

Map & Location programmed by Dimitar

As shown in Figure 1, iWEB has a location tracking system that allows the user to find the locations of all of the fountains, bus stops and bins on the Exeter campus. The user can decide which items appear on the map by pressing the buttons on top of the screen. The red dot shows the user's location, the user can turn on and off location tracking whenever they want. When the user presses on a location they would like to go to, the information about that location is retrieved from the database, as well as an option to find the shortest path to that location and an option to verify that the user has used the location in order to earn points.

Design Choices:

Gamification:

The map incorporates gamification by rather than displaying an API of the campus, I am displaying a more colourful and a more gamelike map.

Reliability:

The map system was created in a way to not be reliant on other APIs, that way if they update their code or stop working this map system isn't affected. I programmed everything from the path finding algorithm to even draw the map of the university. The only external program used is GeoLocation.

Scalability:

The map incorporates scalability in several ways, one of which is that the items (water fountain, bin... ect) are all displayed automatically meaning as soon as they are added by users of this app they appear here, this is scalability since there wouldn't be a problem for this app to be incorporated and expanded to more campuses or even large cities.

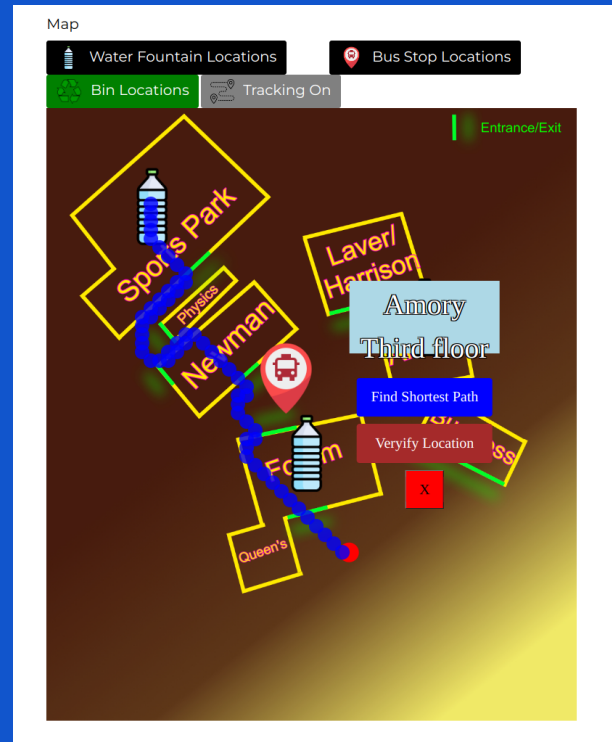


Figure 1

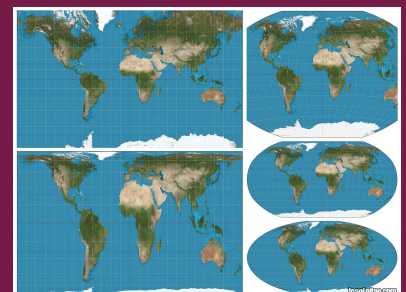
Technical Explanation:

Map Projection:

The mathematics behind the problem:

The problem of projecting a sphere onto a plane has been around for centuries, and it is impossible to do it 100% mathematically accurate, however the system I have developed is far sufficient for relatively small areas such as a University campus. I have used [Equirectangular projection](#) which is the most common way of representing maps onto a plane. Equirectangular projection has 2 equations which turn longitude and latitude into X and Y coordinates, however that gives the X and Y coordinates relative to the whole globe which in this case is useless. My system incorporates this by using the 2 formulas to calculate the X and Y coordinates of the top left and bottom right coordinates then calculates all of the other locations by

representing them as a percentage relative to the 2 corners. What this allows me to do is to display the locations onto a map without having to display the map of the whole world, I can just display the area of the university. Using online tools I found the longitude and latitude of the corners of the Exeter Campus, the longitude and latitude for the other items is accessed automatically. Since I am only mapping the surface area of the Exeter campus I don't need to



$$x = R(\lambda - \lambda_0) \cos \varphi_1$$
$$y = R(\varphi - \varphi_0)$$

- λ is the **longitude** of the location to project;
- φ is the **latitude** of the location to project;
- φ_1 are the standard parallels (north and south of the equator) where the scale of the projection is true;
- φ_0 is the central parallel of the map;
- λ_0 is the central meridian of the map;
- x is the horizontal coordinate of the projected location on the map;

use the central parallel of the map, and I have taken the “standard parallels” to be the corners of the map to improve the accuracy, in the future I will add more parallels to improve the accuracy further.

Displaying the locations and the user:

All of the locations for the different items are stored in the database locations, the X and Y coordinates are calculated and displayed on the map. To display the user using [Geo Location](#) I retrieve the longitude and latitude of the user then convert them to X and Y coordinates. The distance between each item is pretty much as accurate as GPS technology is.

The brown-yellow Map:

The map being displayed (this is subject to change) is a drawing of the Exeter Streatham campus, it shows some of the main buildings, and allows the user to see where they are.

Location verification/Tracking:

Tracking:

I have given the option to turn on and off when the app tracks their location, this is in order for the user to feel safe to use our app but don’t want to be tracked as they are walking. This system is possible because [Geo Location](#) has an option for tracking or to just return the location once. If the user presses the grey button then their location is tracked if they turn it off then then tracking is turned off.

Location verification:

Since our app is based around completing challenges (for example ‘use a certain bin’), I have implemented a system that verifies if the user is close to the item in the challenge. This is done using [Geo Location](#) by taking a one time measurement of their location, then that location is checked to see if the user is close enough with the item.

Path Finding:









The maze representation of the map:

The current system is temporary and will be improved since it isn’t scalable globally. To find the shortest path the algorithm needs to know where it can and can’t go, to do this I have created a text file representation of the map that is drawn which is completely fine for the Exeter Campus, however if you wanted to use this system for a city like London or Paris it would take too much time to write the text representation manually.

Shortest path algorithm:

The current path algorithm will also be updated since it isn’t fast enough, however you run into a problem with mathematics in general as a perfect shortest path finding algorithm doesn’t exist. For now the system currently uses [The A Star algorithm](#) , I have adapted the pseudo code from Wikipedia into JavaScript. I have however altered it to not actually find the shortest path but to find a path that is close to the shortest, this is because the actual algorithm uses too much memory and since the only goal of this path finding algorithm is for visualisation there is no point for it to be exact it can be an estimate. The way I have done this is by eliminating child/neighbour nodes unless they are the end node, for the purpose of visualisation this is completely fine.

The Buttons explained

Functionality	Not pressed	Pressed
Displays the fountains (water bottles) on the map.	 Water Fountain Locations	 Water Fountain Locations
Displays the bus stops on the map.	 Bus Stop Locations	 Bus Stop Locations
Displays the bins on the map	 Bin Locations	 Bin Locations
When pressed, it turns on tracking mode and updates the user’s icon automatically.		 Tracking On

When pressing the icon of a location the user sees 4 things. The first is a box that displays the information stored in the database of that item. The second button generates the path, the third button checks the user's location with the item shown, this is to check wherever the user has actually used the item and finally a button that gets rid of the 4 things.

