node's permanent value - the nodes distanc

console.log(“d2”)

track.push(ConnectedNode.name) // push this push to the track list

ConnectedNode.retraceLastNode = true // Flag the node as visited

if (ConnectedNode != Start) { // Check if the retrace algorithm has reached start node

console.log("d3")

console.log(ConnectedNode)

retraceLastNode(ConnectedNode,Start) // If not then I will use recursion, call retaceLastNode subroutine using the current connected node

} else if (ConnectedNode == Start) { // if the node has return then leave subroutine

break

}

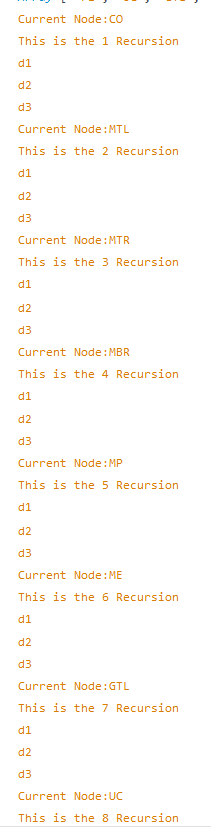
break}

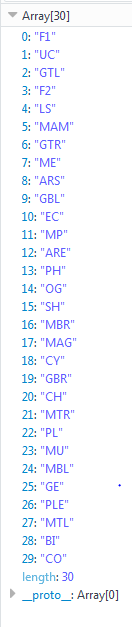
}

}

}

This is a recursive algorithm where I have used to retrace the path of the algorithm. It will start with current node, find the next node that has an permanent value equal to the current’s permanent value minus the distance to the connected node. If the algorithm have not reach, the Start node it will call onto itself and use the connected node as the current node, and work its way to the start Once the Algorithm is set





The console log on the left shows an example of the of a recursion search from the visited list (Array[30]) use recursion to find a possible path from F1 to CO.

For this situation the function called it self 8 times to find a valid path.

The function below is a function that will return the node from the node data base when I input the node name into the function as string.

function Node(input) {

if (input == "EC") {

return EC;

} else if (input == "GTL") {

return GTL

} else if (input == "GTR") {

return GTR

} else if (input == "LS") {

return LS

} else if (input == "UC") {

return UC

} else if (input == "F1") {

return F1

} else if (input == "F2") {

return F2

} else if (input == "ME") {

return ME

} else if (input == "PH") {

return PH

} else if (input == "BI") {

return BI

} else if (input == "CO") {

return CO

} else if (input == "CH") {

return CH

} else if (input == "SH") {

return SH

} else if (input == "MU") {

return MU

} else if (input == "PL") {

return PL

} else if (input == "GE") {

return GE

} else if (input == "GLT") {

return GLT

} else if (input == "GBL") {

return GBL

} else if (input == "ARS") {

return AR\_start

} else if (input == "ARE") {

return AR\_End

} else if (input == "CY") {

return CY

} else if (input == "MTL") {

return MTL

} else if (input == "MTR") {

return MTR

} else if (input == "MBL") {

return MBL

} else if (input == "MBR") {

return MBR

} else if (input == "MP") {

return MP

} else if (input == "MAM"){

return MAM

} else if (input == "MAG") {

return MAG

} else if (input == "OG") {

return OG

} else if (input == "GBR") {

return GBR

} else if (input == "PLE") {

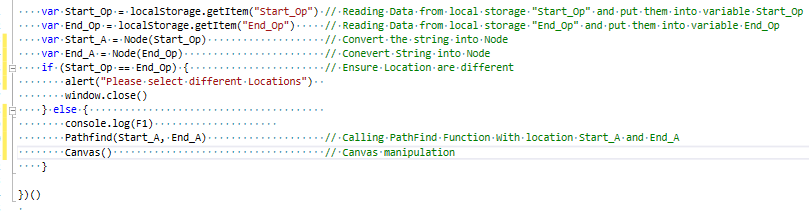
return PLE

}

}

Which is useful in the algorithm as at the last step it is necessary to access the node attributes of a particular node.

Once The Path Finding algorithm is finished, it will carry out another subroutine call canvas ()



function Canvas() { // Get Canvas

canvas = document.getElementById('canvas'); // Store the central canvas as a variable

ctx = canvas.getContext('2d') // get the 2d context for the canvas to draw the map

loadImg() // load the image for the canvas

};

This is the getting the canvas as an object in the java script

function loadImg() { //Load Map Image

img = new Image() // defineing object img

img.src = "map.png" // Source file of the image is map.png

img.onload = function () { //function fo drawing paths on the image

imgIsloaded = true;

drawImg()

}

}

This subroutine loads the image for the map and triggers the next step of actually drawing the image onto the canvas.

function drawImg() { // Draw the image Onto the Canvas

ctx.clearRect(0, 0, canvas.width, canvas.height)

ctx.drawImage(img, 0, 0) // drawing image with object img, at 0,0

drawmap(track)

}

After the image is drawn onto the canvas drawmap(track) draws the path returned from Dijkstra’s algorithm into visualised lines and dots onto the canvas

function drawmap() {

var x = 0 //index for the x co-ordinate

var y = 1 // index for the y co-ordinate

var X\_Begin = Node(track[0]).p1[x] // Value of x coordinate

var Y\_Begin = Node(track[0]).p1[y] // value of y coordinate

var End = Node(track[track.length-1]) // End node

ctx.lineWidth = 5 // Sytle of the line

ctx.lineJoin = 'round' // Line Join Type

ctx.strokeStyle = '#66ccff' // Colour of line

ctx.beginPath(); // Start drawing

ctx.moveTo(X\_Begin,Y\_Begin) // Move pen to the Starting point

for (i = 1; i < track.length ; i++) {

ctx.lineTo(Node(track[i]).p1[x],Node(track[i]).p1[y]) // Connecting all the nodes in the path list

}

ctx.stroke() // Finish drawing

// Start label

ctx.fillStyle = '#00FF00' // Colour of Start Point

ctx.beginPath()

ctx.arc(X\_Begin, Y\_Begin, 5,0,Math.PI\*2) // Draw a circle to indicate the starting point

ctx.fill(); //fill the circle

ctx.fillStyle = "#FF2121" // Colour for the End Point

ctx.beginPath();

ctx.arc(End.p1[0], End.p1[1], 5, 0, Math.PI \* 2) // Draw a circle to indicate the Ending point

ctx.fill() //fill the circle

// End label

}

## Data Base of the Nodes

All Nodes follows the following pattern

function node(name,connect,Coordinate,type) { // type is an optional attribute as it is meant for future expansion and denotation

this.name = name; // name of the node

this.connect = connect; // list of connected nodes

this.distance = 1 // Distance

this.temp = Infinity // temporary value

this.perm = null // Permanent value

this.ord = null // Permanent order

this.retracevisited = false

this.p1 = Coordinate // coordinate of the node

this.type = type // type of node

}

Where all Data is stored as a new node

//////////////////////////////////////////////Way Points//////////////////////////////

var AR\_start = new node("ARS", ["GTL", "GBL","ARE"],[503,470])

var AR\_End = new node("ARE",["ARS","SH","CY"],[210,466])

var MTL = new node("MTL", ["CO", "MTR","BI"],[304, 78])

var MTR = new node("MTR", ["MTL", "MBR"], [403, 75])

var MBL = new node("MBL", ["BI", "CH", "MBR"], [293, 134])

var MBR = new node("MBR", ["CH", "MTR", "MBL", "MP"], [385, 192])

var MP = new node("MP", ['MBR', 'ME', "MAM"], [391, 232])

var MAM = new node("MAM", ["MP", "GTL", "OG"], [421, 275])

var MAG = new node("MAG", ["OG", "MU"], [256, 298])

var GTL = new node("GTL", ["UC", "ME", "MAM","ARS"], [516, 307])

var GTR = new node("GTR", ["LS", "EC"], [844, 367])

var GBL = new node("GBL", ["ARS", "GBR"], [477, 470], [508.583])

var GBR = new node("GBR", ["GBL", "GTR"], [872, 563])

//////////////////////////////////////Locations//////////////////////////////////

var EC = new node("EC", ["GTR"],[815,262])

var LS = new node("LS", ["GTR", "UC"], [776, 335])

var UC = new node("UC", ["F1", "F2","GTL","LS"],[690,321])

var F1 = new node("F1", ["UC"],[706,273])

var F2 = new node("F2", ["UC"], [706, 273])

var ME = new node("ME", ["PH", "GTL","MP"],[482,247])

var BI = new node("BI", ["MTL","MBL",],[269,143])

var CO = new node("CO", ["MTL"],[284,40])

var CH = new node("CH", ["MBL","MBR"],[320,198])

var PH = new node("PH", ["ME"],[519,182])

var SH = new node("SH", ["ARE","CY"],[210,466])

var MU = new node("MU", ["CY","MAG"],[166,361]) //need

var PL = new node("PL", ["CY", "GE","PLE"],[156,578]) // need

var GE = new node("GE", ["PL"],[178,702])

var CY = new node("CY",["ARE","PL","MU","SH"],[157,479])

var OG = new node("OG", ["MAM", "MAG"], [310, 323])

var PLE = new node("PLE",["PL"],[88,572])