Development of a Route Finding Web Application for DSTL

Thomas Ardern Daniel Harper Molly Gamble
Robert Berry Etienne Laurent Deborah Oye

{firstname}.{surname}@myport.ac.uk
Portland Building, Portland Street, Portsmouth, Hampshire, PO1 3AH

1.0 - Introduction:

The Defense Science Technology Laboratory (DSTL) develop systems and technology for use within the defence and security sectors. The task was to develop a web application which can recommend the best routes across a terrain with respect to a variety of different set hazards and variables.

Two nodes are connected together by a path. Attributes will need to be set for both paths and nodes. Attributes represents the various hazards and variables which can affect the routing algorithm. As the route is calculated mediators (functions) can affect any number of attributes depending on the calculations up until that path and the path's own attributes.

1.1 - Advanced functionality

The finished artifact ideally needs to take many variables into account when calculating a route, such as number of troops and path constraints. Attributes set may impact the suitability and reliability of a route and so need to be taken into account when it is calculated. The web app should be developed as a framework in order to allow for expandability, new features and to suit a wide variety of route finding.

To make the path nodes more accessible, the nodes ideally need to know the node before it. To do this, each of the nodes is given a number that relates to the position in which they were placed. By doing so it will make it easier should the user decide they want to adjust the path lines or locations. It will also allow for traversing backwards should this be desired, it could also allow for potential parallel processing.

2.0 - Overview of Path Finding Algorithms

As the application will need to recommend a route given a set of nodes, a variety of path finding algorithms can be used. Path finding algorithms are widely used in games, mapping software and Video Games (AI Blog, 2008). Before attempting to solve the problem, we looked at several algorithms:

2.1 - Floyd- Warshall algorithm

This graph analysis algorithm is used for finding the shortest paths in a weighted graph. It compares all possible paths between each pair of vertices and with a single execution, the lengths of the shortest path between all pairs of vertices can be found. The Floyd-Warshall algorithm examples dynamic programming. ("Floyd-Warshall's Algorithm", 2008).

2.2 - Dijkstra's algorithm

The Dijkstra's algorithm is often used in routing and as a subroutine in other graph algorithms. This algorithm is used for finding costs of shortest paths from a single vertex to a single destination vertex. It is done by stopping the algorithm once the shortest path to the destination vertex has been determined. Dijkstra's algorithm solves single-source shortest path problems for a graph with nonnegative edge path costs and produces a shortest path tree. It also assigns some initial distance values and tries to improve them step by step. ("Data Structures and Algorithms", 1998).

2.3 - Bellman-ford algorithm

The Bellman-ford algorithm is primarily used for graphs with negative edge weights. Each node calculates the distances between itself and all other nodes within the as and stores this information as a table. Each node then sends its table to all neighboring nodes. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes. It tends not to scale well and changes in network topology are not reflected quickly since updates are spread node-by-node. ("One Source Shortest Path: The Bellman-Ford Algorithm", 2007)

2.4 - A* algorithm

This algorithm is widely used in pathfinding and graph traversal. Known for using a best-first search, it finds a least-cost path from a given initial node to one goal node. The A* algorithm follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way. It uses a distance-plus-cost heuristic function of node, xf(x), to determine the order in which the search visits nodes in the tree. (“Introduction to A*, 2013”)

2.5 - Critical Path Analysis

The Critical Path Analysis is a project management tool that sets out all the individual activities that make up a larger project. The order of which tasks have been undertaken is shown via this tool. It also shows tasks that can either be done only directly after a task or simultaneously with another task therefore reducing the completion time of a project.

A Critical Path Analysis must start and end on a node. Each of these node are numbered for identification, to show the earliest start time and to show the latest finish time. It solely depends on the accuracy of the information used in order to work efficiently. Although the critical path analysis is used as a project management tool, it uses nodes and paths and thus could be used within our solution.

2.0 - Technology Considerations

The use of technology was another issue that needed to be addressed. As the task did not require a specific language, a variety of different technologies were available. Although the application is mainly algorithm based, we needed an user interface which provides enough functionality for the user to carry out all the required tasks.

Different options that were available included Raphael.js (Raphael.js), HTML5 Canvas and d3.js (d3.js, 2012). After assessing the different technologies available, the group chose to utilise Google Maps Javascript API V3 (Google, 2012) . Google Maps API provided the base for custom markers, paths and interface. It also allowed the user to easily navigate the map, and provided a feature for custom maps which could later be used.

Google Map can also be as a background (for providing the correct maps) then overlaying a HTML canvas to provide he visualisations for the nodes and paths.

The difference between using Google Maps with a canvas overlay and the Google Maps

API is that within the API most of the marker, lines and map functionality was provided and a detailed documentation with examples is available online. It was then decided that the Google Maps API was a more appropriate method and would complement the project well.

By using Google Maps API it would permit the plotting of nodes to be drawn directly onto a map and then using the features of Google Maps to work out distances and other details that are required for an appropriate solution.

Javascript was the chosen language to use throughout the project. All members of the group have experience in javascript and would be an ideal candidate for future expansion, distribution and the ability to run server side if required. It can also be ran in any internet browser. Google Chrome (Google Chrome, 2012) will be used as the chosen browser as it provides many debugging and developer's tools which will make developing javascript easier.

The interface for the entire project will come from Twitter's Bootstrap User interface framework (Twitter, 2012). Providing all the individual features needed for this project, it has been chosen for its easy use. With its cross browser compatible professional finish, it is an effective way to create usable interfaces, with its readily available elements awaiting to be implemented.

3.0 - Development of project

We took a group decision to introduce the algorithms at a later stage. The reasoning behind this was to decrease complexity during implementation. As it would be easier to add the algorithm to the solution instead of building the solution around the algorithm, this seemed like the best and most suitable approach for us.

3.1 - Development of chosen algorithm

After weighing up and researching the various algorithms, we decided to use a critical path analysis. Although less efficient, it would be easier to implement and provide the desired results. After the framework has been completed, it will be easier to expand the choice of algorithms and add in more as required.

At the start of the implementation the problem was tackled at a very basic level, this was to ensure that the problem and all requirements were completed. It became

apparent that our solution would work best with only a small number of nodes to work with.

3.2 - Developing interface

With the aid of Twitter Bootstrap and Google maps, the interface was able to be developed. We took advantage of the markers, the lines and most importantly the map interface that Google Maps API had to provide. The map interface gave us the ability to zoom in and out as well as panning, increasing the users usability. Later, we then teamed this up with the buttons produced by Bootstrap utilising the layout and form styling Bootstrap also had to offer. jQuery, a multi-browser JavaScript library was also made use of. We used it for the button clicking and launching events.

4.0 - Overview of finished solution

Within the solution, each path (between two nodes) can be assigned any number of attributes. These attributes can refer to any physical property of that route, and each one is assigned a numeric value. An example of this is: for each path we can set the increase in height, distance, number of trees and a danger score.

For each of the attributes, a function known as a mediator is executed. A mediator is a small function that can be added easily, by editing a single javascript file. This function is executed as the journey is calculated and affects the attributes which have been calculated.

Within each mediator function, the user can easily set up logic such as “If the current distance traveled is over 100, and the water level for this path is over 10, then add 10 minutes to travel time”, or “if the risk of heavy enemy artillery is high, a percentage of tanks may be lost on the path”.

The solution uses a critical path analysis technique (often used in project management). Every unique journey (ie a path between the start node and end node) are calculated before the best route is chosen. However, this method is not very efficient as routes are calculated which are not candidate best/worse routes.

Currently, the solution outputs the highest and lowest route for each of the hazards. This allows the user to then pick the route. Ideally, we would like to develop this feature further and add in the ability to pick the criteria for the best route. The critical path analysis allows this to be easily implemented.

Rather than use a graphics library such as Raphael or drawing onto a canvas directly, we have used Google Maps API. This provides some of the user interface such as the nodes, paths and background mapping. The google map can be located anywhere and can also have an image as the background, therefore maps of anything can be used.

Most of the User Interface and User Experience (UI/UX) uses the popular Twitter Bootstrap front-end library. This library makes it easy to create custom interfaces using a minimal amount of code.This provided a simple yet effective way of adding the features, and created a seamless and user friendly interface.

We also used two javascript libraries, underscore.js (http://underscorejs.org/) and jQuery (http://jquery.com/). jQuery is mainly used for events such as “onclick”, and underscore.js used for looping, array functions and templating.

Twitter buttons with jQuery enabled the user to connect the nodes together, add/edit/delete attributes and start the calculation of the route finding.

4.1 - Future features which could be added

When the most part of the solution was complete, addition features for the user were required.

One method to make the solution more complex was the ability to add and remove nodes and the pathways between nodes. This function was added to ensure that if any human error occurred, it could be fixed quickly and easily without any hassle. Not only that but if the user changed their mind or addition information was provided half way through plotting the path then it could be adjusted easily.

Having travel functions added a new level of development to the project, they work by traversing the whole journey and calculate attributes as it goes along the path.

As we used JavaScript to write our application, we could potentially distribute the computation across multiple machines/threads. The web app can export the nodes and routes as a JSON string, then send to other machines using technology such as Node.JS, Pusher or Web Sockets.

Although the current solution only uses a simple critical path finding algorithm, the framework can also be expanded to use different algorithms such as A* or Dijkstra’s.

The solution assumes that everything is traveling together and will only find one route. To improve the algorithm, there could potentially be different number of routes for different vehicles. For example, soldiers on foot may travel one way, and large vehicles can travel another.

Currently, there are no limitations within the paths. Every path is accessible. Ideally, each path could have limitations. For example, a path can only be used if there are 10 soldiers, and have to split.

The output of the algorithm shows the highest and lowest route for each unique hazard. This can be improved by having a better reporting technique. A route could therefore be the "best" if it has the highest amount of water but the lowest amount of danger.

5.0 - Conclusion/Overview

When breaking down the project, the interface, node work and travel/hazard functions were all coded individually then combined. By doing the work in this manner is allowed each element to be coded, tested and checked before adding it to the main body of the project. This was more time effective as it did not result in several errors occurring for multiple different elements.

By splitting the coding this way it meant that the group work was split evenly and each person could work on their own element without disturbing other members. Once all the elements were done they could be put together and a final testing could be completed. By dedicating the majority of the given time on individual areas and their performance allowed each area to be fully completed and refined before moving on to the next element, this made the project more manageable for all involved.

By using a critical path analysis, it allowed us to simplify the task and create a basic framework which can be expanded to make the solution more complex. Once the basic requirements were met and the group had a better understanding of how to create the elements effectively.

6.0 - References

PaulT. (2008). Ai Blog - Retrieved July 26, 2008, from http://www.ai-blog.net/archives/000152.html

Baranovskiy, D. (2012). Raphael.js - Retrieved from http://raphaeljs.com/

Bostock, M. (2012). D3.js - Retrieved from http://d3js.org/

Google Maps Javascript API V3 (2012) Retrieved from the Google developers website: https://developers.google.com/maps/documentation/javascript/tutorial

Google Chrome Developer Tools (2012) Retrieved from the Google developers website: https://developers.google.com/chrome-developer-tools/

Twitter Bootstrap (2012). Retrieved from http://twitter.github.com/bootstrap/

Floyd-Warshall's Algorithm (2008, November 25). Retrieved from the Algorithmist website: http://www.algorithmist.com/index.php/Floyd-Warshall's_Algorithm

Morris, J (1998), *Data Structures and Algorithms*. Retrieved from http://www.cs.auckland.ac.nz/~jmor159/PLDS210/dijkstra.html

One Source Shortest Path: The Bellman-Ford Algorithm (2007. November 29) Retrieved from http://compprog.wordpress.com/2007/11/29/one-source-shortest-path-the-bellman-ford-algorithm

Patel, A., (2013) *Introduction to A**. Retrieved from http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html