**Sailing Yacht Optimal Route Calculator:**

Contents

[Analysis: 5](#_Toc57466862)

[Background Information: 5](#_Toc57466863)

[Description of the Current System: 5](#_Toc57466864)

[End User: 6](#_Toc57466865)

[Evidence of Analysis: 6](#_Toc57466866)

[Websites: 6](#_Toc57466867)

[Prototype Sketch/End-user Feedback: 7](#_Toc57466868)

[End-User Interview: 7](#_Toc57466869)

[Modelling of the Problem: 8](#_Toc57466870)

[Tacking: 8](#_Toc57466871)

[Pathfinding: 10](#_Toc57466872)

[Acceptable Limitations: 14](#_Toc57466873)

[Research Log: 15](#_Toc57466874)

[Data Volumes: 16](#_Toc57466875)

[User Requirements: 16](#_Toc57466876)

[Objectives: 16](#_Toc57466877)

[Documented Design 19](#_Toc57466878)

[Overview: 19](#_Toc57466879)

[IPSO Chart: 19](#_Toc57466880)

[Class Structure: 21](#_Toc57466881)

[User Interface: 21](#_Toc57466882)

[Information: 21](#_Toc57466883)

[Requirements: 22](#_Toc57466884)

[Implementation: 22](#_Toc57466885)

[Object Oriented: 24](#_Toc57466886)

[Data Structures: 24](#_Toc57466887)

[API Management: 25](#_Toc57466888)

[Information: 25](#_Toc57466889)

[Implementation: 25](#_Toc57466890)

[Algorithms: 27](#_Toc57466891)

[A\* Pathfinding: 27](#_Toc57466892)

[Tacking Route Algorithm: 30](#_Toc57466893)

[Waypoint Manipulation Algorithm: 33](#_Toc57466894)

[Tidal Stream Algorithm: 34](#_Toc57466895)

[Map Generation Algorithm: 35](#_Toc57466896)

[*Overview*: 35](#_Toc57466897)

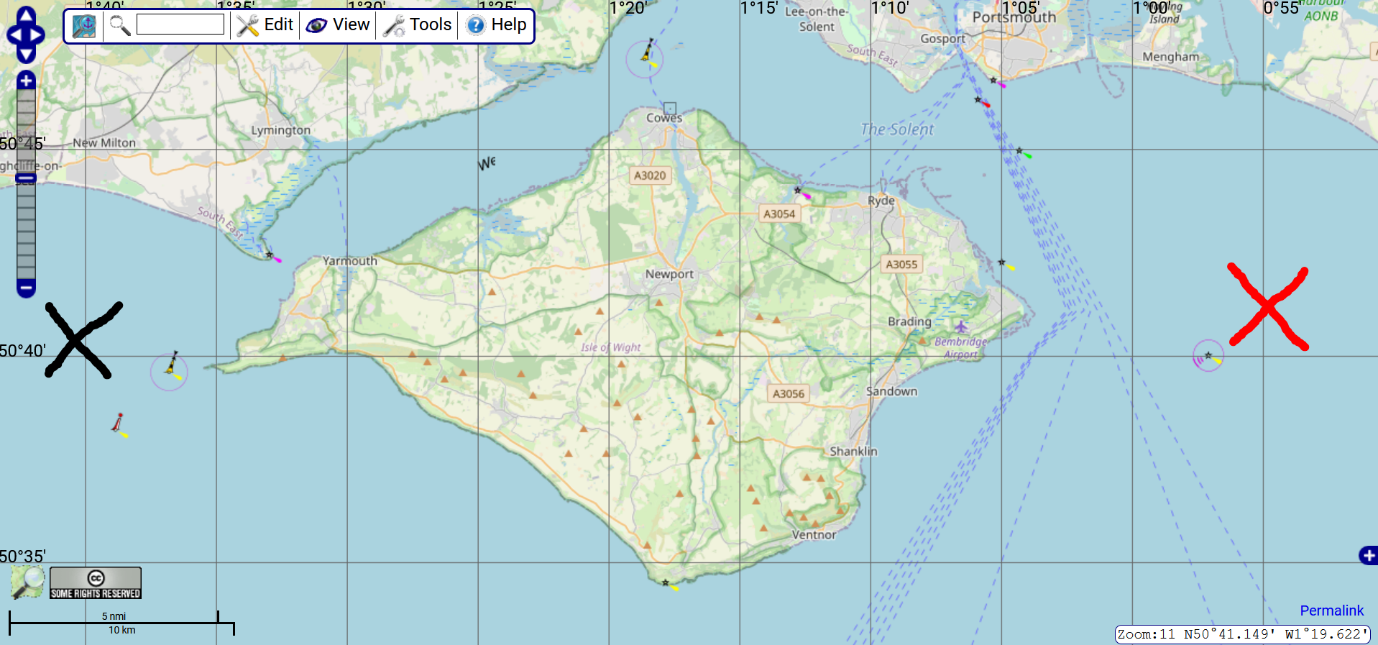
[Testing 35](#_Toc57466898)

# Analysis:

## Background Information:

At weekends I enjoy sailing with my Dad, during this time I noticed that most of the problems we encounter directly resulted from a mistake made whilst planning the route. For example, on multiple occasions we have had to sail at 5knots straight into a 4knot tidal stream which can take hours to travel through, this wouldn’t have occurred if we accounted for the tidal stream and left at a different time or taken a different route. After being stuck in these situations many times I realized that if the route-planning process was automated these mistakes could be mitigated. There are usually hundreds of boats sailing in the isle of weight area, each one has to meticulously plan their journey to find the shortest/safest route, eventually one will make a mistake that could lead to damage to the boat or endangering the crew. My system aims to combat these issues by automatically plotting the fastest route between two user-set waypoints, considering live wind and tidal data furthermore the user should be able to load previously plotted routes.

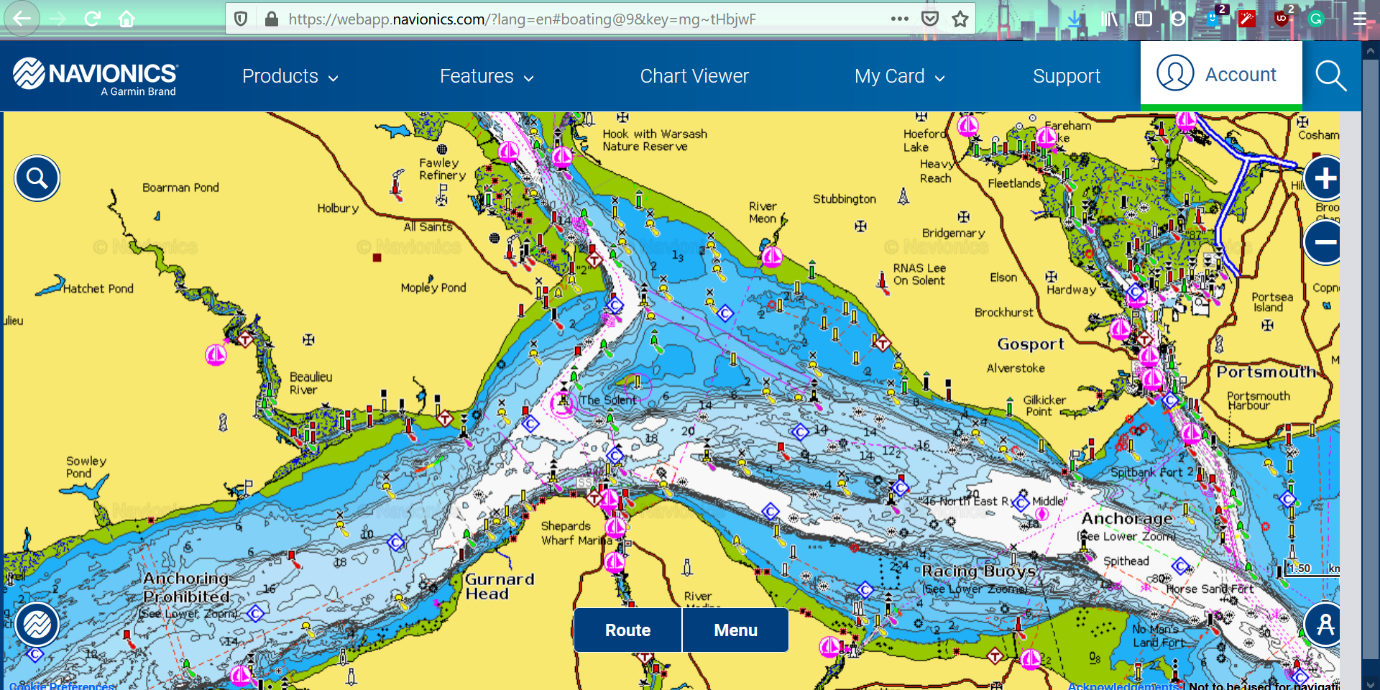
## Description of the Current System:

On most boats the route is determined by the crew before setting sail, usually on paper maps. They will look at the forecasts for wind direction and speed, as well as tidal streams and heights. At this point a general route is ascertained which is normally a straight line from A to B, however this might not always be the fastest route as it’s hard for humans to predict the specific weather conditions far into the future. For example in this screenshot from <https://map.openseamap.org/> it is not initially clear which route to take between the red and black crosses. Since this map doesn’t display tidal streams or wind data. Which could lead to one route being significantly longer.

Another problem with paper maps is that it’s generally considered bad practice to write on maps meaning the crew must memorize the route. There are existing digital systems such as Navionics (<https://www.navionics.com/gbr/>) however solutions like this will provide a chart displaying navigational data but most won’t calculate an optimal route. The following screenshot is from the Navionics website, this map contains a lot of more navigational data such as water depth, however I believe that there are simply far too many variables for people to account for when plotting their own routes.

Main Flaws of the current system include:

* Plotting an accurate route manually can be time consuming depending on its length.
* A manually plotted route is prone to errors since there are too many variables for people to account for. Which can lead to being stuck down-wind/down-tide.
* The fastest route is not always initially obvious.



## End User:

The end user will be Keith Gough – my Dad because he has sailed for a long time, he has also used Navionics in the past, so he knows the issues with commonly used navigational software. The prospective end-user is generally people who sail in the UK.

## Evidence of Analysis:

### Websites:

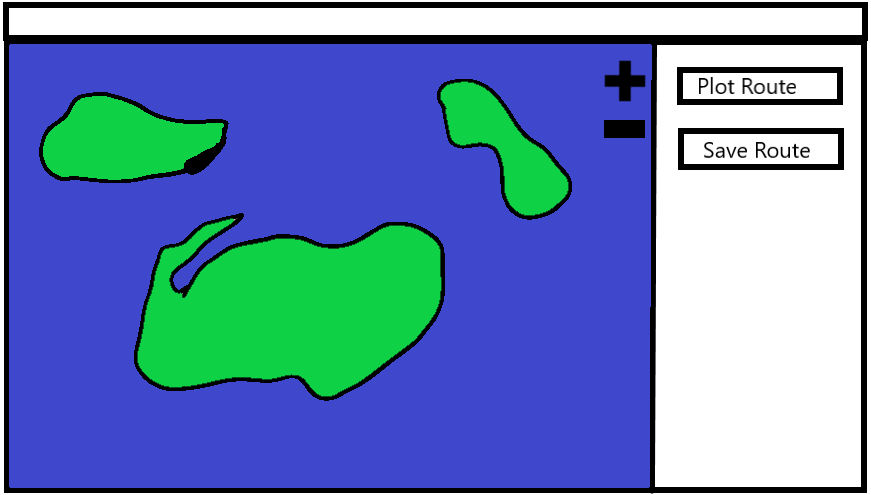
* <https://newt.phys.unsw.edu.au/~jw/sailing.html>

<https://en.wikipedia.org/wiki/Tacking_(sailing)>

These websites explain the basic physics of sailing including the upwind tacking process – This process could be implemented with a recursive algorithm

* <https://www.researchgate.net/publication/282488307_Pathfinding_Algorithm_Efficiency_Analysis_in_2D_Grid> This document explains the most optimal pathfinding algorithm for different situations – Best candidate so far is A\* since it is optimal on large grids – Note that any pathfinding algorithm will require some way of detecting where the land is.
* <https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client> This website explains how to call an API in C#, this will be necessary for accessing live tidal and wind data – Note this process should be asynchronous as response times can be slow.

### Prototype Sketch/End-user Feedback:



End-User Feedback:

* Good simple design
* Requires an input box for the closest angle the boat can sail towards the wind, since that depends on the boat
* The user should be able to place and delete waypoints with a mouse click
* There could be an arrow to show the wind direction and speed

### End-User Interview:

To get an idea of how the system should be structured in order to suit the average sailor I conducted an interview with my Dad (the end user). The following list shows my questions and his summarised responses

Q: “What are the current issues with manual route-plotting on paper maps?”

* My Dad made it clear that whilst plotting his routes on paper maps he often has to go back over and recheck certain parts of the route to make sure they would work with the tidal and wind conditions, he also said that he often finds that he finds himself repeating the same plotting process that he has already completed several times before when planning a previously completed route.
* At this point I realised my system would need to be able to save routes so that the user wouldn’t have to plot the same route twice.
* My system would need to take each of these variables into account when plotting the final route meaning I would need to use an API to gather the live data.

Q: “What features would you like to see in an automated system?”

* My Dad said that it would be useful to be able to place waypoints on a digital map so that at each stage of the journey he would know the exact bearing he’d need to sail on. He also said that it would be nice if the program would be able to put in tacks so he would know where to turn the boat to avoid obstacles. Finally, he said that the program should be intuitive and easy to use since many people he knows who sail are often reluctant to use technology due to overcrowded UIs.
* At this point I decided that my UI would need to be minimal so that it didn’t confuse people, but it still needed to display the relevant information. I was already planning to use waypoints and plot the individual tacks between those waypoints.

Q: “What existing software have you used and what are the good/bad aspects of these?”

* He said that he has used the premium version of Navionics for the past few sailing seasons, he liked the tidal gauges in the app that showed the tide height and direction forward in time. He also liked that its GPS so it could tell the current position and velocity of the user. He disliked the mobile version however because it was too small to see most of the information. My Dad also disliked the fact that in the app you can’t plot the route by adding and removing waypoints, all it does is show navigational information. He also stressed that some Apps will use different units of measurement to the onboard instruments which isn’t particularly helpful
* My system will use the standardised nautical measurements since that’s what most modern instruments use (Nautical Miles, Knots)
* My system will need to be able to also plot a route using shortest path algorithms to avoid land and other obstacles, however it might not be feasible for me to incorporate GPS functionality within the scope of the project due to time constraints, this could be a feature on later versions though.

Q: “What sort of UI features would normally deter you from using a piece of software?”

* He explained that a cluttered and busy UI was often off-putting but on the other hand the UI needs to be full enough that the user could fully understand the program. Ultimately, he said the UI should display the map and some controls to allow the user to navigate around the map.
* In hindsight my Dad is probably biased in regards to the more ‘technical questions’ such as this since he is in the computer science industry but this did cement the idea that my UI would need to minimalist in order for most people to not be scared away from the program.

Q: “Will the system require a user input for the angle-to-wind and hull speed? Or should it use average values”

* My Dad said that Apps such as Navionics don’t take these inputs because they don’t allow the user to plot any sort of route so they aren’t necessary but that for my system to tailor to its user’s specific boat these would be required.
* After this question I learnt that I would need to take in user inputs for several values which means I will need to have some input validation and exception handling, I realised I will also need that in order for the user to save routes as they would require a file name input.

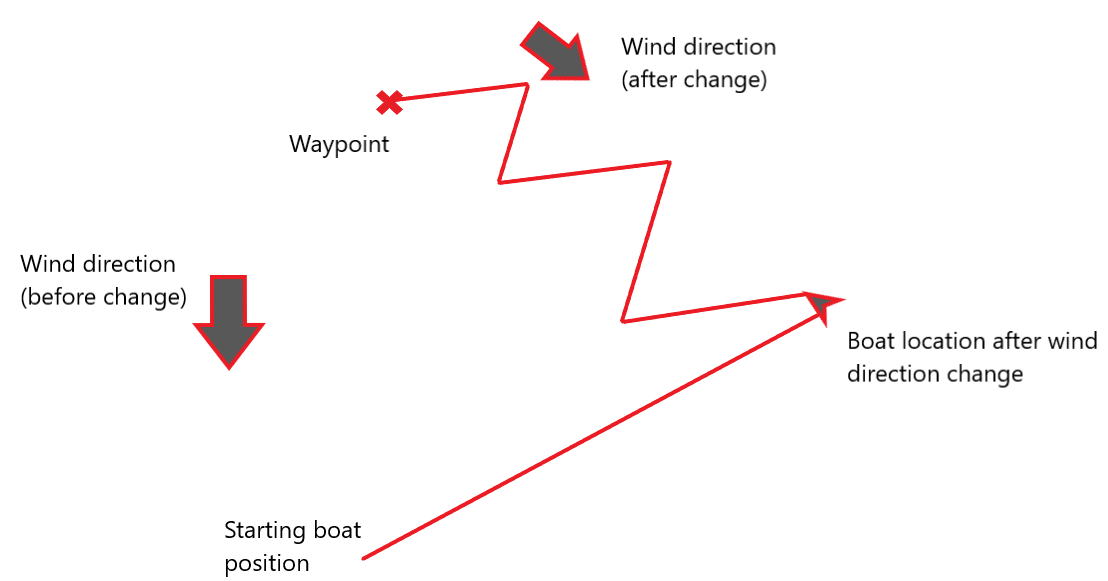
## Modelling of the Problem:

### Tacking:

The system will need to use tacking (zig zag) to reach the waypoint. This means that the system will have to decide at what point to turn through 90 degrees across the wind bearing. Normally this is done by the sailor and they will usually guess the angle but its good practice to never spend too long on a single tack this ensures that if the wind changes direction the boat is never too far down wind. The examples below show this. Taking shorter tacks will result in the route much more resilient to any change in the wind conditions. I made the following diagrams in MS paint 3D, the red line represents the route the boat will have to take, the arrow represents the boat’s position.

Example 1 (Long Tacks):

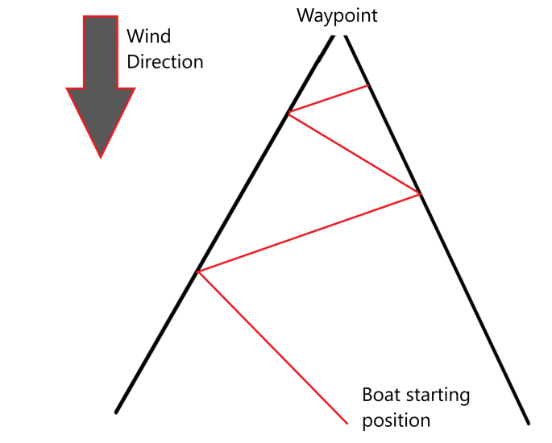
* This diagram shows how if the wind changes after making one long tack this leaves the boat with a much longer route to take to reach the waypoint.



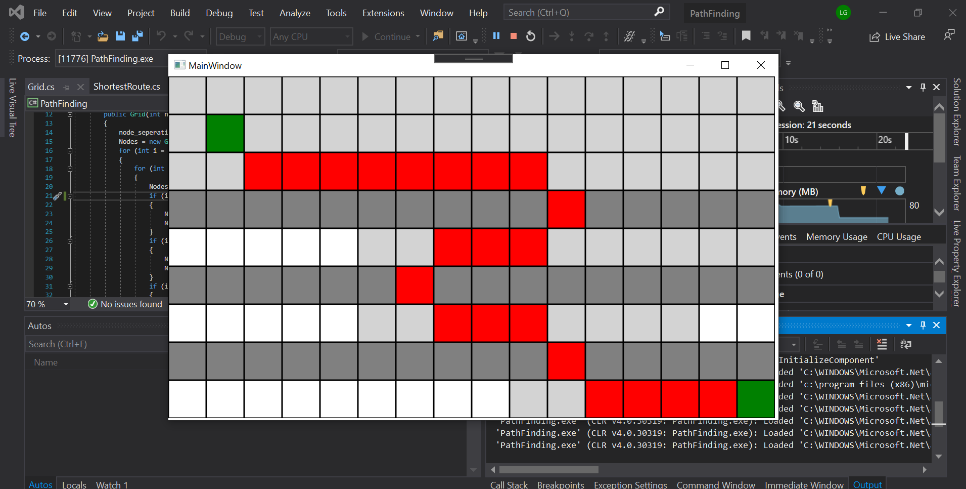
Example 2 (Short Tacks):

* This diagram shows that taking shorter tacks means that if the wind changes the boat will have to tack a much shorter route towards the waypoint



The system will need an algorithm to determine the right point to turn at and start a new tack. A general rule that I have learnt from experience is to start a new tack once the bearing to the boat’s position from waypoint becomes 10 degrees greater or 10 degrees less than the bearing to the boat’s initial position from the waypoint. This allows me to construct a 20-degree cone (shown on the left) from the waypoint, at the points where the boat intersects this cone it can start a new tack.

### Pathfinding:

Below is a small test program for the A\* algorithm that the full solution will implement, the green blocks represent a start and end point, with the dark grey blocks as obstacles.

Programming the prototype enabled me to establish a good understanding of programming a UI in C# using WPF (windows presentation foundation). I decided to use this instead of windows forms because it allowed me to design my graphical interface in a coding environment (written in XAML) rather than a graphical designer. From looking online at tutorials such as <https://youtu.be/Vjldip84CXQ> I learnt all the basic UI elements that I would need to use in the main program:

* Creating UI Elements statically
* Drawing shapes dynamically
* Drawing Text blocks dynamically
* Taking text input from a user
* Buttons
* Opening new windows dynamically

This prototype also raised an issue that I would encounter when programming the full solution, the pathfinding algorithm was becoming a lot slower as I added more nodes for it to plot a path through. To overcome this issue I had to learn about multithreading in C#, <https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming>, this also proved useful later on when I needed to call an API, since response times can be long. I also realized that for the main program I would need to establish a grid of nodes for the algorithm to plot a route through, which would require a method for retrieving whether a pixel contains land or not.

I decided to use A\* for this project because after some research I learnt that this algorithm is a best-first search algorithm means that it requires information about the end location, which in this case will just be its coordinates, unlike other pathfinding algorithms such as Dijkstra’s, it will actively work towards the end point meaning it should have to check fewer nodes and therefore it will take less CPU cycles to complete. A\* combines the information that Dijkstra’s uses, favouring vertices that are close to the starting point, and the information that Greedy Best-First search algorithms use, favouring vertices that are closer to the end point. The code below shows my implementation of the A\* algorithm in the prototype, it takes a start and finish point parameter and should store the list of nodes to traverse.

using System;

using System.Collections.Generic;

using System.Windows.Media;

using System.Windows.Threading;

namespace PathFinding

{

class ShortestRoute

{

private List<GridSquare> Route\_coords = new List<GridSquare>();

private readonly GridSquare start\_square;

private readonly GridSquare end\_square;

private readonly double step;

public ShortestRoute(int[] linePos, double Step, Grid grid)

{

start\_square = grid.GetGrid[linePos[0], linePos[1]];

start\_square.SetFill(Brushes.Green);

end\_square = grid.GetGrid[linePos[2], linePos[3]];

step = Step;

GenerateRoute(grid);

}

public void GenerateRoute(Grid grid)

{

// Node: 0 = X, 1 = Y, 2 = F\_cost, 3 = G\_cost, 4 = index of parent

List<GridSquare> closed\_nodes = new List<GridSquare>();

List<GridSquare> open\_nodes = new List<GridSquare>() { start\_square };

ChooseNodes(open\_nodes, closed\_nodes, grid);

// Follows the chain of parent indexes backwards from the last node to the start node to get the route

int node\_index = closed\_nodes.Count - 1;

while (node\_index != 0)

{

Route\_coords.Add(closed\_nodes[node\_index]);

node\_index = closed\_nodes[node\_index].Parent;

}

Route\_coords.Reverse();

}

public void ChooseNodes(List<GridSquare> openNodes, List<GridSquare> closedNodes, Grid grid)

{

while (openNodes.Count > 0)

{

GridSquare current\_node = PickCurrentNode(openNodes, closedNodes);

current\_node.SetFill(Brushes.LightGray);

// If curren\_node != end position then calculate the neighbors

if (current\_node != end\_square)

{

CheckNeighbor(1, 0, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(0, 1, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(-1, 0, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(0, -1, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(1, 1, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(-1, 1, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(1, -1, current\_node, openNodes, closedNodes, grid);

CheckNeighbor(-1, -1, current\_node, openNodes, closedNodes, grid);

}

else

{

break;

}

}

}

public void CheckNeighbor(int x, int y, GridSquare currentNode, List<GridSquare> openNodes, List<GridSquare> closedNodes, Grid grid)

{

int x\_pos = currentNode.GetLeft + x;

int y\_pos = currentNode.GetTop + y;

if ((x\_pos >= 0 && x\_pos < 800 / step) && (y\_pos >= 0 && y\_pos < 450 / step))

{

GridSquare neighbor = grid.GetGrid[y\_pos, x\_pos];

// If neighbor is land or is already closed then return

if (GetNode(closedNodes, neighbor) != null || neighbor.Obstacle)

{

return;

}

else

{

GridSquare node = GetNode(openNodes, neighbor);

double H\_cost = Math.Abs(neighbor.GetLeft - end\_square.GetLeft) + Math.Abs(neighbor.GetTop - end\_square.GetTop);

double G\_cost = currentNode.G\_cost + NodeDistance(x, y);

// If node already exists

if (node != null)

{

if (G\_cost + H\_cost <= node.G\_cost)

{

// If new path to neighbor is shorter then set the shorter path node as parent

node.G\_cost = G\_cost;

node.F\_cost = H\_cost + G\_cost;

node.Parent = neighbor.Parent;

}

}

else

{

neighbor.Parent = closedNodes.IndexOf(currentNode);

neighbor.G\_cost = G\_cost;

neighbor.F\_cost = H\_cost + G\_cost;

openNodes.Add(neighbor);

}

}

}

}

public double NodeDistance(double x, double y)

{

// If its a diagonal distance = 1.4 if not then distance = 1

if (Math.Abs(x) == Math.Abs(y))

{

return 1.4;

}

else

{

return 1;

}

}

public GridSquare GetNode(List<GridSquare> list, GridSquare item)

{

// Checks if the node positions are the same

foreach (GridSquare node in list)

{

if (node == item)

{

return node;

}

}

return null;

}

public GridSquare PickCurrentNode(List<GridSquare> openNodes, List<GridSquare> closedNodes)

{

double max\_F\_cost = 10000;

int node\_index = 0;

// Looks for the node with the lowest F\_cost

foreach (GridSquare node in openNodes)

{

if (node.F\_cost < max\_F\_cost)

{

node\_index = openNodes.IndexOf(node);

max\_F\_cost = node.F\_cost;

}

}

GridSquare currentNode = openNodes[node\_index];

openNodes.RemoveAt(node\_index);

closedNodes.Add(currentNode);

return currentNode;

}

public double Hypotenuse(double num1, double num2)

{

return Math.Sqrt(num1 \* num1 + num2 \* num2);

}

public List<GridSquare> GetRouteCoords

{

get { return Route\_coords; }

}

}

}

## Acceptable Limitations:

* The system will not be able to get the current wind data as often as existing systems because I can’t call my API more than 1000 times a day without paying for more requests.
* The system will require a network connection to get accurate data however it can still run and plot routes without a network connection.
* My system will disregard neap tides since mathematically this will become a lot more complicated – The tidal range across the globe changes by a small amount periodically however it makes the formulas significantly more complicated so I agreed with the end user that neap tides weren’t required.

## Research Log:

|  |  |
| --- | --- |
| Date | Research Carried Out |
| August 2020 | * Read about different ways to create a UI in C#, I decided to use WPF since it gives me much more control of the UI Elements than Windows Forms * Read about different sailing techniques <https://en.wikipedia.org/wiki/Sailing> * Read about pathfinding and the benefits and drawbacks of different algorithms, I concluded that A\* would work best because it knows the location of the final node and therefore can work towards that node. |
| September 2020 | * Read about calling APIs in C#, which lead on to reading about asynchronous programming in C#. This will be required for gathering live wind and tide data. * <https://www.c-sharpcorner.com/article/calling-web-api-using-httpclient/> * Read about tidal flow in the UK – A full tide cycle will occur every 12hrs 25 mins. * <https://www.kayarchy.co.uk/html/03thesea/006currents.htm> * Wrote the first prototype displaying the finding algorithm and how it finds the optimal route through a maze. This should be easy to transfer over into the main program * Started programming the main program, first I wrote the XAML code which defines the initial state of the UI, at this point I was able to decompose the problem into manageable subroutines each time I finish one of these the program should be functional. * Completed the first functional chunk of code which allows the user to place as many waypoints as they like in series and links them together. * Completed the second functional chunk of code which will plot the tacking route between all the nodes placed by the user. At this point I had to also call the wind API. * Completed the third functional chunk which will plot the fastest route between the waypoints. * Researched about multi-threading and asynchronous programming in C# <https://docs.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading> |
| October 2020 | * Completed the fourth functional chunk of code which will allow the system to display the route length and the route completion time after completing the tacking route plotting algorithm. * Researched writing and reading to a text file using a stream reader/writer in C# * <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-write-to-a-text-file> * Completed the fifth functional chunk of code which allows the user to save waypoint locations to a text file, during this stage I also went back and reworked the tacking route plotter and the land detection system so that it would save the land locations in a text file. |

## Data Volumes:

* Text File – Waypoint Coordinates in order
* Text File – Map Land/Sea text file

## User Requirements:

1. User interface/Console to enter the required data such as hull speed (Maximum boat speed) and the draught (How deep the boat is)
2. The program will then collect API data on wind and tidal streams for a pre-set coastline location
3. The program will then draw the coastline location with simple vector graphics
4. The user will then place waypoint markers onto the map, to do this the program will have to take a mouse button input at a location on the map
5. Next it will calculate the most efficient route based off the parameters from each waypoint to the next.
6. In order to do this the boat can be travelling in two states, tacking, meaning the waypoint is upwind from the boat and therefore the boat cannot point directly at the waypoint and has to move in a zigzag pattern; or running, meaning the point can point at the next waypoint and therefore it can take the fastest line.
7. Based on the boat’s state, the wind/tidal data can then be applied to the route to calculate a realistic track.
8. The final simulation of the boat completing the route will then be run to display the final route to the user. The user can then follow this route to reach the destination in the fastest time.

## Objectives:

Main Objective:

The program must generate an efficient route between two user-placed routes, the route must consider the wind and tide to generate a fast route.

Specific Objectives:

Inputs:

1. The system must allow the user to place a waypoint at the mouse location by clicking the left mouse button on a location on the map.
2. The system must not allow the user to place a waypoint on land
3. The system must allow the user to delete waypoints at the mouse location by right clicking on a waypoint.
4. The system must allow the user to place a tide point at the mouse location by shift-left clicking at a land location
5. The system must allow the user to delete a tide point at the mouse location by shift-right clicking on the tide point
6. The system must allow the user to enter values for maximum and minimum tide velocity at a tide point
7. The system must allow the user to enter a value for the tidal stream bearing at the tide point
8. The system makes sure the user can’t enter a negative value for maximum and minimum tide velocity
9. The system must make sure the user can enter a value greater than 360 or less than 0 for the tidal bearing
10. The system must allow the user to input a specific angle for how close the boat can sail towards the wind
11. The system must allow the user to input a maximum velocity for their boat so that it can calculate an approximate route completion time.
12. The system must allow the user to save the current waypoint locations to a text file by clicking a button and entering a save file name.
13. The system must be able to load waypoint locations from a text file by allowing the user to input an existing save file name.
14. The system must make sure the user cannot enter any invalid file name for files on windows
15. The system must have a button which will run the fastest route plot algorithm
16. The system must have a button which will run the tacking algorithm
17. The system must make sure that the entered by the user are valid for windows
18. The system must make sure the angle to wind is less than 45 degrees and greater than or equal to 0 degrees
19. The system must make sure the maximum velocity input is not negative
20. The system must not allow the user to place a waypoint or tide point outside the map window

Outputs:

1. The system must display a section of map.
2. The system must display a waypoint (red circle) after the user places one.
3. The system must display a line connecting consecutively placed waypoints with a black line.
4. If a waypoint is deleted the system must also delete any lines connecting to it and link the previous and subsequent waypoints with a new line.
5. The system must display a new save/load window when the save/load button is clicked
   1. The save load window must have taken an input of a filename
   2. The save load window must have a text box explaining to the user that the waypoints will be saved.
   3. The save load window must have a button for both saving and loading
      1. When the save button is pressed the system must check for existing files with the filename inputted
      2. If there is no existing file a new file will be created and the location of the individual waypoints will be saved (x, y) line by line.
      3. When the load button is pressed the system must check the file name entered exists, generate new waypoints with the locations listed in the text file.
6. If the fastest route plot button is pressed the system must automatically place new waypoints defined by the fastest route algorithm.
7. If the tacking route plot button is pressed the system must place red lines displaying the new route defined by the tacking algorithm
8. The system must display the new tide point window when the user places a new tide point
9. The system must display the cos wave described by the user’s inputs in the new tide point window
10. The help window must open when the help button is pressed

Processing:

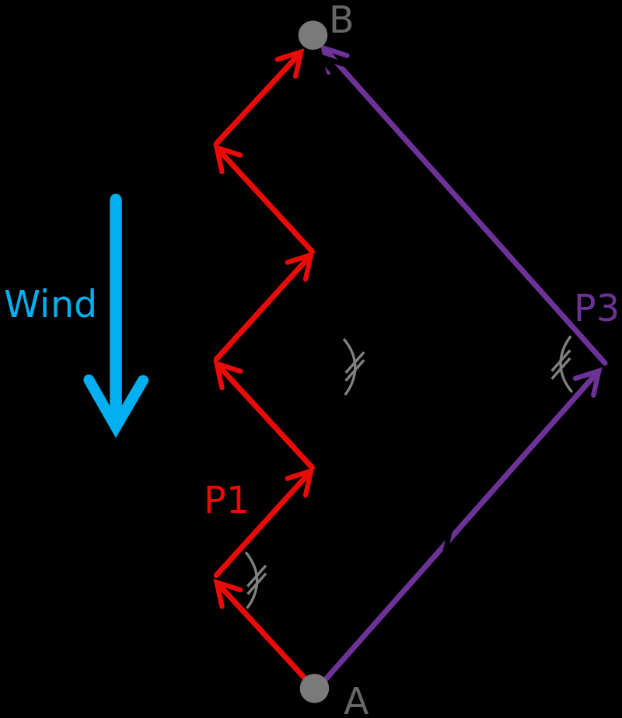
1. Fastest Route Algorithm
   1. The system must run the A\* pathfinding algorithm between the user defined waypoints, this process can be asynchronous because it does require access to the UI.
   2. A\* pathfinding will place nodes which are a predefined distance apart, these nodes will be too close together to be their own waypoints, which means the unnecessary nodes must be deleted.
      1. The system must check for nodes which have line of sight on nodes which are ahead of the selected node in the chain, these nodes are deemed as unnecessary and the system will delete them
      2. This will ensure that nodes are only placed at the points where the boat **must** make a turn to avoid land
   3. The system must then place new waypoints at the positions of the remaining nodes in the chain.
2. Tacking Route Algorithm
   1. Between each waypoint the system must first check if the boat can sail straight towards the waypoint meaning it won’t have to plot a tacking route
   2. If it can sail towards the waypoint the system must place a red line between the two waypoints
   3. If it can’t sail towards the waypoint then it will sail at a default angle of 40° either side of the wind
      1. The system will place the red route line from the start position to the point where the 40° tacking line intersects the edge of a cone 10° either side of the waypoint.
      2. This ensures the boat will never be too far downwind in case the wind changes direction, meaning the route can’t be greatly affected by a change in wind direction.
      3. Once the line is placed, the boat will change tack meaning it will plot the next route line following the opposite 40° tacking angle, this process must be repeated until it can sail directly towards the waypoint
      4. This ensures the boat will ‘zig zag’ towards the waypoint
   4. This process must be repeated for each waypoint, meaning the system will plot a route between each waypoint.
3. The system must save the waypoint locations to a text file
   1. First the system must check if the filename already exists on the system
   2. Next the system must create a new text file which will be saved in the “SaveFiles” folder in the solution folder
   3. The waypoint locations will be written to the file one waypoint per line.
4. The system must load the waypoint locations from a text file
   1. First the system must check if the filename exists
   2. Next the system must read the file line by line, creating a new waypoint for location stored in the text file, linking them consecutively with a black line.
5. Route Metrics
   1. The system must calculate the length of the route in nautical miles, using the DPI of the monitor to calculate the distance (defined by the original scale on the map) per DIP (device independent pixel).
   2. This value can be multiplied by the length of the route in DIPs to get the distance.
   3. The system must divide this distance by the maximum boat speed to get an approximate route completion time (Time = Distance / Velocity)
6. API processing
   1. The wind data API (<https://darksky.net/dev>) is first called when the program is launched, this will return the weather information at the specified location longitude/latitude in the API URL
   2. The wind speed must then be displayed in Knots
   3. The wind bearing is displayed by rotating the arrow.
   4. The tidal data API(<https://api.met.no/weatherapi/oceanforecast/0.9/documentation>) is also called when the program is launched, returning tidal stream data at a specified location, this data will be used in the route plotting process.
7. Tidal Processing
   1. The system must first calculate the equation of the cosine wave that represents the tidal flow with respect to time
   2. The system must allow this cos wave to be queried with a time value to return the tidal flow at that time
   3. The system must then use this value for the tidal flow to calculate the fastest the boat can travel through the specified node in the grid, this value can then be returned the fastest route algorithm
   4. The Fastest route algorithm must use the value returned to plot a route using the tidal weighting on each node

Storage:

1. Waypoint locations are stored in a text file, in the format ‘x, y’ with a new line for each waypoint

# Documented Design

## Overview:

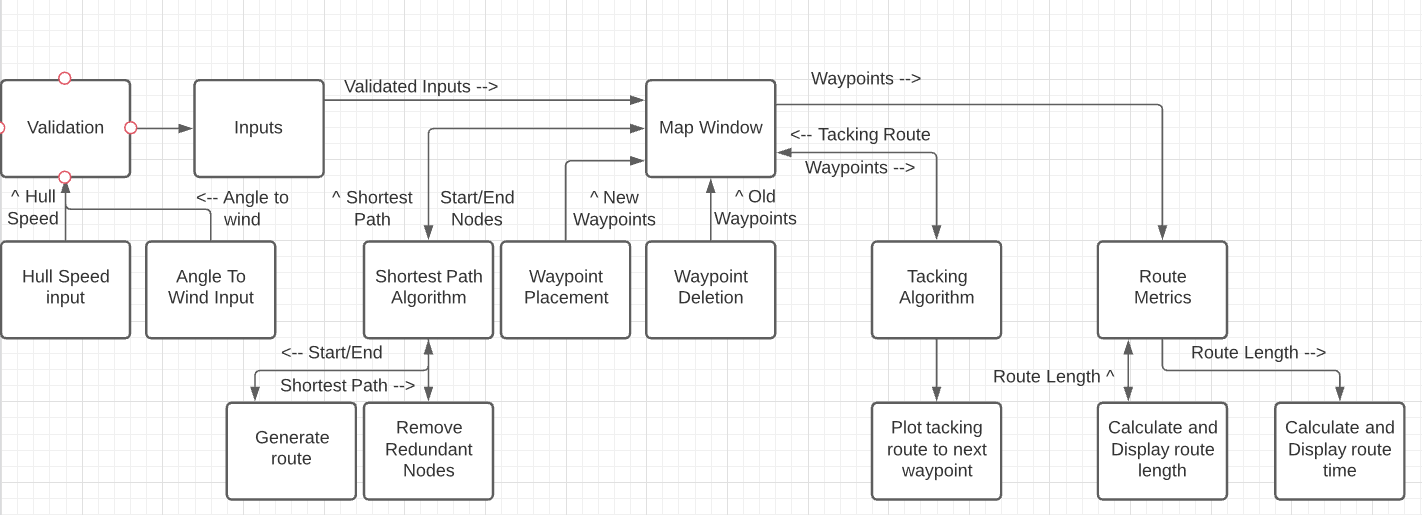
The system I am designing will need to display a map to the user, in the future I could add an option for the user to upload their own maps, the map I chose is of the Isle of Wight since it has interesting features that could expose potential issues with my algorithm. The user will be able to place waypoints on the map, first the system should plot the shortest route between these waypoints, applying real wind and tidal information to these calculations. Next the system will plot the route the boat will actually have to sail, this cannot always be straight lines between the waypoints since it might not sail directly towards each waypoint, this is because if the wind is on the same bearing as the waypoint to the boat, the boat will need to sail directly into the wind. This isn’t possible because the wind won’t fill the sails, to overcome this problem the boat must follow a ‘zig zag’ route towards the waypoint. The system will need to have an algorithm to in order to achieve this pattern. It will also need to be able to save and load previous waypoints so that data isn’t lost when the program is shut down. The algorithm will require a system a detect where the land is and where the sea is in order to plot a route that doesn’t pass through the land. The system will need to be able to switch between specific maps as the user requires meaning all the objects on a map segment should be contained in the map segment so that as the user changes the map segment being displayed the objects on the previous map segment should be removed and the objects on the next map segment will be redrawn.

## IPSO Chart:

|  |  |
| --- | --- |
| Input | Output |
| * Place waypoints with the mouse * Enter the boat’s hull speed with a text input * Enter the boat’s maximum tacking angle * Delete waypoints with the mouse * Navigate to other maps using the keyboard * Start the route plot with the press of a button * Enter a filename to save to or to load from * An API needs to be called to retrieve live wind/tidal data | * A waypoint will be displayed as a circle * Each waypoint will be joined with a black line to illustrate the order they were placed in * When the fastest route is plotted the system will need to draw new waypoints * When the tacking route is plotted the system will display the route the route that the boat will need to travel as a red line * The system will need to display the length of the route in nautical miles * The system will display the time it will take to traverse the route based of the hull speed. * The system will need to display the wind bearing (possibly using an arrow) * The system will need to display the wind speed in Knots. |
| Processing | Storage |
| * If a waypoint is deleted in the middle of the sequence the system will have to calculate where to draw the new connecting line * The system will need to plot the shortest route using the A\* algorithm displayed in the prototype. * The system will need to be able to retrieve the colour of a specific pixel in order to plot a route avoiding land. * The system will need to plot the tacking route between waypoints * The system will need to write waypoint locations to a file and load them from a text file * The system will need to apply the wind/tidal API data in the appropriate places * The system will need to reposition waypoints away from the land if they are too close to plot a tacking route too. * The system will need to validate any user inputs | * Previous waypoint locations will need to be written to a unique file. * Previous waypoint locations will need to be read from an existing file. * A text file should be used to store the land node locations as 1 and water node locations as 0, they should be arranged in the same configuration as the original map, therefore the land detection algorithm will only need to be run once. |

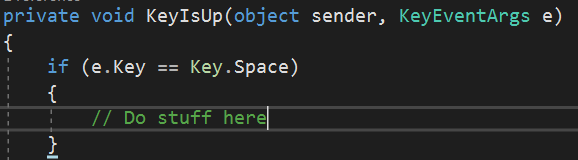
## Class Structure (Initial):

## Structure Chart:



## User Interface:

### Information:

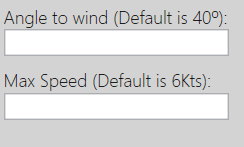
I decided to design my graphical user interface in XAML (Extensible Application Mark-up Language) instead of using C#s built in graphical UI designer because I found it much easier to align several objects using built in structures such as ‘stack panels’ which will vertical or horizontally align all objects contained within the ‘stack panel’ (More info here: <https://docs.microsoft.com/en-us/uwp/api/windows.ui.xaml.controls.stackpanel?view=winrt-19041>). An example of this is shown in the button screenshots below. XAML also allowed me to define my own custom events in the ‘Canvas’ definition (See below), meaning the ‘KeyUp’ event will call the ‘KeyIsUp’ method where the system can differentiate between which keys were pressed.

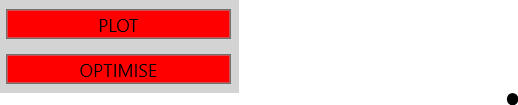
### Requirements:

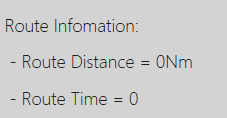
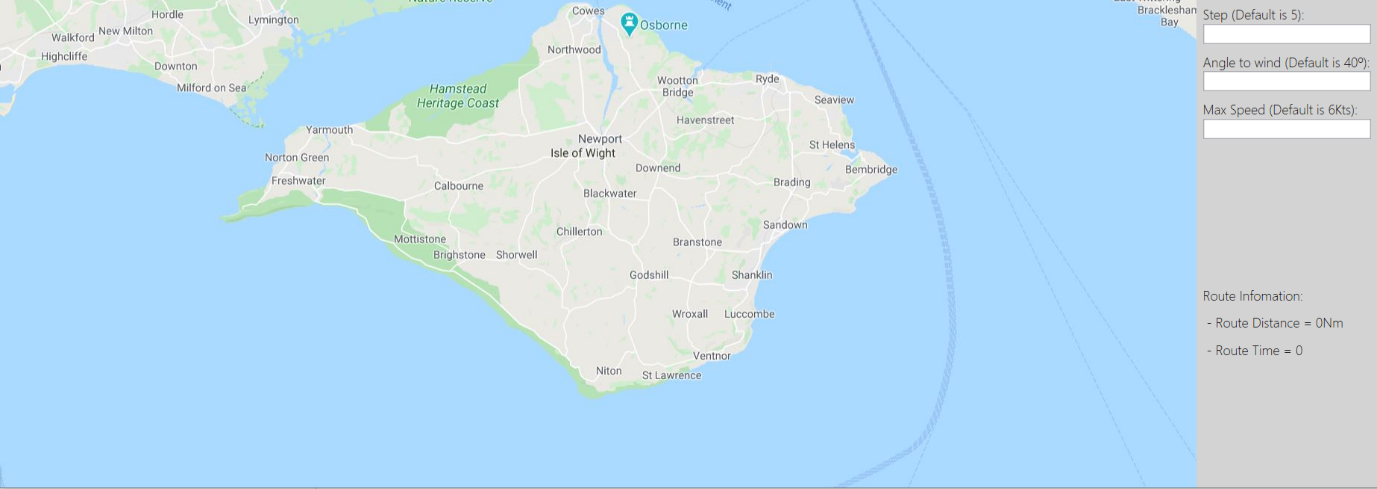
* An Input Text box for the hull speed – this needs to have a default value
* An Input Text box for the boats maximum angle to the wind – Requires a default value
* A map display window
* A ‘plot tacking route’ button
* A ‘plot shortest route’ button
* Route Information display window

### Implementation:

#### Map Window:

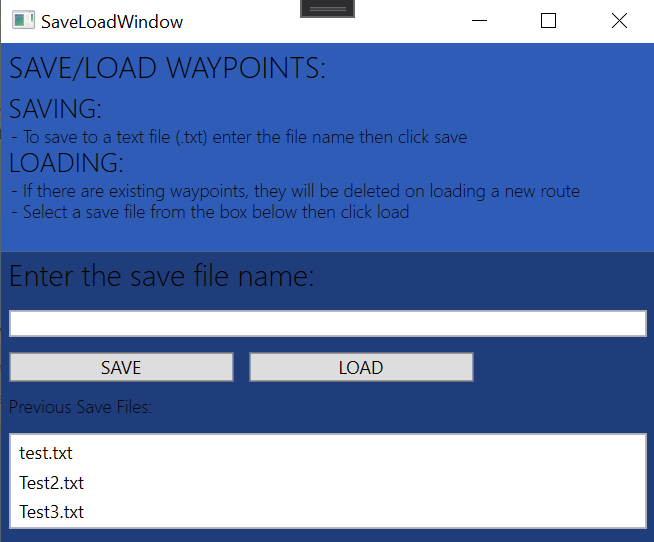
These text boxes will take a string input, if the value in the box changes the program will first validate the input to check that it only contains numbers, the input is then converted to a ‘double’, which is C#s floating point type, the ‘Angle to wind’ and ‘Max Speed’ values are then passed into the setters for the ship object for the current visible map segment. Any new routes plotted will now be calculated with these new values.

The ‘OnPlot’ event will trigger the tacking route algorithm, which will plot the route the boat has to take between existing waypoints. The ‘OnOptimise’ event will trigger the shortest path algorithm which will generate a new set of waypoints between the start and end waypoints that are pre-set by the user.

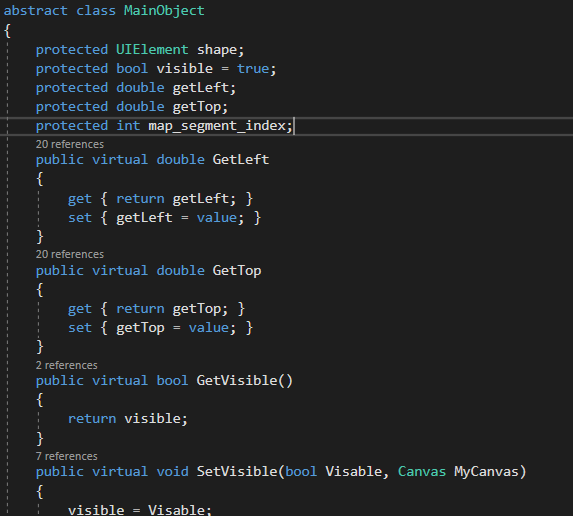
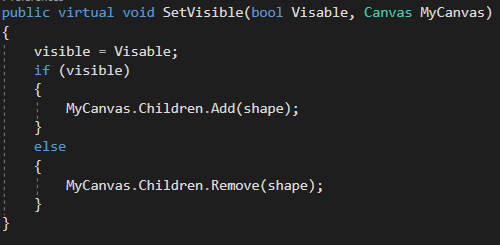
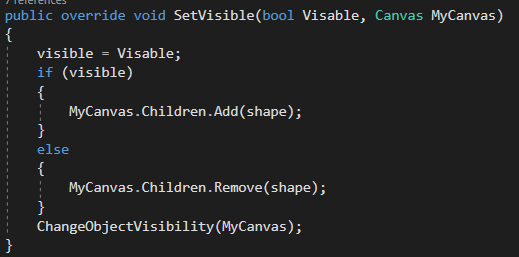
Once the tacking route has been plotted the route distance can then be calculated. Using the scale from the original map, which will have to be hard coded into the program, the system can calculate how many nautical miles per (Device Independent Pixels, see link: <https://microsoft.github.io/Win2D/html/DPI.htm>). Next it can calculate how many DIPs the route is and from that we know the route length in Nautical Miles. This value can be divided by the hull speed (Time = Distance / Velocity) to get the route time.

The map window (shown above) contains a screenshot from <https://www.google.co.uk/maps/>. I choose to use google maps for the map image because out of all the mapping software that I found online, google maps had the most documentation regarding the Hex values for the land and sea colour schemes. This made it far easier to differentiate between which pixels contain land and which one don’t. In the future I plan to add an option for the user to be able to submit their own maps as PNG files, for this to work they will also need to enter the scale of the map in Km per cm. This value can then be converted to Km per DIP. In order to do this, I will have to get the DPI (Dots per inch) of the users monitor. For each of the text inputs in this window I decided to incrementally check for a change in the text input rather than use a button as a trigger event because this will hopefully simplify the UI by reducing the number of things on screen. Later, I learnt that there was an in-built event handler in WPF that checks for a change in a text input however this was too close to the project deadline to implement this at every user input in the program.

#### Save/Load Window:

The save/load window contains several text blocks that display information about the window, explaining how to save and load a save file. It also has a text box input so that the user can enter the save file name. At the bottom there is a list box that contains all the previous save files’ names. So, the user could see the file names and input the file they wanted to load into the text box. I changed this later so that the user could select the file inside the list box and then press the load button. All these windows are designed with stack panels which will scale with the size of the window meaning they should work on any size monitor and they can be resized by the user if necessary.

## Object Oriented:

Object Orientated Programming – Most of the program will be structured in a class-based system, in which all of the physical objects in the program will inherit from the ‘MainObject’ class since that contains all the methods for positioning and drawing objects onto the ‘Canvas’. An instance of this class can never exist on its own which is why it should have the ‘abstract’ modifier. All its methods have the ‘virtual’ modifier which will allow for polymorphism to be used in the child classes (See below). All of its attributes also had to have the ‘Protected’ access modifier such that its child classes would have all the same attributes ([This class structure is displayed here](#_Class_Structure:)). The methods below have the same identifier but different implementations, the ‘virtual’ and ‘override’ modifier allow this to be possible as the override method will be prioritised unless specified using the ‘base’ keyword (base. SetVisible). Objects such as waypoints and lines connecting waypoints will be generated at run-time and stored in lists, these lists will be stored within the Map Segment object that those objects were placed onto, this makes switching between map segments significantly easier because the visibility of the objects on the map can be handled by the map segment that contains those objects. This means that waypoints have a composition association with the map segments as they cannot exist without a map segment. In order to group several related classes that require the same methods, interfaces will be used. For example, both the Ship class and the Waypoint class need to deal with the same angle geometry as they both will have a cone, an interface can be used to group the methods that they both use to handle these operations. Unlike some languages C#s built-in garbage collector means that I don’t have to Initialise every data field inside the constructor, and I don’t need to reclaim and clear the memory of any unused objects.

## Data Structures:

* 2 Dimensional Arrays – There are several uses for 2D arrays throughout the program for example the ‘LandPixelMap’ array, this 2D array of Booleans, represents the map segment as either a section of land (a True) or a section of sea (a False). Another example of a 2D array is in the shortest path algorithm class, where the nodes that represent the shortest path are stored in their respective positions on the map.
* Lists – Lists will be used throughout the program. They are a dynamic data structure similar to Arrays in that they store a collection of indexed items, however the main difference is that lists are dynamic meaning their size can be changed at runtime, they also differ from arrays in that the items cannot be accessed randomly, it must walk the entire chain to locate an item, which makes them slower to access. The system uses lists in several places where it needs to store several objects of the same type, but the exact number of objects is unknown. For example, the waypoints and the lines that join the waypoints together in each individual map segment are all stored in a list of Type ‘MainObject’. This allows the program to iterate through the waypoints and their adjoining lines.
* 1 Dimensional Arrays – These can be used in simple circumstances when the exact number of items to be stored is known, for example a set of coordinates can be easily stored as a one-dimensional array of ‘doubles. Unlike stacks and Queues, arrays and lists are more suited to accessing elements by their index. Collection types such as Arrays and lists also implement C#s ‘IEnumerable<T>’, which is compatible with LINQ, (.NETs built in component for data querying) providing a good basis for adding a SQL database in future versions of the project.
* Dictionaries – Dictionaries can be used when each piece of data has a unique key instead of an index, an example of where this can used in the system is in conversion from JSON API Object to a C# construct, in most cases the mirroring construct in C# will be a dictionary since all the attributes in a JSON Object will have a string key and a value.
* 2 Dimensional Arrays – A 2 dimensional array is used in the system to store the land locations on the map, a section of land is presented as true and water is false, this is useful since the location of the section of the map is represented as its location within the 2D array.
* Queues – A queue is a first in first out collection data structure, they are useful for storing a collection of items that need to be processed in the order they were instantiated. The main used of queues in my system is in the ‘PlaceGeneratedWaypoints’ method. When the user tries to plot a route between more than two waypoints the journey must be subdivided into several legs, so the system will create a ‘ShortestRouteObject’ (which contains the route) for each leg of the journey. These are then added to the back of the queue so that the waypoints that make up the new route will be plotted in the order they were instantiated.

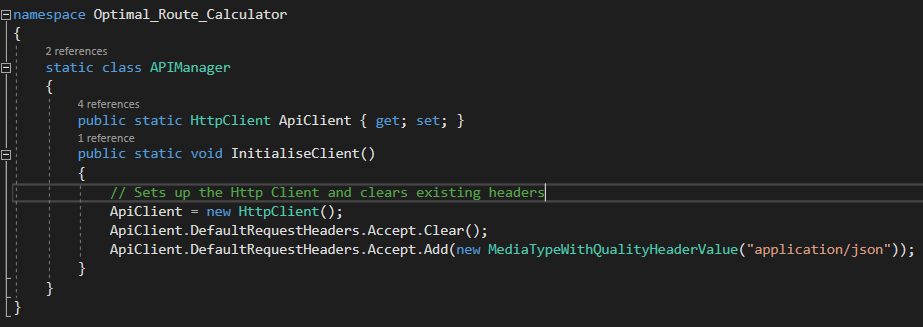
## API Management:

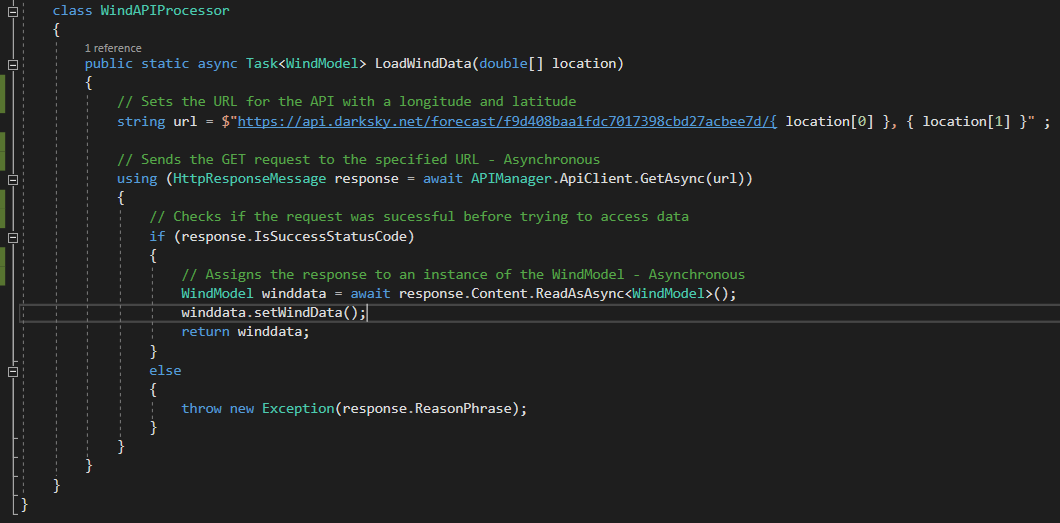
### Information:

The program will require live data for wind speed, wind direction, tidal speed and tidal stream direction in order to calculate an accurate route. Calling the API to get this data must be an asynchronous process (Executed on a separate thread) since API calls are event-driven request-based processes they can take a relatively long time to execute which will freeze the UI whilst the API is called. I first established how to implement asynchronous tasks using the .Net documentation (<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>).

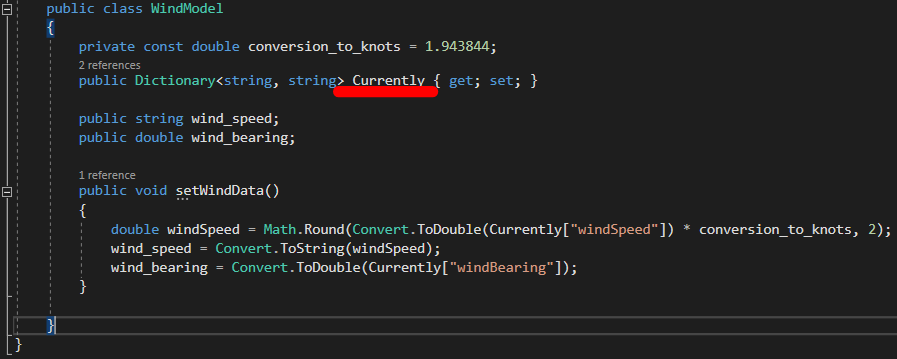
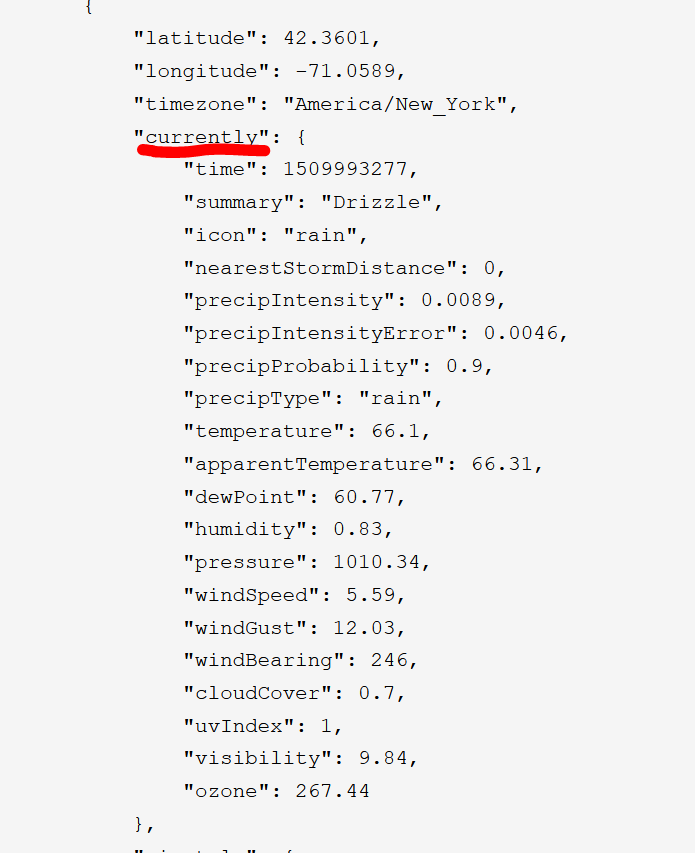
### Implementation:

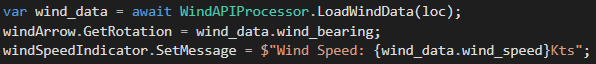
First, I needed to set up an API client for the whole program, that multiple APIs could use. This class has the static modifier because it doesn’t need to be instantiated. This class will clear any default headers. Headers represent the meta-data associated with the API request. In this case I needed to define the content type of the response as JSON, as this is what the website (<https://darksky.net/dev>) uses.





Next, I had to add a class for calling the API and processing its data. The method in this class is asynchronous returning an instance of the ‘WindModel’ class, the location array parameter is then passed into the URL. Subsequently the API is called and if a successful response code is received the data can be assigned to a new instance of the ‘WindModel’ class. This will only happen if the response status code is the success status code otherwise it will throw an exception.

The screenshot on the left shows the JSON documentation for the ‘DarkSky’ API weather data. The structure’s surrounded by curly brackets can be interpreted in C# with a dictionary of strings with the same name as structure in the JSON (red underline). This is shown in the class below. The data is then accessed from the dictionary using the names of the keys listed in the JSON documentation. In this case the wind speed must be converted to Knots by multiplying it by a constant and then rounding to decimal two places.

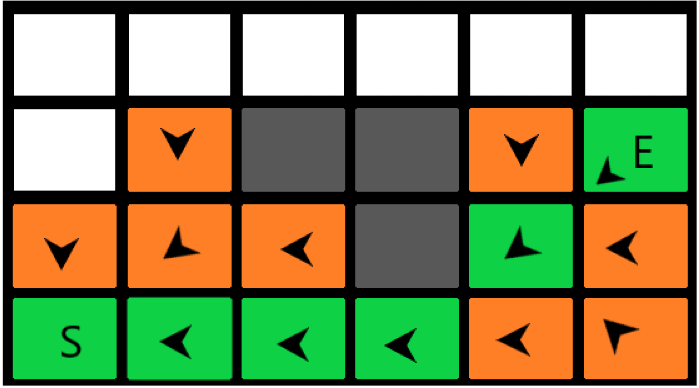
The wind model can then be easily accessed by the rest of the program since it now holds the current wind data for the specified latitude and longitude. This same process can be done with the Tidal API as well, using the same API client since the header can remain the same.

## Algorithms:

### A\* Pathfinding:

#### Overview:

The A\* pathfinding algorithm will need to take in a start and end point, these will be two locations on a grid of nodes, it will then need check the all adjacent nodes to the current node selected, at each of these nodes first it will check if it’s the end node, if not then calculate the G cost and H cost for that node. The G cost is the distance from the starting node and the H cost is the straight-line distance between the end node and the current node. From the sum of these two values the F cost is calculated. The program will then select the node with the lowest F cost as the new current node. As each current node is selected, they are added to a list of closed nodes. Every time a node’s F cost is calculated its parent node is also set. The parent node is the node that this node was accessed from. This means that once the full list of current nodes (nodes with the lowest F costs’ ending at the final target node) is generated, this list can be tracked backwards following the chain of parent nodes until it reaches the start node. The diagram below shows how the nodes point towards their respective parent nodes. So, following the chain of nodes from the end node to the start node will be the shortest route. Finally, these nodes can be added into a list and then reversed to get the final list of nodes in order.



#### Pseudocode:

Once this algorithm finds the end node another method can iterate backwards through the list of closed nodes, each node in this list will have a parent in this list with, the index of the parent can be stored with the current node. That way the shortest path can be determined from the chain of nodes in the list.

OPEN NODES // Group of nodes to be evaluated

CLOSED NODES // Group of nodes already evaluated

OPEN NODES 🡨 Start Node

While (Current Node != Final Node)

Current Node 🡨 Node in OPEN NODES with lowest F Cost

Remove Current Node from OPEN NODES

CLOSED NODES 🡨 Current Node

Foreach Neighbor of the Current node

NeighborGCost 🡨 CalculateNodeWeight(changeInX, changeInY)

If Neighbor is land or neighbor is in CLOSED NODES

Skip to the next Neighbor

End For

If new path to Neighbor is shorter OR Neighbor is not in OPEN NODES

Set F Cost of Neighbor

Set parent of Neighbor to Current Node

If Neighbor is not in OPEN NODES

OPEN NODES 🡨 Neighbor

End While

Sub CalculateNodeWeight(ChangeInX, ChangeInY)

If (Abs(ChangeInX) == Abs(ChangeInY)

Return 1.4

Else

Return 1

End Sub

ShortestPath

selectedNode 🡨 CLOSED NODES.lastNode

While selectedNode != start node

ShortestPath.Add(selectedNode)

selectedNode 🡨 CLOSED NODES[CLOSED NODES.IndexOf(selectedNode.ParentNode)]

End While

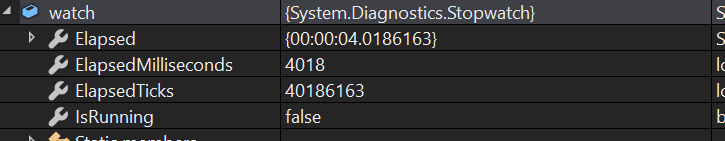
ShortestPath.Reverse

#### Later Improvements:

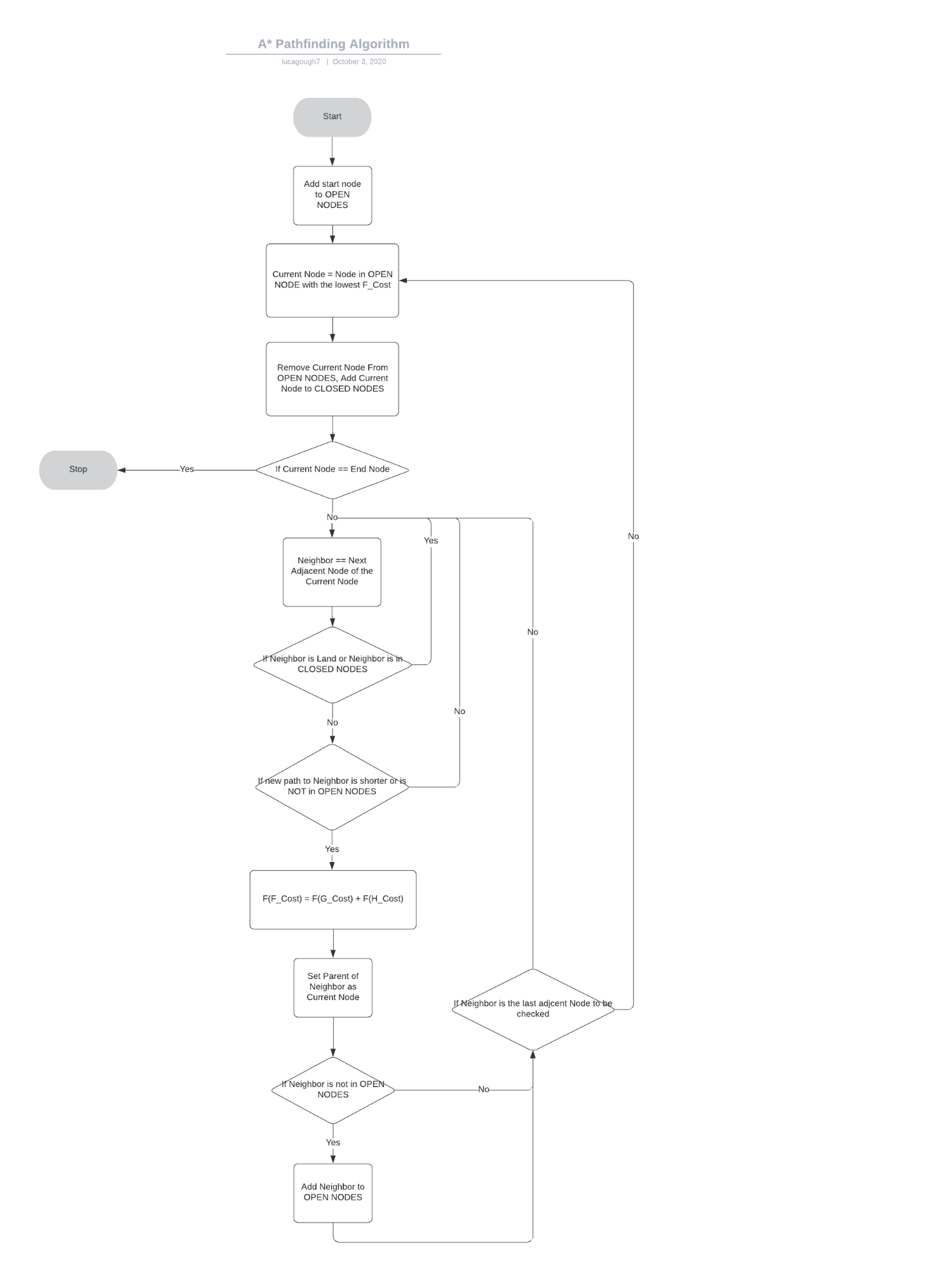
After some research with the original pathfinding prototype I realised that even with multi-threading the shortest path algorithm was fairly slow, the bottleneck in this algorithm was the method which retrieved the colour of a pixel, I found that when calling this method ~2000 times, roughly the number of pixels I would need to check the colour of, was taking up to 30 seconds. I needed to either rework the method for getting the pixel colour or reduce the total number of nodes that I would need to check. In the end I decided to do both.

In order to reduce the number of pixels I would need to check I decided to set a constant distance of pixels between each node, this will reduce the accuracy of the route as it’s not plotting between individual pixels anymore. However, it did nearly half the pathfinding load times. After testing with different values, I decided on a horizontal and vertical node separation of 5 pixels, this ensured the route didn’t cut over much since there would only ever be 5 pixels of land that could be between two nodes.

In addition to this realised that the system for detecting the colour of a pixel was being called on the same pixel several times, which was unnecessary so I decided to call this method on every map pixel when the program is first started and then store the status of each pixel as a Boolean (Land = True, Sea = False) in a 2D array, this ensured that whenever I the pathfinding algorithm needed to access the colour of a pixel all it has to do is query the 2D array with a specified location. Furthermore, I tried to use multithreading in the pixel colour detection method, however I learnt that UI elements cannot be accessed on separate threads, meaning I couldn’t access the ‘Canvas’ and get the colour of a specified location from a second thread.

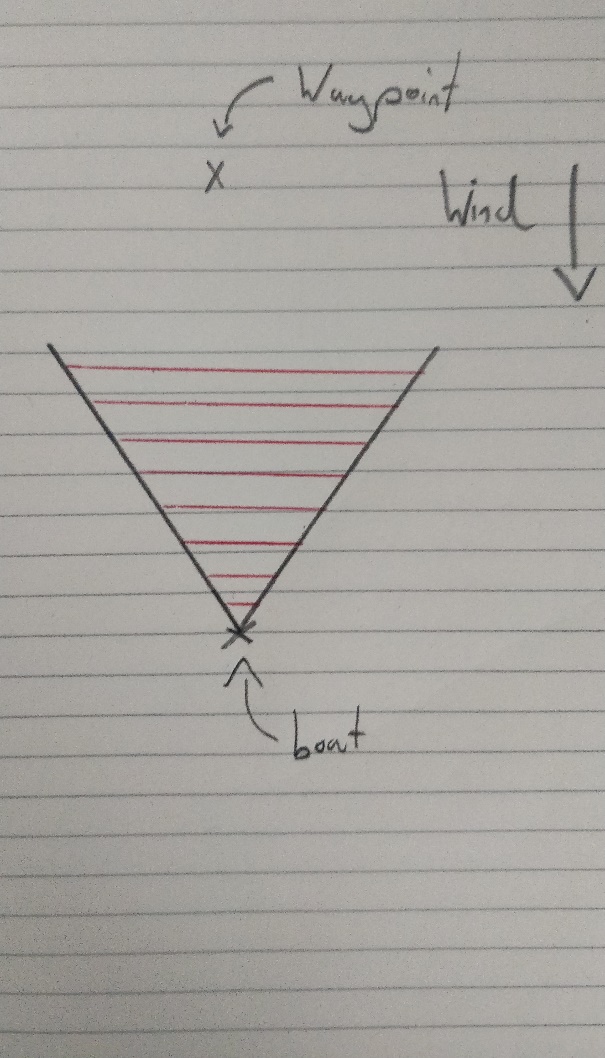
Both improvements lead to a great reduction in the route processing time, for example calculating the original route that took 30 seconds on the old algorithm now took 4 seconds.

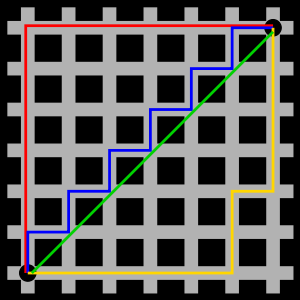
#### Flow Chart: -- Make This A3 in final project

* I made this flowchart in Lucid Chart ([https:// www.lucidchart.com/pages/](https://www.lucidchart.com/pages/))

### Tacking Route Algorithm:

#### Overview:

The tacking route algorithm will need to plot a route between all the waypoints in the order they were placed by the user. First the algorithm needs to determine the angle to the waypoint from the last waypoint, it can then know if the boat can sail directly towards the waypoint, since the system already knows the wind bearing. If the bearing of the next waypoint from the current waypoint is inside the 45° cone either side of the wind bearing. This means the boat will have to sail on the 45° cone edge to get sail fastest towards the waypoint (assuming the wind speed is fast enough to sail at hull speed on a close reach). Next the system will have to decide when to sail on the other tack (the other 45° cone edge), most people tend to judge this in the moment since it doesn’t matter how many tacks you make the route length will always remain the same. This concept is known as taxicab geometry (<https://en.wikipedia.org/wiki/Taxicab_geometry>).

In this case I decided to make the algorithm perform a tack once it reached 10° either side of the initial bearing of the next waypoint, I chose 10° because it ensures the boat will never travel too far away from the waypoint in either axis ensuring that if the wind changes direction it can never be very far downwind. That way in order to get the tacking location all the algorithm had to was locate the intersection of the 45° wind cone and the 10° tacking cone. This process can be completed with some simple linear geometry.

First the algorithm will need to turn the bearings into gradients since I needed to plot the route lines in cartesian coordinates rather than in their current polar form, which can be defined by Gradient = Tan(Theta), this can be derived from the fact that, where m is the gradient and . Once the gradient has been ascertained both equations need to be normalised into the form Y = mx + c, for this the system must find a value for C, the y intercept. This can be found since we have point for each line that they pass through, meaning we can substitute those values into the equation to get the equation of each line. Finally, to get the intersection the lines can be set to equal each other, this process eliminates the unknown variable Y and allows us to solve for X, finally this X value can be substituted back into either equation to get the Y value. This (X, Y) point is the end point of the first tacking route line and the start of the next one, meaning this process can be repeated until the boat can sail towards to final waypoint, once it reaches the waypoint the system can display these individual route lines using the locations calculated. This process will be completed using recursion since it will simplify the iterative instead of embedding lots of while loops, since the program is small and the maximum number of subroutine calls that can be made is constrained by the map size, the call stack will not be impacted too much.

#### Pseudocode:

##### Intersection Algorithm:

Tack Cone Point

Wind Cone Point

// M = Tan(theta)

Tack Cone Gradient 🡨 Tan (TackConeBearing)

Wind Cone Gradient 🡨 Tan (WindConeBearing)

// C = M \* X + Y

Tack Cone Y-Intercept 🡨 Tack Cone Gradient \* Tack Cone Point.X + Tack Cone Point.Y

Wind Cone Y-Intercept 🡨 Wind Cone Gradient \* Wind Cone Point.X + Wind Cone Point.Y

XF = (C1 – C2) / (M1 – M2)

Final X pos 🡨 (Tack Cone Y-Intercept – Wind Cone Y-Intercept) / (Tack Cone Gradient – Wind Cone Gradient)

YF = M1 \* XF + C1

Final Y pos 🡨 Tack Cone Gradient \* Final X pos + Tack Cone Y-Intercept

Plotting Algorithm:

CalcNextRouteLine (Waypoint)

{

If Boat can sail towards waypoint:

nextLocation 🡨 TargetWaypointLocation

PlaceRouteLine (shipLocation, nextLocation)

Else:

NextLocation = CalculateLineIntersection (Wind Cone Bearing, Tack Cone Bearing)

PlaceRouteLine (shipLocation, NextLocation)

shipLocation 🡨 NextLocation

ship.WindConeSwapSide

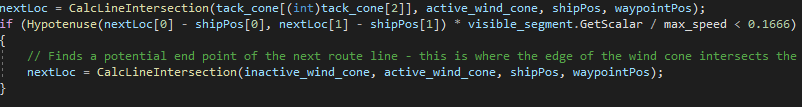
waypoint.TackConeSwapSide

CalcNextRouteLine(Waypoint)

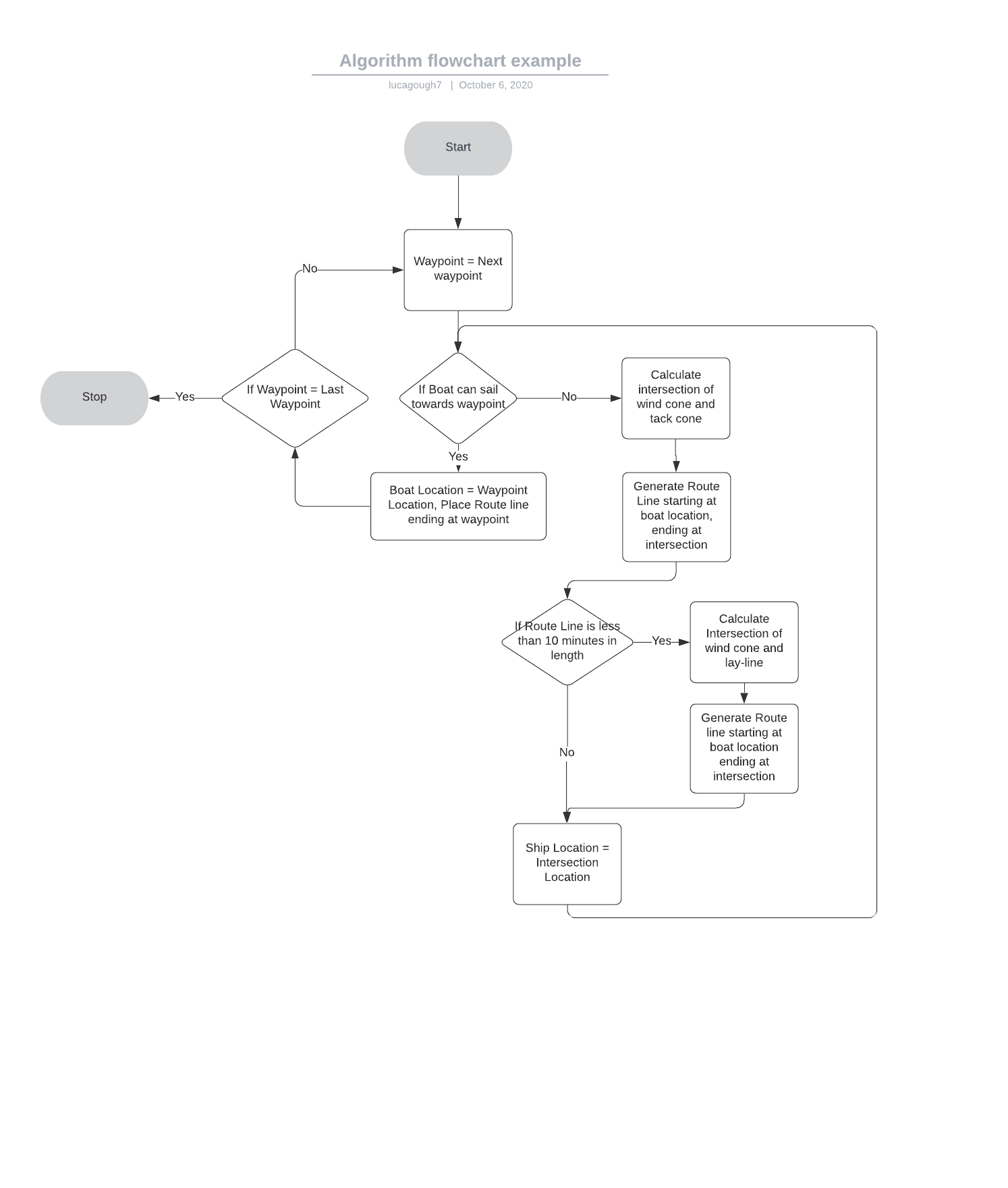
}

#### Later Improvements:

This algorithm had an issue that was immediately obvious. Since the tacking cone converges on the waypoint the algorithm was tacking at shorter and shorter intervals, theoretically it would tack infinitely before it reached the waypoint the system would eventually round the coordinates to the integer coordinates of the waypoint. However sometimes this didn’t occur, and the tacks were still becoming impossibly small and it would tack past the waypoint.

In order to fix this issue, I added a special case, if the next tacking line to be plotted will take less than 10 minutes to sail then the boat should tack at the lay line instead. The lay-lines are the lines that form a cone 45° either side of the wind bearing on that waypoint, this means that the boat can always sail along a lay-line to reach the waypoint therefore if the boat tacks exactly where its current path will cross a lay-line the boat can then sail directly towards the waypoint. This ensures the tacks can never be too small whilst still making progress towards the waypoint.

#### Flow Chart:



### Waypoint Manipulation Algorithm:

#### Overview:

The system needs a way for the user to place and delete waypoints using the mouse buttons, these waypoints should be linked in the order they were placed with a black line so that the user knows the order in which the boat should traverse the route. Furthermore, if the user decides to delete a waypoint the previous and subsequent waypoints should be linked by a new line and the two lines which linked them to the deleted waypoint should be deleted. In order to do this, I decided to store lines and waypoints in the same enumerable data structure. Since they are from the same base class, I was able to create a list of that base class, this made accessing these elements much easier than having them in several different lists. The user should only be able to place waypoints in sea locations meaning there needed to be input validation for the mouse location. In this case I was able to able to discern the weather the mouse clicked on a land location by querying the existing array of Booleans.

#### Pseudocode:

##### Waypoint Placement:

Coords[] 🡨 MousePosition

If PixelIsntLand(Coords)

New\_waypoint 🡨 New Waypoint(Coords[])

WaypointsAndLines.Add(new\_waypoint)

If new\_waypoint is not the first waypoint

Previous\_waypoint = WaypointsAndLines[WaypointsAndLines.Count - 2]

PlaceWaypointLine(new\_waypoint\_location, previous\_waypoint\_location)

##### Waypoint Deletion:

Index 🡨 WaypointToDeleteIndex

DelWaypointsOrLine(index) // Removes lines/waypoints from the canvas then deletes from the array

Count 🡨 WaypointsAndLines.Count

If Count != 0

If Count == Index

DelWaypointOrLine(index – 1)

Else if index == 0

DelWaypointOrLine(index)

Else

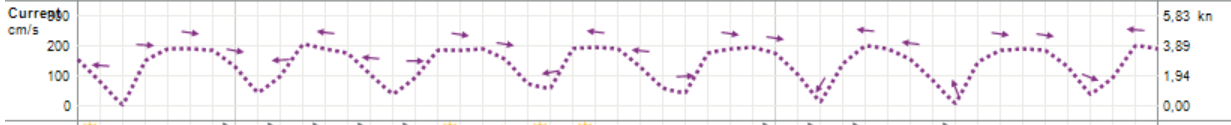
DelWaypointOrLine(index - 1)

DelWaypointOrLine(index - 1)

PlaceWaypointLine(previous waypoint location, next waypoint location)

### Tidal Stream Algorithm:

#### Overview:

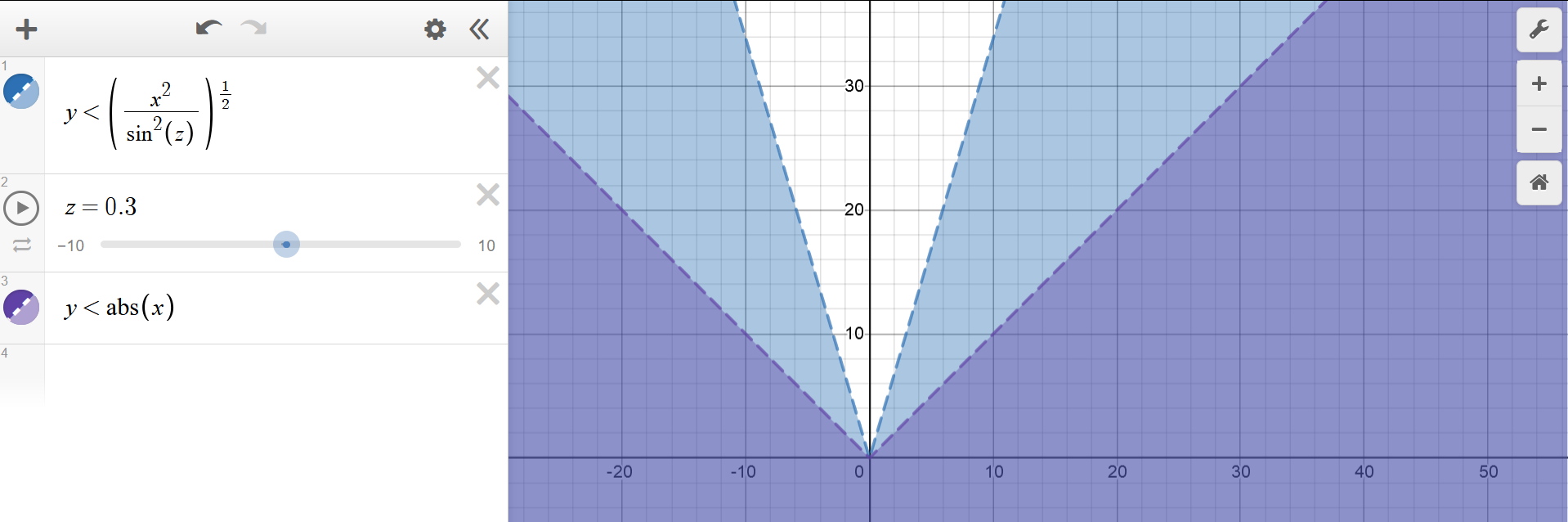
The tidal data was much harder to acquire than the wind data because its very niche, however I eventually found an API (<https://api.met.no/weatherapi/documentation>) that would provide this information. This website also had a request cap meaning I would need to limit my requests if possible. In order to achieve this I decided that if my program could predict the tidal velocity and direction with some accuracy then I would only need to store a few starting values, from their I could build a mathematical function using a time value for an input. After looking at some existing data from a specific location over a few tidal cycles I concluded that this pattern could be approximated with a cosine wave, meaning I would only need to know the range of the wave in order to construct it, in this case it would be the fastest and slowest points of tide. The direction of the tide was easier to calculate as it will also flip 180° each time the cosine wave passes a minimum. I also decided to store the amplitude of the wave in a text file that way I would only need to call the API once per launch of the program to calibrate the position of the wave. In order to ascertain the amplitude of the wave I originally intended to write a small program that would call the tide height API periodically. This is because the derivative of the distance function is the velocity function and meaning the velocity function is 90° ahead of the distance function. I then would store all these values in a text file which can be used by the main program to construct the cosine wave. However, with the time constraints of the assignment I decided to save time and allow the user to set their own custom tide points the user can enter the maximum and minimum flow and the bearing of the tide. Which will flip 180 degrees every 6.2 hours when the tide changes direction. Therefore the only data I would need to access from an API would be the time until the next high water, since I can use that value to offset the cos function in the x-direction which will synchronise my model for the tide with the real life tidal flow.

#### Inputs:

For the tidal algorithm to work it will require a maximum and minimum input for the tidal flow and one of the tidal bearings at that point, the program should automatically rotate the direction every 6hrs (at each minimum). Next, I decided to display to user the actual tidal flow that they are describing with these values by displaying the actual cosine graph. In order to do this I decided to use a library – OxyPlot ([https://oxyplot.readthedocs.io/en/latest/](https://oxyplot.readthedocs.io/en/latest/%20)) – since its fairly easy to use. All I needed to do was create a new Function Series which is a series of points that all lie on the same function. These points are created iteratively stepping the function input each time. Finally, I assigned the function series to a Plot Model which is the graph itself. This plot model was instantiated in the XAML code so after adding the function series the graph will automatically update.

#### Applying the data:

Next, I needed to be able to query the cos function at these tide points so that I could apply that data to the shortest path algorithm. This means that the weighting of the travel time between nodes now depends on the direction that the node is entered from. This now presented a new problem as if the boat enters the node at oblique angles to the tide bearing the tidal stream will now affect the direction that the boat is travelling in, to simplify this issue I assumed that the boat will continue travelling on the same bearing and instead the boat can point further towards the tide meaning a greater proportion of its total velocity opposes the tide meaning its total velocity is reduced but it continues travelling in the same direction as before the tide was applied. In order to calculate this new velocity I needed to derive an equation, I tried two methods using the sine rule and cosine rule to produce these two equations where U = Initial Velocity, V = Final Velocity, ϴ = Entry Angle, R = Tidal Velocity, and The second equation is much simpler so I decided to use that one however it’s a quadratic therefore it will produce two solutions meaning the program will have to discern which one is the right solution (Non-Negative). Another issue became apparent when I needed to calculate the entry angle since in order for the equation to work the angle has to be acute to achieve this geometrically I set the tide bearing parallel to the y-axis, this allowed me to calculate the acute angle between the initial velocity vector and the x-axis. Finally, I was able to construct a triangle between the resultant velocity vector, the tidal velocity vector and the boats velocity vector. This triangle is the same triangle I used to derive the equations listed above meaning this angle would work with these equations. The quadratic also has a chance to produce an imaginary root, this only occurs when the tidal velocity is greater than the boat’s velocity in which case the boat will not be able to sail into that node. This can be shown mathematically be substituting the quadratic equations coefficients into its discriminant, this then forms the following equation which is true for values of r which will produce real roots, . I then graphed this function in Desmos (<https://www.desmos.com/calculator>) to show the range of values for which r is valid.



This shows that if is true then the quadratic will produce real roots and the boat can sail into that node. This means that the program should never have an error with square rooting a negative since if the tidal velocity is ever greater than the boat’s velocity the angle between the projected final velocity and the initial velocity will be greater than 90 degrees meaning the boat cannot travel through that node. Finally, the final velocity value calculated by the quadratic equation is then passed into the shortest route algorithm which has the effect of applying one-way resistance to a group of nodes near each tide point. This causes the shortest path algorithm to plot around areas of stronger tide.

#### Implementation:

TidalFlow 🡨 Amplitude \* Cos(TimeUntilNextHighWater \* PI \* Freq)^2 + MinimumFlow

FinalVelocity 🡨 Abs(TidalFlow \* Sin(EntryAngle) – Sqrt((2 \* Tidal Flow \* Sin(EntryAngle))^2 – 4 \* TidalFlow^2 \* InitialVelocity^2) / 2)

If (Abs(ChangeInX) == Abs(ChangeInY)

NodeWeight 🡨 1.4 / FinalVelocity

Else

NodeWeight 🡨 1 / FinalVelocity

### Map Generation Algorithm:

#### Overview:

First, I needed an algorithm to detect whether a location contains land or sea. I originally tried to retrieve the byte stream from the bitmap itself and then parse those colours into a 2D array however I ran into several issues as the colours that I was retrieving from the bitmap didn’t seem to match those on the image. I suspected that I wasn’t formatting the data correctly however after spending too much time and making little progress I developed a new method. First, I needed to create a new 1x1 rectangular shaped bitmap with no fill or outline. Next, I could use the inbuilt WPF “CopyPixels” method which takes parameters of a byte array and the stride of the bitmap. The stride is the number of bytes used to store one row of a bitmap. In this case the stride should be 4 because the row is 1 pixel long and each pixel requires 4 bytes for 32-bit graphics. The 4 bytes are returned into the byte array which can then be converted into an ARGB colour. This system of sampling the pixel colours is slow because it must do multiple operations so this will be an area for optimisation in later versions.

On the first launch of the program it must generate a 2D array for each map that specifies whether a location contains land or sea. This 2D array should then be written to file so that when the same Map segment is instantiated a second time the 2D can be loaded from the text file. This makes accessing the terrain state of a location much faster. Generating these map files will still take a fairly long time since it would have to access around 1,500,000 pixels, in order to reduce the number of pixels my program would have to sample I decided to make the program sample every 4 pixels which brings the number of pixels down to around 50,000. I chose 4 because after some testing this created a good balance between the speed of route plotting and the accuracy of the terrain array.

#### Pseudocode:

Sub GenerateLandMap()

Const NODE\_SEPERATION 🡨 4

Int[] MapSegmentDimensions

Bool[,] TerrainMap 🡨 new bool[MapSegmentDimensions[0] / NODE\_SEPERATION, MapSegmentDimensions[1] / NODE\_SEPERATION]

For (i = 0 to MapSegementDimensions[0] / NODE\_SEPERATION; i++)

For ( f = 0 to MapSegmentDimensions[1] / NODE\_SEPERATION; f++)

TerrainMap[i, f] 🡨 PixelIsLand(i, f)

End For

End For

End Sub

Sub Bool PixelIsLand(row, column)

Colour 🡨 GetPixelColour(row, column)

If (Colour is between a range of blue colours)

Return false

Else

Return true

End Sub

Sub Color GetPixelColour(row, column)

DrawingVisual.Open().DrawRectangle(new VisualBrush ViewBox = {column, row}, 1, 1)

Bitmap 🡨 new RenderTargetBitmap()

Bitmap.Render(DrawingVisual)

Byte[] bytes 🡨 new Byte[4]

Int Stride 🡨 4

Bitmap.CopyPixels(bytes, istride)

Return Color.FromArgb(byte[0], byte[1], byte[2], byte[3])

End Sub

Sub LoadMapFromFile(path)

Bool[,] TerrainMap 🡨 new Bool[SegmentDimensions[0] / NODE\_SEPERATION, SegmentDimensions[1] / NODE\_SEPERATION]

line\_ptr 🡨 0

Using (StreamReader terrain\_file 🡨 new StreamReader(Path))

While ((line 🡨 file.readline()) != null)

char\_ptr 🡨 0

foreach character in line

TerrainMap[line\_ptr, char\_ptr] = character == ‘1’

char\_ptr++

End For

End While

End Using

End Sub

Sub GenerateTerrainMapFile(path)

Using (StreamWriter file 🡨 new StreamWriter(path))

For (i 🡨 0 to TerrainPixelMap.RowCount)

Line 🡨 “”

For (f 🡨 0 to TerrainPixelMap.ColumnCount)

Line += TerrrainPixelMap[i ,f] == true ? “1” : “0”

End for

File.WriteLine(line)

End For

End Using

End Sub

#### Code:

public bool PixelIsLandFromVisual(int pixel\_row, int pixel\_col)

{

MapSegmentObject Segment = GetFullMap.VisibleSegment();

double[] segment\_dimentions = { Segment.GetHeight, Segment.GetWidth };

Color colour = GetPixelColor(Segment.GetRectangle, pixel\_row, pixel\_col, segment\_dimentions);

if (colour.R <= 225 && colour.R >= 185 && colour.G <= 180 && colour.G >= 145 && colour.B <= 255 && colour.B >= 235)

{

return false;

}

else

{

return true;

}

}

public Color GetPixelColor(Visual visual, int row, int col, double[] segmentDimentions)

{

GC.WaitForPendingFinalizers();

// Viewbox uses values between 0 & 1 so normalize the Rect with respect to the canvas Height & Width

Rect percentSrceenRec = new Rect(col / segmentDimentions[1], row / segmentDimentions[0],

1 / segmentDimentions[1], 1 / segmentDimentions[0]);

var bmpOut = new RenderTargetBitmap((int)(DPI.X / 96.0),

(int)(DPI.Y / 96.0),

DPI.X, DPI.Y, PixelFormats.Default); // generalized for monitors with different dpi

DrawingVisual drawingVisual = new DrawingVisual();

using (DrawingContext dc = drawingVisual.RenderOpen())

{

// Creates the rectangle without drawing it

dc.DrawRectangle(new VisualBrush { Visual = visual, Viewbox = percentSrceenRec }, null, new Rect(0, 0, 1.0, 1.0));

}

bmpOut.Render(drawingVisual);

var bytes = new byte[4];

int iStride = 4; // = 4 \* bmpOut.Width (for 32 bit graphics with 4 bytes per pixel - 4 \* 8 bits = 32)

bmpOut.CopyPixels(bytes, iStride, 0);

return Color.FromArgb(bytes[0], bytes[1], bytes[2], bytes[3]);

}

public static Point GetScreenDPI(Visual visual)

{

PresentationSource source = PresentationSource.FromVisual(visual);

Point Dpi;

Dpi = new Point(96.0 \* source.CompositionTarget.TransformToDevice.M11, 96.0 \* source.CompositionTarget.TransformToDevice.M22);

return Dpi;

}

public void GenerateTerrainMap(MainWindow main\_window)

{

double[] segment\_dimentions = { GetHeight, GetWidth };

bool[,] terrain\_map = new bool[(int)segment\_dimentions[0] / NODE\_SEPERATION + 1, (int)segment\_dimentions[1] / NODE\_SEPERATION + 1];

for (int i = 0; i < segment\_dimentions[0] / NODE\_SEPERATION; i++)

{

for (int f = 0; f < segment\_dimentions[1] / NODE\_SEPERATION; f++)

{

terrain\_map[i, f] = main\_window.PixelIsLandFromVisual(i \* NODE\_SEPERATION, f \* NODE\_SEPERATION);

}

}

TerrainPixelMap = terrain\_map;

}

public bool TerrainMapFileExists { get; set; } = false;

/// <summary>

/// Checks the if TerrainMapFile exists, if not a new one is generated

/// </summary>

public void CheckForMapFile()

{

// Checks if the terrain map already exists

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

string path = "TerrainMapFiles\\" + $"Map{map\_segment\_index}";

path = FileHandler.PathToAppDirectory(path, 0);

if (!File.Exists(path))

{

GenerateTerrainMap(main\_window);

GenerateMapFile(path);

TerrainMapFileExists = true;

}

else

{

LoadFromMapFile(path);

}

}

/// <summary>

/// Creates the terrain map file and writes the terrain map as 1s and 0s to that file

/// 1 = Land, 0 = Water

/// </summary>

/// <param name="path"></param>

private void GenerateMapFile(string path)

{

string line;

using (StreamWriter file = new StreamWriter(path))

{

for (int i = 0; i < TerrainPixelMap.GetLength(0); i++)

{

line = "";

for (int f = 0; f < TerrainPixelMap.GetLength(1); f++)

{

// If its land then place 1, if its sea then place a zero

line += TerrainPixelMap[i, f] == true ? "1" : "0";

}

file.WriteLine(line);

}

}

}

/// <summary>

/// Loads the terrain map from the terrain map file

/// </summary>

/// <param name="path"></param>

private void LoadFromMapFile(string path)

{

double[] segment\_dimentions = { GetHeight, GetWidth };

bool[,] terrain\_map = new bool[(int)segment\_dimentions[0] / NODE\_SEPERATION + 1, (int)segment\_dimentions[1] / NODE\_SEPERATION + 1];

string line;

int line\_num = 0;

int char\_num;

using (StreamReader file = new StreamReader(path))

{

// Iterates through the text file line by line

while ((line = file.ReadLine()) != null)

{

char\_num = 0;

// Iterates through each line

foreach (char character in line)

{

// If character == '1' return true

// Else return false

terrain\_map[line\_num, char\_num] = character == '1';

char\_num++;

}

line\_num++;

}

}

TerrainPixelMap = terrain\_map;

}

### Redundant-Node Removal Algorithm:

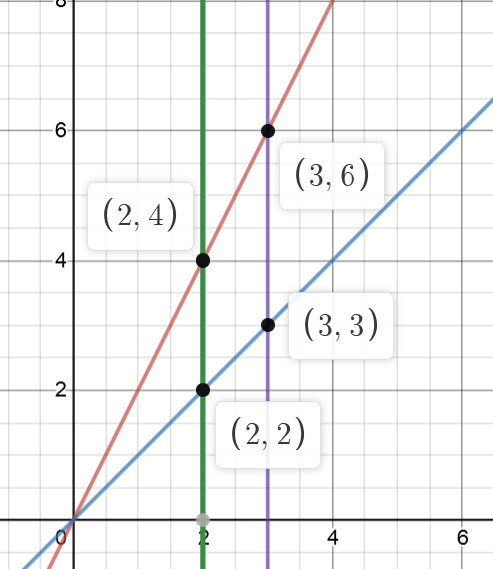
#### Overview:

The pathfinding algorithm will generate a list of nodes which are evenly spaced apart, with a separation of 4 DIP. This creates a problem as the program will then place new waypoints every 4 DIPs and the tacking algorithm will try to plot between these tightly packed waypoints resulting in very small tacks. In order to prevent this the program needs to remove any of the redundant nodes. I decided that there should only be a waypoint where the boat must turn to avoid land. From this rule I was able to develop an algorithm. Any nodes that a previous node in the chain has line of sight on should be removed since they are redundant. This means that the system should start at the first node and identify if there is land between the first node and every other node ahead in the chain. It should then remove every node that it has line of sight on excluding the last node in the chain that it has line of sight on. This next node becomes the new point to check all the nodes in front from. In order to do this my system will need a method for detecting whether a node has line of sight on nodes ahead in the chain, it could do this by getting the equation of the line connecting the two nodes and then stepping forward through the line sampling the terrain beneath the line. If it ever detects land, then the system knows that the previous node doesn’t have line of sight on the subsequent node.

Example:

* In this example nodes 2 and 3 are redundant because node 4 can be accessed in a straight line from node 1
* Algorithm:
  + Node 1 – Line of sight:
    - Node 2
    - Node 3
    - Node 4
  + Remove Node 2 and 3:
  + Node 4 – Line of sight:
    - Node 5



To work out whether nodes have line of sight on each other I needed to write an algorithm to step through the line connecting the two nodes and checking if the line ever passes over land. Since the cartesian equation of the line is generally an equation of Y in terms of X, this means that if the step is the same for every line the change in distance between the points sampled will not remain constant. This is demonstrated with the diagram on the right, the length of the right line between X = 2 and X = 3 is whereas the distance between X = 2 and X = 3 on the blue line is . In order to get a constant distance between two points on any gradient line I needed to form an equation for the step in X value in terms of the gradient. I started with the arc length equation

Where a is the step in X value and dy/dx is the derivative of the original line equation. Since the curve is linear the derivate will be a constant which in all cases is the gradient of the line. This means we can rewrite the equation as finally this can be rearranged to . Its clear to see that this equation is valid for all values of the gradient because the gradient can never be negative as its squared and the +1 inside the root means its always non-zero in effect the denominator is always non-zero. I can then change the Arc Length constant to optimise accuracy and plotting time. Once the step in X is calculated this can then by substituted into the original line equation iteratively, starting at the start node and ending at the end node adding the step value to itself each time, which will output the Y coordinate. At each of the points produced by the equation these can be inputted into the ‘PixelIsLand’ method to detect whether the line crosses over areas of land.

#### Pseudocode:

routeNodes

for (i 🡨 0 to routeNodes.Count; i++)

List<Node> NodesToRemove

For (f 🡨 i + 1 to routeNodes.Count; f++)

If (LineIntersectsLand(routeNodes[i].coords, routeNodes[f].coords)

NodesToRemove.Add(routeNodes[f])

End For

Foreach node in NodesToRemove

routeNodes.Remove(node)

End For

End For

Sub LineIntersectsLand(lineStart, lineEnd)

landPixelCount 🡨 0

step, Gradient, YIntercept 🡨 CalculateLineEquation(lineStart, lineEnd)

For (X = lineStart.X to LineEnd.X; X += step)

Y 🡨 Gradient \* lineStart.X + YIntercept

If PixelIsLand(X, Y)

LandPixelCount++

End For

If LandPixelCount > 2

Return true

Return false

End Sub

Sub CalculuteLineEquation(lineStart, lineEnd)

Gradient 🡨 (lineStart.Y – lineEnd.Y) / (lineStart.X – lineEnd.x)

YIntercept 🡨 Gradient \* lineStart.X + lineStart.Y

If (Gradient == PositiveInfinity)

Gradient 🡨 999999

Else if (Gradient == NegativeInfinity)

Gradient 🡨 999999

Step 🡨 ArcLength / sqrt(1 + Gradient^2)

Return Step, Gradient, YIntercept

End Sub

#### Code:

public static double[] CalculateLineEquation(double[] line\_pos)

{

// Gradient = Change in Y / Change in X

double grad\_to\_node = (line\_pos[1] - line\_pos[3]) / (line\_pos[0] - line\_pos[2]);

double X\_dist\_to\_node = line\_pos[0] - line\_pos[2];

double Y\_intercept = grad\_to\_node \* -line\_pos[0] + line\_pos[1];

// If line is near-vertical Y\_intercept will default to infinity

// This ensures that it can never be infinity

if (double.IsPositiveInfinity(Y\_intercept))

{

Y\_intercept = 99999;

}

if (double.IsNegativeInfinity(Y\_intercept))

{

Y\_intercept = -99999;

}

double step\_in\_X = LINE\_STEP\_DIST / Math.Sqrt(1 + (grad\_to\_node \* grad\_to\_node));

if (X\_dist\_to\_node > 0)

{

step\_in\_X \*= -1;

}

return new double[] { step\_in\_X, grad\_to\_node, Y\_intercept };

}

public bool LineIntersectsLand(double[] line\_pos)

{

// 0 = Step, 1 = Gradient to node, 2 = Y-intercept

double[] line\_info = CalculateLineEquation(line\_pos);

int LandPixelCount = 0;

double Y;

for (double X = line\_pos[0]; X > line\_pos[2] + 1 || X < line\_pos[2] - 1; X += line\_info[0])

{

// Y = mX + c

Y = line\_info[1] \* X + line\_info[2];

if (PixelIsLand((int)Y, (int)X))

{

LandPixelCount++;

if (LandPixelCount > 10)

{

return true;

}

}

}

return false;

}

## Technical Solution:

### MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Threading;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

readonly DispatcherTimer mainTimer = new DispatcherTimer();

public WindArrow windArrow;

private TextBlockObject waypointTimeIndicator;

private TextBlockObject windSpeedIndicator;

private TextBlockObject routeDistanceIndicator;

private TextBlockObject routeTimeIndicator;

private TextBlockObject clockDisplay;

private delegate void NoArgDelegate();

private const double WAYPOINT\_RADIUS = 25;

private const double NODE\_SEPERATION = 4;

private const double LINE\_STEP\_DIST = 1;

private readonly double HEIGHT = 620;

private readonly double WIDTH = 1250;

private DateTime plot\_time;

private double wind\_angle = 40;

private double max\_speed = 6;

private Point DPI;

private bool first\_run = true;

public MainWindow()

{

InitializeComponent();

// Initialises API

APIManager.InitialiseClient();

WindAPICall();

GenerateMap();

// Attaches Main loop as an event handler to the Timer

mainTimer.Tick += MainLoop;

mainTimer.Interval = TimeSpan.FromMilliseconds(20);

mainTimer.Start();

MyCanvas.Focus();

Height = HEIGHT;

Width = WIDTH;

}

private async void WindAPICall()

{

windArrow = new WindArrow(MyCanvas);

windSpeedIndicator = new TextBlockObject(0, 0, "Wind Speed: 0 Kts", MyCanvas, 22, 1);

WindModel wind\_data;

double[] loc = { 50.800314, -1.289956 };

if (IsConnectedToInternet())

{

wind\_data = await WindAPIProcessor.LoadWindData(loc);

}

else

{

// No network connection:

wind\_data = new WindModel

{

wind\_bearing = 90,

wind\_speed = "0"

};

}

windArrow.Rotation = wind\_data.wind\_bearing;

windSpeedIndicator.SetMessage = $"Wind Speed: {wind\_data.wind\_speed}Kts";

}

public static bool IsConnectedToInternet()

{

try

{

using (var client = new WebClient())

using (client.OpenRead("http://google.com/generate\_204"))

return true;

}

catch

{

return false;

}

}

public void GenerateMap()

{

GetFullMap.SetScalar = 0.0375956;

GetFullMap.SetScalar = 0.0190481;

for (int i = 1; i <= 2; i++)

{

//Find a better method to check if image exists

try

{

new BitmapImage(new Uri($"pack://application:,,,/Images/Map{i}.JPG"));

new MapSegmentObject(i, GetFullMap);

}

catch (IOException)

{

throw new Exception("Map Image file not found");

}

}

GetFullMap.SetVisiblePos(MyCanvas, -1);

windArrow = new WindArrow(MyCanvas);

windSpeedIndicator = new TextBlockObject(0, 0, "Wind Speed: 0 Kts", MyCanvas, 22, 1);

routeDistanceIndicator = new TextBlockObject(1070, 425, " - Route Distance = 0Nm", InfoPanel, 12, 0);

routeTimeIndicator = new TextBlockObject(1070, 450, " - Route Time = 0", InfoPanel, 12, 0);

clockDisplay = new TextBlockObject(990, 0, "", MyCanvas, 22, 1);

}

internal FullMapObject GetFullMap { get; } = new FullMapObject();

private void UpdateClock()

{

string time = DateTime.Now.ToString("T");

clockDisplay.SetMessage = time;

}

private void MainLoop(object sender, EventArgs e)

{

if (first\_run)

{

// Add anything here that can't be run in the constructor but has to be run on launch

first\_run = !first\_run;

DPI = GetScreenDPI(MyCanvas);

GetFullMap.VisibleSegment().CheckForMapFile();

LoadingCompass.Visibility = Visibility.Collapsed;

LoadingRectangle.Visibility = Visibility.Collapsed;

LoadingText.Visibility = Visibility.Collapsed;

}

if (InputValidator.TextInputCheck(WindAngleInput, wind\_angle))

{

ChangeWindAngle();

}

if (InputValidator.TextInputCheck(MaxHullSpeedInput, max\_speed))

{

ChangeMaxSpeed();

}

if (waypointTimeIndicator != null)

{

if (MouseOverWaypoint() == default)

{

waypointTimeIndicator.SetVisible(false, MyCanvas);

waypointTimeIndicator = null;

}

}

UpdateClock();

}

#region UserInputs

/// <summary>

/// Returns the index of the waypoint that the mouse is over if any

/// </summary>

/// <returns></returns>

private int MouseOverWaypoint()

{

Point point = Mouse.GetPosition(MyCanvas);

MapSegmentObject segment = GetFullMap.VisibleSegment();

for (int i = 0; i < segment.GetWaypointsAndLines().Count; i += 2)

{

MainObject waypoint = segment.GetWaypointsAndLines()[i];

if (Hypotenuse(point.X - waypoint.GetLeft - WAYPOINT\_RADIUS, point.Y - waypoint.GetTop - WAYPOINT\_RADIUS) < WAYPOINT\_RADIUS)

{

return i;

}

}

return default;

}

/// <summary>

/// Sums the total length of all route lines

/// </summary>

/// <param name="final\_index"></param>

/// <returns></returns>

private double TotalRouteLineLength(int final\_index)

{

double route\_line\_length = 0;

for (int i = final\_index; i > 0; i -= 2)

{

route\_line\_length += ((LineObject)GetFullMap.VisibleSegment().GetWaypointsAndLines()[i]).RouteLineLength();

}

// Converts from DIPs to Nm

route\_line\_length \*= GetFullMap.VisibleSegment().GetScalar;

return route\_line\_length;

}

/// <summary>

/// Display the time when you should reach this waypoint

/// </summary>

/// <param name="index"></param>

private void DisplayWaypointData(int index)

{

MainObject waypoint = GetFullMap.VisibleSegment().GetWaypointsAndLines()[index];

double route\_time\_mins = 60 \* TotalRouteLineLength(index - 1) / max\_speed;

DateTime waypoint\_arrival\_time = plot\_time.AddMinutes(route\_time\_mins);

string arrival\_time\_str = waypoint\_arrival\_time.ToString("t");

waypointTimeIndicator = new TextBlockObject((int)waypoint.GetLeft - 10, (int)waypoint.GetTop - 10, $"Arrival Time = {arrival\_time\_str}", MyCanvas, 12, 1);

waypointTimeIndicator.SetBackground(Brushes.White);

}

private void ChangeMaxSpeed()

{

string input = MaxHullSpeedInput.Text;

if (InputValidator.NumValidate(input))

{

max\_speed = Convert.ToDouble(input);

InputStatusMessageTextBlock.Foreground = Brushes.Black;

InputStatusMessageTextBlock.Text = "Hull Speed Accepted";

return;

}

InputStatusMessageTextBlock.Foreground = Brushes.Red;

InputStatusMessageTextBlock.Text = "Hull Speed Invalid";

}

private void ChangeWindAngle()

{

// WindAngle = How close the boat can point to the wind (Default = 40 degrees either side)

string input = WindAngleInput.Text;

if (InputValidator.NumValidate(input))

{

double new\_angle = Convert.ToDouble(input);

if (new\_angle < 45 && new\_angle >= 0)

{

GetFullMap.VisibleSegment().GetShip.GetBoatToWind = wind\_angle;

wind\_angle = new\_angle;

InputStatusMessageTextBlock.Foreground = Brushes.Black;

InputStatusMessageTextBlock.Text = "Wind Angle Accepted";

return;

}

}

InputStatusMessageTextBlock.Foreground = Brushes.Red;

InputStatusMessageTextBlock.Text = "Invalid Wind Angle";

}

private void KeyIsUp(object sender, KeyEventArgs e)

{

if (e.Key == Key.D)

{

GetFullMap.SetVisiblePos(MyCanvas, 1);

if (!GetFullMap.VisibleSegment().TerrainMapFileExists)

{

// Forces a UI refresh which is necesarry to build a new terrain map file if required

MyCanvas.Dispatcher.Invoke(DispatcherPriority.ApplicationIdle, (NoArgDelegate)delegate { });

GetFullMap.VisibleSegment().CheckForMapFile();

}

}

if (e.Key == Key.A)

{

GetFullMap.SetVisiblePos(MyCanvas, -1);

}

if (e.Key == Key.F)

{

int waypoint\_index = MouseOverWaypoint();

if (waypoint\_index != default)

{

DisplayWaypointData(MouseOverWaypoint());

}

}

}

private void KeyIsDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.LeftShift)

{

if (Mouse.LeftButton == MouseButtonState.Pressed) // Add Tide Point

{

NewTidalPointWindow newTidalPointWindow = new NewTidalPointWindow(Mouse.GetPosition(MyCanvas));

newTidalPointWindow.Show();

}

if (Mouse.RightButton == MouseButtonState.Pressed) // Delete Tide Point

{

Point mousePos = Mouse.GetPosition(MyCanvas);

List<TidalFlowPoint> tidePoints = GetFullMap.VisibleSegment().GetTidalFlowPoints;

foreach (TidalFlowPoint tidePoint in tidePoints)

{

if (Hypotenuse(tidePoint.GetLeft - mousePos.X, tidePoint.GetTop - mousePos.Y) <= 25)

{

GetFullMap.VisibleSegment().DelTidePoint(tidePoints.IndexOf(tidePoint), MyCanvas);

return;

}

}

}

}

}

private void OnPlot(object sender, RoutedEventArgs e)

{

plot\_time = DateTime.Now;

MapSegmentObject visible\_segment = GetFullMap.VisibleSegment();

if (visible\_segment.GetWaypointsAndLines().Count >= 3)

{

// Removes existing route lines

visible\_segment.DelRouteLines(MyCanvas);

// Plot the new route

TackingRoutePlotter tacking\_route = new TackingRoutePlotter(visible\_segment, this, max\_speed);

routeDistanceIndicator.SetMessage = $" - Route Distance = {Math.Round(tacking\_route.RouteDistance, 2)}Nm";

routeTimeIndicator.SetMessage = $" - Route Time = {tacking\_route.RouteTime}";

}

}

private async void OnOptimise(object sender, RoutedEventArgs e)

{

await OptimiseStart(this); // Asynchronous

}

private void OnReset(object sender, RoutedEventArgs e)

{

Reset();

}

/// <summary>

/// Deletes all waypoints and tide points on all map segements

/// </summary>

public void Reset()

{

foreach (MapSegmentObject segment in GetFullMap.GetMapSegmentArr())

{

if (segment != null)

{

segment.DelRouteLines(MyCanvas);

for (int i = 0; i < segment.GetWaypointsAndLines().Count;)

{

segment.DelWaypointOrLine(i, MyCanvas);

}

for (int i = 0; i < segment.GetTidalFlowPoints.Count; i++)

{

segment.DelTidePoint(i, MyCanvas);

}

}

}

}

private void OnHelp(object sender, RoutedEventArgs e)

{

HelpWindow help\_window = new HelpWindow();

help\_window.Show();

}

private void LeftMouseIsUp(object sender, MouseButtonEventArgs e)

{

double[] coords = { e.GetPosition(MyCanvas).X, e.GetPosition(MyCanvas).Y };

if (coords[0] < 1070 && coords[1] < HEIGHT && !Keyboard.IsKeyDown(Key.LeftShift)) // Add waypoint

{

PlaceWaypoint(coords);

}

}

private void RightMouseIsUp(object sender, MouseButtonEventArgs e)

{

Point point = e.GetPosition(MyCanvas);

for (int i = 0; i < GetFullMap.VisibleSegment().GetWaypointsAndLines().Count(); i += 2) // Delete Waypoint

{

MainObject waypoint = GetFullMap.VisibleSegment().GetWaypointsAndLines()[i];

if (Hypotenuse(waypoint.GetLeft + WAYPOINT\_RADIUS - point.X, waypoint.GetTop + WAYPOINT\_RADIUS - point.Y) <= WAYPOINT\_RADIUS)

{

RemoveWaypoint(i);

}

}

}

private void OnSaveLoad(object sender, RoutedEventArgs e)

{

SaveLoadWindow saveLoad = new SaveLoadWindow();

saveLoad.Show();

}

#endregion

#region Waypoints

public void RemoveWaypoint(int index)

{

//Find a better way of doing this

MapSegmentObject segment = GetFullMap.VisibleSegment();

segment.DelWaypointOrLine(index, MyCanvas);

int Count = segment.GetWaypointsAndLines().Count();

if (Count != 0)

{

if (index == Count)

{

// If its the lastt waypoint then only delete the line before

segment.DelWaypointOrLine(index - 1, MyCanvas);

}

else if (index == 0)

{

// If its the first waypoint then only delete the line after

segment.DelWaypointOrLine(index, MyCanvas);

}

else

{

// If its not the first or last waypoint then delete the line either side of the waypoint and place a line between the two waypoints either side

segment.DelWaypointOrLine(index - 1, MyCanvas);

segment.DelWaypointOrLine(index - 1, MyCanvas);

PlaceWaypointLine(index - 1, segment.GetWaypointsAndLines()[index - 1], segment.GetWaypointsAndLines()[index - 2]);

}

}

}

public static double Hypotenuse(double num1, double num2)

{

return Math.Sqrt(num1 \* num1 + num2 \* num2);

}

public void PlaceWaypoint(double[] coords)

{

if (PixelIsLand((int)Math.Round(coords[1]), (int)Math.Round(coords[0])) == false) // If not land

{

Waypoint new\_waypoint = new Waypoint(MyCanvas, coords[0] - WAYPOINT\_RADIUS, coords[1] - WAYPOINT\_RADIUS);

List<MainObject> waypoints = GetFullMap.VisibleSegment().GetWaypointsAndLines();

if (waypoints.Count > 0)

{

MainObject old\_waypoint = waypoints[waypoints.Count - 1];

PlaceWaypointLine(GetFullMap.VisibleSegment().GetWaypointsAndLines().Count(), old\_waypoint, new\_waypoint); // Add connector line

}

GetFullMap.VisibleSegment().AddWaypointOrLine(GetFullMap.VisibleSegment().GetWaypointsAndLines().Count(), new\_waypoint);

}

}

private void PlaceWaypointLine(int index, MainObject object1, MainObject object2)

{

double[] LinePos = { object1.GetLeft + WAYPOINT\_RADIUS, object1.GetTop + WAYPOINT\_RADIUS, object2.GetLeft + WAYPOINT\_RADIUS, object2.GetTop + WAYPOINT\_RADIUS };

GetFullMap.VisibleSegment().AddWaypointOrLine(index, new LineObject(MyCanvas, LinePos, Brushes.Black));

}

private async Task OptimiseStart(MainWindow mainWindow)

{

// Generating the route on a seperate thread so the program doesnt freeze up

Queue<ShortestRouteObject> shortestRoutes = new Queue<ShortestRouteObject>();

List<MainObject> lines = GetFullMap.VisibleSegment().GetWaypointsAndLines();

for (int i = 1; i < lines.Count; i += 2)

{

//LinePositions.Enqueue(((LineObject)lines[i]).LinePos);

shortestRoutes.Enqueue(await Task.Run(() => Optimise(mainWindow, ((LineObject)lines[i]).LinePos)));

}

// The new waypoints have to be placed on the main thread because the UI can only be accessed on one thread

PlaceGeneratedWaypoints(shortestRoutes, mainWindow);

}

private ShortestRouteObject Optimise(MainWindow main\_window, double[] line\_pos)

{

ShortestRouteObject short\_route = new ShortestRouteObject(line\_pos, NODE\_SEPERATION, main\_window);

return short\_route;

}

private static void PlaceGeneratedWaypoints(Queue<ShortestRouteObject> shortestRoutes, MainWindow main\_window)

{

for (int i = 0; i < 2 \* shortestRoutes.Count; i += 2) // Remove user placed waypoints

{

main\_window.RemoveWaypoint(2);

}

while (shortestRoutes.Count != 0) // Add Generated waypoints in order from the queue

{

ShortestRouteObject short\_route = shortestRoutes.Dequeue();

foreach (GridNode node in short\_route.GetRouteCoords)

{

main\_window.PlaceWaypoint(new double[] { node.X, node.Y });

}

}

}

#endregion

#region LandDetection

public static double[] CalculateLineEquation(double[] line\_pos)

{

// Gradient = Change in Y / Change in X

double grad\_to\_node = (line\_pos[1] - line\_pos[3]) / (line\_pos[0] - line\_pos[2]);

double X\_dist\_to\_node = line\_pos[0] - line\_pos[2];

double Y\_intercept = grad\_to\_node \* -line\_pos[0] + line\_pos[1];

// If line is near-vertical Y\_intercept will default to infinity

// This ensures that it can never be infinity

if (double.IsPositiveInfinity(Y\_intercept))

{

Y\_intercept = 99999;

}

if (double.IsNegativeInfinity(Y\_intercept))

{

Y\_intercept = -99999;

}

double step\_in\_X = LINE\_STEP\_DIST / Math.Sqrt(1 + (grad\_to\_node \* grad\_to\_node));

if (X\_dist\_to\_node > 0)

{

step\_in\_X \*= -1;

}

return new double[] { step\_in\_X, grad\_to\_node, Y\_intercept };

}

public bool LineIntersectsLand(double[] line\_pos)

{

// 0 = Step, 1 = Gradient to node, 2 = Y-intercept

double[] line\_info = CalculateLineEquation(line\_pos);

int LandPixelCount = 0;

double Y;

for (double X = line\_pos[0]; X > line\_pos[2] + 1 || X < line\_pos[2] - 1; X += line\_info[0])

{

// Y = mX + c

Y = line\_info[1] \* X + line\_info[2];

if (PixelIsLand((int)Y, (int)X))

{

LandPixelCount++;

if (LandPixelCount > 10)

{

return true;

}

}

}

return false;

}

/// <summary>

/// Queries the Terrain map array to see if a specified location contains land or sea

/// </summary>

/// <param name="row"></param>

/// <param name="col"></param>

/// <returns></returns>

public bool PixelIsLand(int row, int col)

{

bool[,] terrain\_pixel\_map = GetFullMap.VisibleSegment().GetTerrainPixelMap;

row /= (int)NODE\_SEPERATION;

col /= (int)NODE\_SEPERATION;

if (row > terrain\_pixel\_map.GetLength(0) - 1 || col > terrain\_pixel\_map.GetLength(1) - 1 || row < 0 + 1 || col < 0 + 1)

{

return true;

}

else

{

try

{

return terrain\_pixel\_map[row, col];

}

catch (IndexOutOfRangeException)

{

throw new Exception($"Land Detection out of range, coords: ({col \* NODE\_SEPERATION}, {row \* NODE\_SEPERATION})");

}

}

}

/// <summary>

/// Gets whether a pixel is land from the image itself

/// </summary>

/// <param name="pixel\_row"></param>

/// <param name="pixel\_col"></param>

/// <returns></returns>

public bool PixelIsLandFromVisual(int pixel\_row, int pixel\_col)

{

MapSegmentObject Segment = GetFullMap.VisibleSegment();

double[] segment\_dimentions = { Segment.GetHeight, Segment.GetWidth };

Color colour = GetPixelColor(Segment.GetRectangle, pixel\_row, pixel\_col, segment\_dimentions);

if (colour.R <= 225 && colour.R >= 185 && colour.G <= 180 && colour.G >= 145 && colour.B <= 255 && colour.B >= 235)

{

return false;

}

else

{

return true;

}

}

/// <summary>

/// Gets the pixel colour at a specified place on the screen

/// </summary>

/// <param name="visual"></param>

/// <param name="row"></param>

/// <param name="col"></param>

/// <param name="segmentDimentions"></param>

/// <returns></returns>

public Color GetPixelColor(Visual visual, int row, int col, double[] segmentDimentions)

{

GC.WaitForPendingFinalizers();

// Viewbox uses values between 0 & 1 so normalize the Rect with respect to the canvas Height & Width

Rect percentSrceenRec = new Rect(col / segmentDimentions[1], row / segmentDimentions[0],

1 / segmentDimentions[1], 1 / segmentDimentions[0]);

var bmpOut = new RenderTargetBitmap((int)(DPI.X / 96.0),

(int)(DPI.Y / 96.0),

DPI.X, DPI.Y, PixelFormats.Default); // generalized for monitors with different dpi

DrawingVisual drawingVisual = new DrawingVisual();

using (DrawingContext dc = drawingVisual.RenderOpen())

{

// Creates the rectangle without drawing it

dc.DrawRectangle(new VisualBrush { Visual = visual, Viewbox = percentSrceenRec }, null, new Rect(0, 0, 1.0, 1.0));

}

bmpOut.Render(drawingVisual);

var bytes = new byte[4];

int iStride = 4; // = 4 \* bmpOut.Width (for 32 bit graphics with 4 bytes per pixel - 4 \* 8 bits = 32)

bmpOut.CopyPixels(bytes, iStride, 0);

return Color.FromArgb(bytes[0], bytes[1], bytes[2], bytes[3]);

}

public static Point GetScreenDPI(Visual visual)

{

PresentationSource source = PresentationSource.FromVisual(visual);

Point Dpi;

Dpi = new Point(96.0 \* source.CompositionTarget.TransformToDevice.M11, 96.0 \* source.CompositionTarget.TransformToDevice.M22);

return Dpi;

}

#endregion

}

}

### APIManager.cs

using System.Net.Http;

using System.Net.Http.Headers;

namespace Optimal\_Route\_Calculator

{

static class APIManager

{

public static HttpClient ApiClient { get; set; }

public static void InitialiseClient()

{

// Sets up the Http Client and clears existing headers

ApiClient = new HttpClient();

ResetClient();

}

public static void ResetClient()

{

// Clears existing headers and adds JSON as a media return type

ApiClient.DefaultRequestHeaders.Accept.Clear();

ApiClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

}

}

### Arrow.cs

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Both the Wind arrow and the Tidal Points require an arrow

/// They should inherit from this class

/// </summary>

public class Arrow : MainObject

{

protected ImageBrush Skin = new ImageBrush();

protected BitmapImage bitmapImage;

protected string uri;

protected RotateTransform rotate = new RotateTransform();

protected double rotateAngle;

public virtual double Rotation

{

set

{

// Rotations are zeroed at the +ve x-axis so add 90

rotate.Angle = AngleAddition(value, -90) - rotateAngle;

rotateAngle += rotate.Angle;

shape.RenderTransform = rotate;

}

get { return rotateAngle; }

}

}

}

### FileHandler.cs

using System;

using System.IO;

namespace Optimal\_Route\_Calculator

{

abstract class FileHandler

{

/// <summary>

/// Gets the full path of the file specified from the localPath parameter

/// The file code parameter determines wether the file is a text file or not so it knows whether to check for .txt or not

/// </summary>

/// <param name="localPath"></param>

/// <param name="fileCode">If its a text file then this should be 0</param>

/// <returns></returns>

public static string PathToAppDirectory(string localPath, int fileCode)

{

// Gets the current directory that the program is stored in

string currentDir = Environment.CurrentDirectory;

// Gets the stored infomation about that directory

DirectoryInfo directory = new DirectoryInfo(Path.GetFullPath(Path.Combine(currentDir, @"..\..\" + localPath)));

// If the localPath needs to lead to a text file and the suggested local path doesnt have a .txt extension then its added

if (!(currentDir = directory.ToString()).EndsWith(".txt") && fileCode == 0)

{

return currentDir += ".txt";

}

return currentDir;

}

}

}

### FullMapObject.cs

using System.Collections.Generic;

using System.Windows.Controls;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Acts as a collection for all the map segments

/// </summary>

class FullMapObject

{

private readonly MapSegmentObject[] map\_segment\_arr = new MapSegmentObject[5];

const int MAX\_MAP\_SEGMENTS = 5;

public FullMapObject()

{

}

public double SetScalar

{

set { GetScalar.Add(value); }

}

public List<double> GetScalar { get; } = new List<double>();

/// <summary>

/// Changes the currently displayed map segment

/// </summary>

/// <param name="MyCanvas"></param>

/// <param name="index"></param>

public void SetVisiblePos(Canvas MyCanvas, int index)

{

if (GetVisibleSegmentIndex + index > -1 && GetVisibleSegmentIndex + index < MAX\_MAP\_SEGMENTS)

{

if (map\_segment\_arr[GetVisibleSegmentIndex + index] != null)

{

GetVisibleSegmentIndex += index;

// Changes the visibility of next map segment and previous map segment + all its contained objects

MapSegmentObject visibleMapSegment = map\_segment\_arr[GetVisibleSegmentIndex];

MapSegmentObject oldVisibleMapSegment = map\_segment\_arr[GetVisibleSegmentIndex - index];

visibleMapSegment.SetVisible(true, MyCanvas);

oldVisibleMapSegment.SetVisible(false, MyCanvas);

}

}

}

public int GetVisibleSegmentIndex { get; set; } = 2;

/// <summary>

/// Returns the Currently displayed segment

/// </summary>

/// <returns>Displayed Map Segment</returns>

public MapSegmentObject VisibleSegment()

{

return map\_segment\_arr[GetVisibleSegmentIndex];

}

public MapSegmentObject[] GetMapSegmentArr()

{

return map\_segment\_arr;

}

public void SetMapSegmentArr(int index, MapSegmentObject mapSegment)

{

map\_segment\_arr[index] = mapSegment;

}

}

}

### MapSegmentObject.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// The class that defines the areas of map

/// It contains all of the objects on that specific map

/// When the visible map is changed this class will handle changing the visibility of its objects

/// </summary>

class MapSegmentObject : MainObject

{

private readonly ImageBrush Skin = new ImageBrush();

private readonly BitmapImage bitmapImage;

private bool[,] TerrainPixelMap;

private readonly List<MainObject> waypointsLines = new List<MainObject>(); // Waypoint -- Line -- Waypoint -- Line -- Waypoint

private readonly string uri;

private const int NODE\_SEPERATION = 4;

public string HighWater { get; set; } = "";

public MapSegmentObject(int index, FullMapObject fullMap)

{

// Get the map image URI

uri = ($"pack://application:,,,/Images/Map{index}.JPG");

// Set the bitmapImage of the map with the source URI

bitmapImage = new BitmapImage(new Uri(uri));

// Add the bitmap as the fill of the rectangle for the map

Skin.ImageSource = bitmapImage;

GetHeight = bitmapImage.Height;

GetWidth = bitmapImage.Width;

shape = new Rectangle { Width = GetWidth, Height = GetHeight, Fill = Skin };

map\_segment\_index = index;

GetScalar = fullMap.GetScalar[index - 1];

Canvas.SetLeft(shape, GetLeft);

Canvas.SetTop(shape, GetTop);

// Makes sure that the map will always be behind any objects placed onto it

Panel.SetZIndex(shape, -1);

fullMap.SetMapSegmentArr(index, this);

}

/// <summary>

/// Iterates through the tide points and returns the tide point thats nearest to the specified position and less than 400 DIPs away

/// </summary>

public TidalFlowPoint NearestTidePoint(Point pos)

{

double dist = 400;

double hypot;

TidalFlowPoint closest\_point = null;

foreach (TidalFlowPoint tide\_point in GetTidalFlowPoints)

{

hypot = MainWindow.Hypotenuse(tide\_point.GetLeft - pos.X, tide\_point.GetTop - pos.Y);

if (hypot < dist)

{

dist = hypot;

closest\_point = tide\_point;

}

}

return closest\_point;

}

public double GetBoatSpeedAt(Point pos, double inbound\_bearing)

{

TidalFlowPoint closest\_point = NearestTidePoint(pos);

if (closest\_point == null)

{

return GetShip.GetMaxSpeed;

}

//Calculate time until the next high water which will be used to offset the cos graph in order to calculate the tidal flow now

closest\_point.CalculateTimeOffset(HighWater);

// u^2 = v^2 + r^2 - 2vr cos(theta) -- Consine rule, rearrange to equal 0;

// v^2 + 2r sin(theta)v + r^2 - u^2 = 0 -- Sub coefficients into quadratic formula

// rcos(theta) +- sqrt((-2rcos(theta))^2 - 4r^2 + 4u^2) = v

// Where r = tidal velocity,

// u = initial velocity,

// v = final velocity,

// theta = angle between initial velocity vector and tidal velocity vector

// Calculate Flow at time = 0, Now

double r = closest\_point.CalculateFlow(0);

double u = GetShip.GetMaxSpeed;

double tide\_bearing = BearingNormaliser(closest\_point.Bearing);

double ship\_bearing = BearingNormaliser(GetShip.AngleAddition(inbound\_bearing \* 180 / Math.PI, 90)) \* Math.PI / 180;

double theta = tide\_bearing - ship\_bearing;

// Math.Abs ensures the equation always returns the positive root

double v = Math.Abs(r \* Math.Sin(theta) - Math.Sqrt(Math.Pow(2 \* r \* Math.Sin(theta), 2) - 4 \* (r \* r - u \* u)) / 2);

return v;

}

/// <summary>

/// Converts a North-0 based bearing to a angle from the X-axis

/// </summary>

/// <param name="bearing"></param>

/// <returns></returns>

public double BearingNormaliser(double bearing)

{

// TODO: This process can be optmised most likely using MOD

if (bearing > 3 \* Math.PI / 2)

{

bearing = 2 \* Math.PI - bearing;

}

else if (bearing > Math.PI)

{

bearing = -(bearing - Math.PI);

}

else if (bearing > Math.PI / 2)

{

bearing -= Math.PI / 2;

}

else

{

bearing = -bearing;

}

return bearing;

}

public List<TidalFlowPoint> GetTidalFlowPoints { get; } = new List<TidalFlowPoint>();

public void AddTidePoint(Point pos, Canvas MyCanvas, double Bearing, double Max, double Min)

{

GetTidalFlowPoints.Add(new TidalFlowPoint(MyCanvas, pos.X, pos.Y, Bearing, Max, Min, HighWater));

}

public void DelTidePoint(int index, Canvas MyCanvas)

{

// Removes the specified tide point

GetTidalFlowPoints[index].SetVisible(false, MyCanvas);

GetTidalFlowPoints.RemoveAt(index);

}

public bool[,] GetTerrainPixelMap

{

get { return TerrainPixelMap; }

}

/// <summary>

/// This method will only be called on the first launch of the program, it generates the terrain map 2D array

/// It will iterate through the bitmap every 5 pixels getting the colour and returing true to the array if land is detected

/// This process can be slow so this tarrain map will be saved to file

/// </summary>

/// <param name="main\_window"></param>

public void GenerateTerrainMap(MainWindow main\_window)

{

double[] segment\_dimentions = { GetHeight, GetWidth };

bool[,] terrain\_map = new bool[(int)segment\_dimentions[0] / NODE\_SEPERATION + 1, (int)segment\_dimentions[1] / NODE\_SEPERATION + 1];

for (int i = 0; i < segment\_dimentions[0] / NODE\_SEPERATION; i++)

{

for (int f = 0; f < segment\_dimentions[1] / NODE\_SEPERATION; f++)

{

terrain\_map[i, f] = main\_window.PixelIsLandFromVisual(i \* NODE\_SEPERATION, f \* NODE\_SEPERATION);

}

}

TerrainPixelMap = terrain\_map;

}

public bool TerrainMapFileExists { get; set; } = false;

/// <summary>

/// Checks the if TerrainMapFile exists, if not a new one is generated

/// </summary>

public void CheckForMapFile()

{

// Checks if the terrain map already exists

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

string path = "TerrainMapFiles\\" + $"Map{map\_segment\_index}";

path = FileHandler.PathToAppDirectory(path, 0);

if (!File.Exists(path))

{

GenerateTerrainMap(main\_window);

GenerateMapFile(path);

TerrainMapFileExists = true;

}

else

{

LoadFromMapFile(path);

}

}

/// <summary>

/// Creates the terrain map file and writes the terrain map as 1s and 0s to that file

/// 1 = Land, 0 = Water

/// </summary>

/// <param name="path"></param>

private void GenerateMapFile(string path)

{

string line;

using (StreamWriter file = new StreamWriter(path))

{

for (int i = 0; i < TerrainPixelMap.GetLength(0); i++)

{

line = "";

for (int f = 0; f < TerrainPixelMap.GetLength(1); f++)

{

// If its land then place 1, if its sea then place a zero

line += TerrainPixelMap[i, f] == true ? "1" : "0";

}

file.WriteLine(line);

}

}

}

/// <summary>

/// Creates the terrain map file and writes the terrain map as 1s and 0s to that file

/// 1 = Land, 0 = Water

/// </summary>

/// <param name="path"></param>

private void GenerateMapFile(string path)

{

string line;

using (StreamWriter file = new StreamWriter(path))

{

for (int i = 0; i < TerrainPixelMap.GetLength(0); i++)

{

line = "";

for (int f = 0; f < TerrainPixelMap.GetLength(1); f++)

{

// If its land then place 1, if its sea then place a zero

line += TerrainPixelMap[i, f] == true ? "1" : "0";

}

file.WriteLine(line);

}

}

}

/// <summary>

/// Loads the terrain map from the terrain map file

/// </summary>

/// <param name="path"></param>

private void LoadFromMapFile(string path)

{

double[] segment\_dimentions = { GetHeight, GetWidth };

bool[,] terrain\_map = new bool[(int)segment\_dimentions[0] / NODE\_SEPERATION + 1, (int)segment\_dimentions[1] / NODE\_SEPERATION + 1];

string line;

int line\_num = 0;

int char\_num;

using (StreamReader file = new StreamReader(path))

{

// Iterates through the text file line by line

while ((line = file.ReadLine()) != null)

{

char\_num = 0;

// Iterates through each line

foreach (char character in line)

{

// If character == '1' return true

// Else return false

terrain\_map[line\_num, char\_num] = character == '1';

char\_num++;

}

line\_num++;

}

}

TerrainPixelMap = terrain\_map;

}

public List<MainObject> GetWaypointsAndLines()

{

return waypointsLines;

}

public void AddWaypointOrLine(int index, MainObject mainObject)

{

waypointsLines.Insert(index, mainObject);

}

public void DelWaypointOrLine(int index, Canvas MyCanvas)

{

if (index % 2 != 0 || (index == 0 && waypointsLines.Count % 2 == 0))

{

((LineObject)waypointsLines[index]).ChangeRouteLineVisibility(MyCanvas);

}

waypointsLines[index].SetVisible(false, MyCanvas);

waypointsLines.RemoveAt(index);

}

public void DelRouteLines(Canvas MyCanvas)

{

// If its a LineObject it will remove its RouteLines

for (int i = 0; i < waypointsLines.Count - 1; i++)

{

if (i % 2 != 0)

{

((LineObject)waypointsLines[i]).ChangeRouteLineVisibility(MyCanvas);

((LineObject)waypointsLines[i]).RemoveRouteLines();

}

}

}

public override void SetVisible(bool Visable, Canvas MyCanvas)

{

// Map segments needs to override this method because it needs to also change its objects visibility

visible = Visable;

if (visible)

{

MyCanvas.Children.Add(shape);

}

else

{

MyCanvas.Children.Remove(shape);

}

ChangeObjectVisibility(MyCanvas);

}

public void ChangeObjectVisibility(Canvas MyCanvas)

{

// Iterate through waypoints and lines and change visibilty

for (int i = 0; i < waypointsLines.Count(); i++)

{

MainObject mainObject = waypointsLines[i];

mainObject.SetVisible(!mainObject.GetVisible(), MyCanvas);

// If its a LineObject it will reverse the line's 'route lines' visibility

if (i % 2 != 0)

{

((LineObject)mainObject).ChangeRouteLineVisibility(MyCanvas);

}

}

foreach (TidalFlowPoint tidePoint in GetTidalFlowPoints)

{

tidePoint.SetVisible(!tidePoint.GetVisible(), MyCanvas);

}

}

public UIElement GetRectangle

{

get { return shape; }

}

public double GetHeight { get; }

public double GetWidth { get; }

public ShipObject GetShip { get; } = new ShipObject();

public double GetScalar { get; }

}

}

### GridNode.cs

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Grid nodes used in A\* pathfinding

/// </summary>

class GridNode

{

public double F\_Cost { get; set; }

public double G\_Cost { get; set; }

public double H\_Cost { get; set; }

public int Parent { get; set; }

public double X { get; set; }

public double Y { get; set; }

}

}

### InputValidator.cs

using System.Windows.Controls;

namespace Optimal\_Route\_Calculator

{

static class InputValidator

{

/// <summary>

/// Verifies that a text input will actually change the exsting value and that it contains a value

/// </summary>

/// <param name="textBox"></param>

/// <param name="existing\_value"></param>

/// <returns></returns>

public static bool TextInputCheck(TextBox textBox, double existing\_value)

{

if (textBox.Text != "" && textBox.Text != existing\_value.ToString())

{

return true;

}

return false;

}

public static bool NumValidate(string inp)

{

// Verifies that the input only consists of numbers

foreach (char character in inp)

{

// ASCII: 48 = '0', 57 = '9', 46 = '.'

if ((character < 48 || character > 57) && character != 46)

{

return false;

}

}

return true;

}

}

}

### LineObject.cs

using System.Collections.Generic;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace Optimal\_Route\_Calculator

{

class LineObject : MainObject

{

private double[] line\_pos = new double[4];

private double Length { get; }

public LineObject(Canvas MyCanvas, double[] LinePos, SolidColorBrush colour)

{

shape = new Line

{

X1 = line\_pos[0] = LinePos[0],

Y1 = line\_pos[1] = LinePos[1],

X2 = line\_pos[2] = LinePos[2],

Y2 = line\_pos[3] = LinePos[3],

Stroke = colour,

StrokeThickness = 2

};

Length = MainWindow.Hypotenuse(line\_pos[0] - line\_pos[2], line\_pos[1] - line\_pos[3]);

MyCanvas.Children.Add(shape);

}

public double RouteLineLength()

{

double total = 0;

foreach (LineObject line in GetRouteLines)

{

// Sums its own route lines if it has any

total += line.Length;

}

return total;

}

public double[] LinePos

{

get { return line\_pos; }

set { line\_pos = value; }

}

public List<LineObject> GetRouteLines { get; } = new List<LineObject>();

public void AddRouteLine(LineObject line)

{

GetRouteLines.Add(line);

}

/// <summary>

/// Changes its own route line visibility if it has any

/// </summary>

/// <param name="MyCanvas"></param>

public void ChangeRouteLineVisibility(Canvas MyCanvas)

{

foreach (LineObject line in GetRouteLines)

{

line.SetVisible(!line.GetVisible(), MyCanvas);

}

}

/// <summary>

/// removes its own route lines if it has any

/// </summary>

public void RemoveRouteLines()

{

GetRouteLines.Clear();

}

public Line GetShape

{

get { return (Line)shape; }

}

}

}

### MainObject.cs

using System;

using System.Windows;

using System.Windows.Controls;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// The parent class for any object that has a physical position

/// </summary>

public abstract class MainObject

{

protected UIElement shape;

protected bool visible = true;

protected double getLeft;

protected double getTop;

protected int map\_segment\_index;

public virtual double GetLeft

{

get { return getLeft; }

set { getLeft = value; }

}

public virtual double GetTop

{

get { return getTop; }

set { getTop = value; }

}

public virtual bool GetVisible()

{

return visible;

}

public virtual void SetVisible(bool Visable, Canvas MyCanvas)

{

visible = Visable;

if (visible)

{

MyCanvas.Children.Add(shape);

}

else

{

MyCanvas.Children.Remove(shape);

}

}

public virtual int GetMapSegmentIndex

{

get { return map\_segment\_index; }

set { map\_segment\_index = value; }

}

public virtual double AngleAddition(double angle1, double angle2)

{

// TODO: this can be done more efficeiently used MOD

// makes sure the angle can never exceed 360 or go below 0

if (angle1 + angle2 >= 360)

{

return (angle1 + angle2 - 360);

}

else if (angle1 + angle2 < 0)

{

return 360 - Math.Abs(angle1 + angle2);

}

else

{

return angle1 + angle2;

}

}

public virtual void DrawObject(FrameworkElement MyCanvas)

{

// Adds the UIElement to the FrameworkElement (usually the canvas) and sets its position

Canvas.SetLeft(shape, getLeft);

Canvas.SetTop(shape, getTop);

((Canvas)MyCanvas).Children.Add(shape);

}

}

}

### NewTidePointWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using OxyPlot;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Interaction logic for NewTidalPointWindow.xaml

/// </summary>

public partial class NewTidalPointWindow : Window

{

private string HighWater { get; set; }

Point tidal\_point\_loc;

private bool point\_placed = false;

public NewTidalPointWindow(Point mouse\_pos)

{

InitializeComponent();

APIManager.ResetClient();

TideAPICall();

tidal\_point\_loc = mouse\_pos;

}

private bool LocalInputValidation()

{

if (InputValidator.TextInputCheck(txtBoxMaxFlow, -1) && (InputValidator.TextInputCheck(txtBoxMinFlow, -1)) &&

InputValidator.TextInputCheck(txtBoxTideBearing, -1) && !point\_placed)

{

string min\_str = txtBoxMinFlow.Text;

string max\_str = txtBoxMaxFlow.Text;

string bearing\_str = txtBoxTideBearing.Text;

// Check all inputs are numbers

if (InputValidator.NumValidate(min\_str) && InputValidator.NumValidate(max\_str) && InputValidator.NumValidate(bearing\_str))

{

return true;

}

}

return false;

}

private void BtnNewTidePointClick(object sender, RoutedEventArgs e)

{

if (LocalInputValidation())

{

double min = Convert.ToDouble(txtBoxMinFlow.Text);

double max = Convert.ToDouble(txtBoxMaxFlow.Text);

double bearing = Convert.ToDouble(txtBoxTideBearing.Text);

if (min >= 0 && max >= 0 && bearing <= 360 && bearing >= 0 && max > min)

{

InputStatusMessageTextBlock.Foreground = Brushes.Black;

InputStatusMessageTextBlock.Text = "New Tide Point Created";

// Create the tideFlowModel and draw the graph

TidalFlowModel tidalFlowModel = new TidalFlowModel();

tidalFlowModel.SetMaxMin(max, min, HighWater);

TidalFlowPlotModel.Model = tidalFlowModel.TideModel;

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

main\_window.GetFullMap.VisibleSegment().AddTidePoint(tidal\_point\_loc, main\_window.MyCanvas, bearing, max, min);

point\_placed = true;

return;

}

}

InputStatusMessageTextBlock.Foreground = Brushes.Red;

InputStatusMessageTextBlock.Text = "Invalid Input";

}

private async void TideAPICall()

{

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

string existing\_high\_water = main\_window.GetFullMap.VisibleSegment().HighWater;

if(existing\_high\_water == "")

{

// Calls the Tide Height API with the station ID: 0065 - Portsmouth

var tide\_data = await TideAPIProcessor.LoadTideData("0065");

HighWater = tide\_data.HighWater;

main\_window.GetFullMap.VisibleSegment().HighWater = HighWater;

}

else

{

HighWater = existing\_high\_water;

}

}

private void BtnCloseWindow(object sender, RoutedEventArgs e)

{

Close();

}

}

}

### SaveLoadWindow.xaml.cs

using System;

using System.IO;

using System.Windows;

using System.Windows.Threading;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Interaction logic for SaveLoadWindow.xaml

/// </summary>

public partial class SaveLoadWindow : Window

{

private readonly DispatcherTimer saveLoadTimer = new DispatcherTimer();

private readonly string saveFilesPath;

private readonly string[] invalid\_fileNames = new string[] { "CON", "PRN", "AUX", "NUL", "COM", "LPT",

"<", "?", ">", "/", "|", "\*" };

public SaveLoadWindow()

{

InitializeComponent();

saveLoadTimer.Tick += mainLoop;

saveLoadTimer.Interval = TimeSpan.FromMilliseconds(20);

saveLoadTimer.Start();

saveFilesPath = FileHandler.PathToAppDirectory("SaveFiles", 1);

UpdateListBox();

}

private void mainLoop(object sender, EventArgs e)

{

}

private void UpdateListBox()

{

string saveFileName;

string[] existingSaveFiles = Directory.GetFiles(saveFilesPath);

foreach (string SaveFilePath in existingSaveFiles)

{

saveFileName = Path.GetFileName(SaveFilePath);

if (!lstBoxExistingFiles.Items.Contains(saveFileName))

{

lstBoxExistingFiles.Items.Add(saveFileName);

}

}

}

private void OnLoad(object sender, RoutedEventArgs e)

{

string path;

object saveFile = lstBoxExistingFiles.SelectedItem;

if (saveFile != null)

{

path = "SaveFiles\\" + saveFile.ToString();

path = FileHandler.PathToAppDirectory(path, 0);

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

main\_window.Reset();

using (StreamReader file = new StreamReader(path))

{

int line\_num = 0;

string line;

while ((line = file.ReadLine()) != null)

{

line\_num++;

// Splits the line into a (x, y) position of the waypoint

int comma\_index = line.IndexOf(',');

double[] coords = { Convert.ToDouble(line.Substring(0, comma\_index)) + 25, Convert.ToDouble(line.Substring(comma\_index + 1, line.Length - comma\_index - 1)) + 25 };

main\_window.PlaceWaypoint(coords);

}

}

StatusMsg.Text = "Waypoints Loaded";

}

else

{

StatusMsg.Text = "No File Selected";

}

}

private bool ValidFileName(string path)

{

foreach (string fileName in invalid\_fileNames)

{

if (path.Contains(fileName))

{

return false;

}

}

if (File.Exists(path))

{

return false;

}

return true;

}

private void OnSave(object sender, RoutedEventArgs e)

{

MainWindow main\_window = (MainWindow)Application.Current.MainWindow;

string path = "SaveFiles\\" + txtBoxFileName.Text;

path = FileHandler.PathToAppDirectory(path, 0);

if (ValidFileName(path))

{

using (StreamWriter file = new StreamWriter(path))

{

for (int i = 0; i < main\_window.GetFullMap.VisibleSegment().GetWaypointsAndLines().Count; i += 2)

{

MainObject waypoint = main\_window.GetFullMap.VisibleSegment().GetWaypointsAndLines()[i];

file.WriteLine($"{waypoint.GetLeft},{waypoint.GetTop}");

}

}

StatusMsg.Text = "Waypoints Saved";

UpdateListBox();

}

else

{

StatusMsg.Text = "File already exists or filename is invalid, try a new name";

}

}

}

}

### ShipObject.cs

namespace Optimal\_Route\_Calculator

{

class ShipObject : MainObject, IShipsandWaypoints

{

private readonly double[] windConeAngles = { 0, 0, 0 };

private double boat\_to\_wind = 40;

public ShipObject()

{

}

public double GetMaxSpeed { get; set; } = 6;

public double GetWindConeAngles(int getCode)

{

// getCode 1 == Get active cone angle

// getCode 2 == Get inactive cone angle

if (getCode == 1)

{

return windConeAngles[(int)windConeAngles[2]];

}

else

{

if (windConeAngles[2] == 0)

{

return windConeAngles[1];

}

else

{

return windConeAngles[0];

}

}

}

public void GenerateWindConeAngles(double wind\_angle)

{

// Gets the wind cone angles by adding 180 to the wind bearing and then adding and substracting the angle to wind

// This value can be set by the user but the default is 40 degrees

wind\_angle = AngleAddition(wind\_angle, 180);

windConeAngles[0] = AngleAddition(wind\_angle, boat\_to\_wind);

windConeAngles[1] = AngleAddition(wind\_angle, -boat\_to\_wind);

windConeAngles[2] = 0;

}

public bool CanSailTowards(double waypoint\_angle)

{

// Gets the difference in angle between the angle to waypoint and the wind cone angles

double angle\_diff = (AngleAddition(windConeAngles[1], boat\_to\_wind) - waypoint\_angle + 180 + 360) % 360 - 180;

// If the waypoint bearing is inside the wind cone the boat cannot sail towards the waypoint

if (angle\_diff <= boat\_to\_wind && angle\_diff >= -boat\_to\_wind)

{

return false;

}

else

{

return true;

}

}

public void ConeSwapSide()

{

// Change the active wind cone

if (windConeAngles[2] == 0)

{

windConeAngles[2] += 1;

}

else

{

windConeAngles[2] -= 1;

}

}

public double GetBoatToWind { set { boat\_to\_wind = value; } }

}

}

### ShipsandWaypoints.cs

namespace Optimal\_Route\_Calculator

{

interface IShipsandWaypoints

{

void ConeSwapSide();

}

}

### ShortestRouteObject.cs

using System;

using System.Collections.Generic;

using System.Windows;

namespace Optimal\_Route\_Calculator

{

class ShortestRouteObject

{

// A\* Pathfinding:

// G\_cost = Distance from starting node - following the route through parent nodes

// H\_cost = Straight distance from ending node

// F\_cost = G\_cost + H\_cost

private readonly List<GridNode> Route\_coords = new List<GridNode>();

private readonly GridNode start\_pos = new GridNode();

private readonly GridNode end\_pos = new GridNode();

private readonly double step;

public ShortestRouteObject(double[] linePos, double Step, MainWindow mainWindow)

{

start\_pos.X = linePos[0];

start\_pos.Y = linePos[1];

end\_pos.X = linePos[2];

end\_pos.Y = linePos[3];

step = Step;

GenerateRoute(mainWindow);

}

/// <summary>

/// 1. Runs A\* pathfinding from start\_pos to end\_pos with a step of 5 pixels between each node

/// 2. Removes all the unessescary nodes

/// </summary>

public void GenerateRoute(MainWindow main\_window)

{

// Node: 0 = X, 1 = Y, 2 = F\_cost, 3 = G\_cost, 4 = index of parent

List<GridNode> closed\_nodes = new List<GridNode>();

List<GridNode> open\_nodes = new List<GridNode> { start\_pos };

ChooseNodes(open\_nodes, closed\_nodes, main\_window);

// Follows the chain of parent indexes backwards from the last node to the start node to get the route

int node\_index = closed\_nodes.Count - 1;

if (node\_index != -1) // If a route can be plotted

{

while (node\_index != 0)

{

Route\_coords.Add(closed\_nodes[node\_index]);

node\_index = (int)closed\_nodes[node\_index].Parent;

}

Route\_coords.Reverse();

RemoveRedundantNodes(main\_window);

}

}

public void RemoveRedundantNodes(MainWindow main\_window)

{

// Deletes All nodes with line of sight on the node before, reducing the list of nodes whilst still following the land

for (int i = 0; i < Route\_coords.Count - 1; i++)

{

List<GridNode> nodes\_to\_remove = new List<GridNode>();

for (int f = i + 1; f < Route\_coords.Count - 1; f++)

{

double[] LinePos = { Route\_coords[i].X, Route\_coords[i].Y, Route\_coords[f].X, Route\_coords[f].Y };

if (!main\_window.LineIntersectsLand(LinePos))

{

nodes\_to\_remove.Add(Route\_coords[f]);

}

}

foreach (GridNode node in nodes\_to\_remove)

{

Route\_coords.Remove(node);

}

}

Route\_coords.RemoveAt(0);

// route\_coords.RemoveAt(route\_coords.Count - 1);

}

public void ChooseNodes(List<GridNode> openNodes, List<GridNode> closedNodes, MainWindow main\_window)

{

while (openNodes.Count > 0)

{

GridNode current\_node = PickCurrentNode(openNodes, closedNodes);

// If current\_node != end position then calculate the neighbors

if (!(current\_node.X <= end\_pos.X + step && current\_node.X >= end\_pos.X - step && current\_node.Y <= end\_pos.Y + step && current\_node.Y >= end\_pos.Y - step))

{

CheckNeighbor(step, 0, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(0, step, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(-step, 0, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(0, -step, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(step, step, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(-step, step, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(step, -step, current\_node, openNodes, closedNodes, main\_window);

CheckNeighbor(-step, -step, current\_node, openNodes, closedNodes, main\_window);

}

else

{

break;

}

}

}

public void CheckNeighbor(double x, double y, GridNode currentNode, List<GridNode> openNodes, List<GridNode> closedNodes, MainWindow main\_window)

{

GridNode neighbor = new GridNode()

{

X = currentNode.X + x,

Y = currentNode.Y + y,

F\_Cost = currentNode.F\_Cost,

G\_Cost = currentNode.G\_Cost + CalculateNodeWeight(x, y, main\_window, new Point(currentNode.X + x, currentNode.Y + y)),

Parent = closedNodes.IndexOf(currentNode)

};

// If neighbor is land or is already closed then return

if (main\_window.PixelIsLand((int)neighbor.Y, (int)neighbor.X) || GetNode(closedNodes, neighbor) != null)

{

return;

}

else

{

// If node is already in open nodes, the path from the shorter parent must be taken

double vel = main\_window.GetFullMap.VisibleSegment().GetShip.GetMaxSpeed;

GridNode node = GetNode(openNodes, neighbor);

double H\_cost = (Math.Abs(neighbor.X - end\_pos.X) + Math.Abs(neighbor.Y - end\_pos.Y)) / vel;

double G\_cost = neighbor.G\_Cost;

if (node != null)

{

if (neighbor.G\_Cost <= node.G\_Cost)

{

// If new path to neighbor is shorter then set the shorter path node as parent

node.G\_Cost = neighbor.G\_Cost;

node.F\_Cost = H\_cost + G\_cost;

node.Parent = neighbor.Parent;

}

}

else

{

neighbor.F\_Cost = H\_cost + G\_cost;

openNodes.Add(neighbor);

}

}

}

public double CalculateNodeWeight(double x, double y, MainWindow mainWindow, Point node\_loc)

{

double inboundBearing = Math.Atan2(y, x);

inboundBearing = (inboundBearing + (2 \* Math.PI)) % (2 \* Math.PI);

double boat\_velocity\_through\_node = mainWindow.GetFullMap.VisibleSegment().GetBoatSpeedAt(node\_loc, inboundBearing);

// If its a diagonal distance = 1.4 if not then distance = 1

if (Math.Abs(x) == Math.Abs(y))

{

return 1.4 / (boat\_velocity\_through\_node \* 1 / 2.4);

}

else

{

return 1 / (boat\_velocity\_through\_node \* 1.4 / 2.4);

}

}

public GridNode GetNode(List<GridNode> list, GridNode item)

{

// Checks if the node positions are the same

foreach (GridNode node in list)

{

if (node.X == item.X && node.Y == item.Y)

{

return node;

}

}

return null;

}

public GridNode PickCurrentNode(List<GridNode> openNodes, List<GridNode> closedNodes)

{

double max\_F\_cost = 10000;

int node\_index = 0;

// Looks for the node with the lowest F\_cost

foreach (GridNode node in openNodes)

{

if (node.F\_Cost < max\_F\_cost)

{

node\_index = openNodes.IndexOf(node);

max\_F\_cost = node.F\_Cost;

}

}

GridNode currentNode = openNodes[node\_index];

openNodes.RemoveAt(node\_index);

closedNodes.Add(currentNode);

return currentNode;

}

public List<GridNode> GetRouteCoords

{

get { return Route\_coords; }

}

}

}

### TackingRoutePlotter.cs

using System;

using System.Linq;

using System.Windows.Media;

namespace Optimal\_Route\_Calculator

{

class TackingRoutePlotter

{

const int WAYPOINT\_RADIUS = 25;

public double RouteDistance { get; set; }

public string RouteTime { get; set; }

public TackingRoutePlotter(MapSegmentObject visible\_segement, MainWindow mainWindow, double max\_speed)

{

GenerateRoute(visible\_segement, mainWindow, max\_speed);

}

private void GenerateRoute(MapSegmentObject visible\_segment, MainWindow mainWindow, double max\_speed)

{

// Get the first waypoint in the list

int end\_point\_index = visible\_segment.GetWaypointsAndLines().Count() - 1;

MainObject firstPoint = visible\_segment.GetWaypointsAndLines()[0];

// Get the starting coordinates of the ship, the first waypoint location + waypoint radius

// Placing it at the centre of the waypoint

visible\_segment.GetShip.GetTop = firstPoint.GetTop + WAYPOINT\_RADIUS;

visible\_segment.GetShip.GetLeft = firstPoint.GetLeft + WAYPOINT\_RADIUS;

// Only selects waypoints not lines

for (int i = 2; i <= end\_point\_index; i += 2)

{

// Get the next point in the list

MainObject nextPoint = visible\_segment.GetWaypointsAndLines()[i];

// Generate the tacking cone based off the bearing from the ship to the waypoint

((Waypoint)nextPoint).GenerateMaxTackCone(visible\_segment.GetShip.GetLeft, visible\_segment.GetShip.GetTop);

// Generate the ships wind cone bassed off the wind direction

visible\_segment.GetShip.GenerateWindConeAngles(mainWindow.windArrow.Rotation);

CalcNextRouteLine(nextPoint, visible\_segment, mainWindow, max\_speed);

}

RouteDistance = TotalRouteLineLength(visible\_segment.GetWaypointsAndLines().Count() - 2, mainWindow);

CalculateRouteTime(max\_speed);

}

private void CalculateRouteTime(double maxSpeed)

{

string time = "";

// Time = Distance / Speed

double route\_time\_hrs = RouteDistance / maxSpeed;

// Seperate hours

time += Math.Floor(route\_time\_hrs).ToString() + "hrs ";

// Seperate Minutes

double mins = (route\_time\_hrs - Math.Floor(route\_time\_hrs)) \* 60;

// Add the placeholder zero if mins is less than 10

time += mins < 10 ? "0" : "";

time += Math.Round(mins) + "mins";

RouteTime = time;

}

private void CalcNextRouteLine(MainObject next\_point, MapSegmentObject visible\_segment, MainWindow mainWindow, double max\_speed)

{

// Get the index of the next point in the list

int next\_point\_index = visible\_segment.GetWaypointsAndLines().IndexOf(next\_point);

// Get the ships position

double[] shipPos = { visible\_segment.GetShip.GetLeft, visible\_segment.GetShip.GetTop };

// Define a new array for the end of the next route line - This will become the boats new position

double[] nextLoc = { 0, 0 };

// Get the edge of the wind cone that the boat is currently travelling along

double active\_wind\_cone = visible\_segment.GetShip.GetWindConeAngles(1);

double[] tack\_cone = ((Waypoint)next\_point).GetMaxTackCone;

// Get the bearing to the waypoint using Atan2 which converts cartesian coords into a polar angle

double angle\_to\_waypoint = (180 / Math.PI) \* Math.Atan2(-visible\_segment.GetShip.GetTop + next\_point.GetTop + WAYPOINT\_RADIUS, -visible\_segment.GetShip.GetLeft + next\_point.GetLeft + WAYPOINT\_RADIUS);

// Convert polar angle into a bearing

angle\_to\_waypoint = (angle\_to\_waypoint + 360) % 360;

// If the ship can sail towards the waypoint then it will

if (visible\_segment.GetShip.CanSailTowards(angle\_to\_waypoint))

{

// Next location is the next waypoint + waypoint radius to set it in the middle of the waypoint

nextLoc[0] += next\_point.GetLeft + WAYPOINT\_RADIUS;

nextLoc[1] += next\_point.GetTop + WAYPOINT\_RADIUS;

// Creates a new route line with the next waypoints coordinates as the end location

double[] linePos = { shipPos[0], shipPos[1], nextLoc[0], nextLoc[1] };

PlaceRouteLine(linePos, next\_point\_index - 1, mainWindow);

// Set the ships location as the waypoints location

visible\_segment.GetShip.GetLeft = nextLoc[0];

visible\_segment.GetShip.GetTop = nextLoc[1];

}

else

{

// Gets the next waypoints coords

double[] waypointPos = { next\_point.GetLeft + WAYPOINT\_RADIUS, next\_point.GetTop + WAYPOINT\_RADIUS };

// Get the wind cone angle that the boat is not travelling along

double inactive\_wind\_cone = visible\_segment.AngleAddition(visible\_segment.GetShip.GetWindConeAngles(2), 180);

if (MainWindow.Hypotenuse(shipPos[0] - waypointPos[0], shipPos[1] - waypointPos[1]) > 1)

{

// Finds a potential end point of the next route line - this is where the edge of the wind cone intersects the edge of the tacking cone

nextLoc = CalcLineIntersection(tack\_cone[(int)tack\_cone[2]], active\_wind\_cone, shipPos, waypointPos);

if (MainWindow.Hypotenuse(nextLoc[0] - shipPos[0], nextLoc[1] - shipPos[1]) \* visible\_segment.GetScalar / max\_speed < 0.1666) // if route line is less than 10 mins

{

// Finds a potential end point of the next route line - this is where the edge of the wind cone intersects the other edge of the wind cone

nextLoc = CalcLineIntersection(inactive\_wind\_cone, active\_wind\_cone, shipPos, waypointPos);

}

double[] linePos = { shipPos[0], shipPos[1], nextLoc[0], nextLoc[1] };

// Places the route line, then adds the route line length to the total length

PlaceRouteLine(linePos, next\_point\_index - 1, mainWindow);

// Moves the ship to the end of the new Route Line

visible\_segment.GetShip.GetLeft = nextLoc[0];

visible\_segment.GetShip.GetTop = nextLoc[1];

// Allows the program to switch between each edge of the tack cone/wind cone

visible\_segment.GetShip.ConeSwapSide();

((Waypoint)next\_point).ConeSwapSide();

// Uses recursion to keep placing Lines until it can go straight to the waypoint

CalcNextRouteLine(next\_point, visible\_segment, mainWindow, max\_speed);

}

}

}

public double[] CalcLineIntersection(double tack\_cone\_angle, double wind\_cone\_angle, double[] ship\_pos, double[] waypoint\_pos)

{

// M = tan(theta (In radians))

double tack\_cone\_gradient = Math.Tan(tack\_cone\_angle \* (Math.PI / 180));

double wind\_cone\_gradient = Math.Tan(wind\_cone\_angle \* (Math.PI / 180));

// C = Gradient \* -Xcoord + Ycoord

double tack\_cone\_Yintercept = tack\_cone\_gradient \* -waypoint\_pos[0] + waypoint\_pos[1];

double wind\_cone\_Yintercept = wind\_cone\_gradient \* -ship\_pos[0] + ship\_pos[1];

// X intersection = Cone Y-intercept - Wind Y-Intercept / Wind Gradient - Cone Gradient

// Y intersection = Wind Gradient \* X intersection + Wind Y-intercept

double[] final\_coords = { 0, 0 };

final\_coords[0] += (tack\_cone\_Yintercept - wind\_cone\_Yintercept) / (wind\_cone\_gradient - tack\_cone\_gradient);

final\_coords[1] += wind\_cone\_gradient \* final\_coords[0] + wind\_cone\_Yintercept;

return final\_coords;

}

private double TotalRouteLineLength(int final\_index, MainWindow mainWindow)

{

FullMapObject fullMap = mainWindow.GetFullMap;

double route\_line\_length = 0;

for (int i = final\_index; i > 0; i -= 2)

{

// Sums the route lines length

route\_line\_length += ((LineObject)fullMap.VisibleSegment().GetWaypointsAndLines()[i]).RouteLineLength();

}

// Converts from DIPs to Nm

route\_line\_length \*= fullMap.VisibleSegment().GetScalar;

return route\_line\_length;

}

private void PlaceRouteLine(double[] line\_pos, int waypoint\_line\_index, MainWindow mainWindow)

{

// adds a new Route Line between the ship and the next location, new line is stored in a List, the list of route lines is a property of each waypoint line

((LineObject)mainWindow.GetFullMap.VisibleSegment().

GetWaypointsAndLines()[waypoint\_line\_index]).

AddRouteLine(new LineObject(mainWindow.MyCanvas, line\_pos, Brushes.Red));

}

}

}

### TextBlockObject.cs

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

namespace Optimal\_Route\_Calculator

{

class TextBlockObject : MainObject

{

private readonly string message;

public TextBlockObject(int x, int y, string msg, FrameworkElement element, int font\_size, int type\_code)

{

shape = new TextBlock { Text = msg, FontSize = font\_size, Style = (Style)Application.Current.Resources["CustomFont"], Margin = new Thickness(5, 5, 5, 5), FontWeight = FontWeights.Thin };

message = msg;

GetLeft = x;

GetTop = y;

// Type\_code == 0: Text block is added to a stack panel, Type\_code == 1: Text block is added to a defined location on the canvas

if (type\_code == 0)

{

DrawObject(element);

}

else

{

base.DrawObject(element);

}

}

public string SetMessage

{

get { return message; }

set { ((TextBlock)shape).Text = value; }

}

public void SetBackground(SolidColorBrush colour)

{

((TextBlock)shape).Background = colour;

}

public override void DrawObject(FrameworkElement stack\_panel)

{

((StackPanel)stack\_panel).Children.Add(shape);

}

}

}

### TidalFlowModel.cs

using OxyPlot;

using OxyPlot.Axes;

using OxyPlot.Series;

using System;

namespace Optimal\_Route\_Calculator

{

class TidalFlowModel

{

// Low tide to High tide in 6hrs 12.5mins

// 22,350 Seconds

// Frequency = 1 / Period

const double FREQ = 0.161030596;

public PlotModel TideModel { get; private set; }

public double[] max\_min = new double[2] { 0, 0 };

private string HighWaterStr { get; set; } = "01/01/0001 00:00:00";

private double HighWaterNum { get; set; }

public TidalFlowModel()

{

// Creates the OxyPlot model and sets the axis

TideModel = new PlotModel { Title = "Tidal Flow Model", TitleFontSize = 10 };

TideModel.Axes.Add(new LinearAxis { Position = AxisPosition.Left, AbsoluteMaximum = 5, MajorStep = 2, AbsoluteMinimum = 0, Title = "Velocity (Kts)" });

TideModel.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, AbsoluteMaximum = 10, MajorStep = 1, Title = "Time (Hrs)" });

TideModel.Series.Add(GetFunction());

}

public void SetMaxMin(double maximum, double minimum, string highWater)

{

// Re-Calculates the graph with new min, max values and resest axis

HighWaterStr = highWater;

CalculateTimeOffset();

max\_min[0] = maximum;

max\_min[1] = minimum;

TideModel.Axes[0].AbsoluteMaximum = maximum + 3;

TideModel.Series.Add(GetFunction());

}

private FunctionSeries GetFunction()

{

// Places a new point at specified intervals forming the graph

FunctionSeries fs = new FunctionSeries();

for (double i = 0; i < 10; i += 0.1)

{

DataPoint dp = new DataPoint(i, GetFlow(i));

fs.Points.Add(dp);

}

return fs;

}

private void CalculateTimeOffset()

{

DateTime high\_water\_time = Convert.ToDateTime(HighWaterStr);

TimeSpan time\_diff = high\_water\_time - DateTime.Now;

HighWaterNum = time\_diff.TotalHours;

}

private double GetFlow(double i)

{

// Y = Acos2(X \* pi \* Frequency) + min

double Amplitude = max\_min[0] - max\_min[1];

return Amplitude \* Math.Pow(Math.Cos((i - HighWaterNum) \* Math.PI \* FREQ), 2) + max\_min[1];

}

}

}

### TidalFlowPoint.cs

using System;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace Optimal\_Route\_Calculator

{

class TidalFlowPoint : Arrow

{

private const double ARROW\_WIDTH = 20;

private const double ARROW\_HEIGHT = 20;

private readonly double[] max\_min = new double[2];

private const double FREQ = 0.161030596;

private double high\_water\_diff;

private double bearing;

private readonly string high\_water;

private readonly Ellipse circle;

public TidalFlowPoint(Canvas MyCanvas, double SetLeft, double SetTop, double \_bearing, double max, double min, string highWaterTime)

{

circle = new Ellipse { Width = ARROW\_HEIGHT + 5, Height = ARROW\_HEIGHT + 5, Stroke = Brushes.Blue, Fill = Brushes.Transparent };

uri = ($"pack://application:,,,/Images/Arrow.png");

bitmapImage = new BitmapImage(new Uri(uri));

Skin.ImageSource = bitmapImage;

shape = new Rectangle { Width = ARROW\_WIDTH, Height = ARROW\_HEIGHT, Fill = Skin };

rotate.CenterX = GetLeft + 10;

rotate.CenterY = GetTop + 10;

// Sets high water time, bearing, min-max tide

high\_water = highWaterTime;

Bearing = \_bearing;

max\_min[0] = max;

max\_min[1] = min;

getLeft = SetLeft;

getTop = SetTop;

// Adds the arrow

Canvas.SetLeft(shape, GetLeft + 2.5);

Canvas.SetTop(shape, GetTop + 2.5);

MyCanvas.Children.Add(shape);

// Adds the circle

Canvas.SetLeft(circle, getLeft);

Canvas.SetTop(circle, getTop);

MyCanvas.Children.Add(circle);

Rotation = bearing;

}

public double Bearing

{

get

{

// 6hrs 12.5mins

// Gets the current tidal bearing which will flip 180 degrees every 6hrs 12.5mins

double time\_diff = Math.Abs((Convert.ToDateTime(high\_water) - DateTime.Now).TotalHours);

double cycle\_region = time\_diff % 12.4;

double deg\_bearing;

if (cycle\_region > 6.2)

{

deg\_bearing = AngleAddition(bearing, 180);

}

else

{

deg\_bearing = bearing;

}

// Converts to radians

return Math.PI / 180 \* deg\_bearing;

}

set { bearing = value; }

}

public void CalculateTimeOffset(string high\_water\_str)

{

// Calculates the time until next high water

DateTime high\_water\_time = Convert.ToDateTime(high\_water\_str);

TimeSpan time\_diff = high\_water\_time - DateTime.Now;

high\_water\_diff = time\_diff.TotalHours;

}

public double CalculateFlow(int t)

{

// Calculates the flow at a specified time

double Amplitude = max\_min[0] - max\_min[1];

return Amplitude \* Math.Pow(Math.Cos((t - high\_water\_diff) \* Math.PI \* FREQ), 2) + max\_min[1];

}

public override void SetVisible(bool Visable, Canvas MyCanvas)

{

visible = Visable;

if (visible)

{

MyCanvas.Children.Add(shape);

MyCanvas.Children.Add(circle);

}

else

{

MyCanvas.Children.Remove(shape);

MyCanvas.Children.Remove(circle);

}

}

}

}

### TideAPIProcessor.cs

using System;

using System.Net.Http;

using System.Threading.Tasks;

namespace Optimal\_Route\_Calculator

{

class TideAPIProcessor

{

public static async Task<TideHeightModel> LoadTideData(string stationId)

{

// Sets the URL for the API with a longitude and latitude

APIManager.ApiClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "d4b511b338194272bd9080044c97cbe2");

string url = $"https://admiraltyapi.azure-api.net/uktidalapi/api/V1/Stations/{stationId}/TidalEvents?";

// Sends the GET request to the specified URL - Asynchronous

using (HttpResponseMessage response = await APIManager.ApiClient.GetAsync(url))

{

// Checks if the request was sucessful before trying to access data

if (response.IsSuccessStatusCode)

{

//string test = await response.Content.ReadAsStringAsync();

// Assigns the response to an instance of the WindModel - Asynchronous

string json = await response.Content.ReadAsStringAsync();

TideHeightModel tidedata = new TideHeightModel(json);

return tidedata;

}

else

{

throw new Exception(response.ReasonPhrase);

}

}

}

}

}

### TideHeightModel.cs

using Newtonsoft.Json.Linq;

using System;

namespace Optimal\_Route\_Calculator

{

class TideHeightModel

{

public JArray JObjects { get; set; }

public string HighWater { get; private set; }

public TideHeightModel(string json)

{

JObjects = JArray.Parse(json);

GetNextHighWaterTime();

}

private void GetNextHighWaterTime()

{

// Searches through JSON objects and get the nearest High water time

foreach (JObject tide\_event in JObjects)

{

if (tide\_event.GetValue("EventType").ToString() == "HighWater")

{

string tmp\_high\_water = tide\_event.GetValue("DateTime").ToString();

if (Convert.ToDateTime(tmp\_high\_water) > DateTime.Now)

{

HighWater = tmp\_high\_water;

return;

}

}

}

}

}

}

### Waypoint.cs

using System;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace Optimal\_Route\_Calculator

{

class Waypoint : MainObject, IShipsandWaypoints

{

private readonly double Width = 50;

private readonly double Height = 50;

public Waypoint(Canvas MyCanvas, double SetLeft, double SetTop)

{

shape = new Ellipse { Width = Width, Height = Height, Stroke = Brushes.Red, Fill = Brushes.Transparent };

getLeft = SetLeft;

getTop = SetTop;

Canvas.SetLeft(shape, getLeft);

Canvas.SetTop(shape, getTop);

MyCanvas.Children.Add(shape);

}

public void GenerateMaxTackCone(double ship\_x, double ship\_y)

{

// Generate the tack cone angles using the angle to the ship

// tack\_cone\_centre = bearing from the waypoint to the ship

// using Atan2(x, y) I can convert from the cartesian coords of the ship to its polar angle from the waypoint

double tack\_cone\_centre = (180 / Math.PI) \* Math.Atan2(getTop - ship\_y + 25, getLeft - ship\_x + 25);

// Adds 180 degrees

tack\_cone\_centre = AngleAddition(tack\_cone\_centre, 180);

// Adds and subtracts 10 degree either side which creates a cone

GetMaxTackCone[0] = AngleAddition(tack\_cone\_centre, -10);

GetMaxTackCone[1] = AngleAddition(tack\_cone\_centre, 10);

GetMaxTackCone[2] = 0;

}

public double[] GetMaxTackCone { get; } = { 0, 0, 0 };

public void ConeSwapSide()

{

// Swaps the tack cone sides

if (GetMaxTackCone[2] == 0)

{

GetMaxTackCone[2] += 1;

}

else

{

GetMaxTackCone[2] -= 1;

}

}

}

}

### WindAPIProcessor.cs

using System;

using System.Net.Http;

using System.Threading.Tasks;

namespace Optimal\_Route\_Calculator

{

class WindAPIProcessor

{

public static async Task<WindModel> LoadWindData(double[] location)

{

// Sets the URL for the API with a longitude and latitude

string url = $"https://api.darksky.net/forecast/f9d408baa1fdc7017398cbd27acbee7d/{ location[0] }, { location[1] }";

// Sends the GET request to the specified URL - Asynchronous

using (HttpResponseMessage response = await APIManager.ApiClient.GetAsync(url))

{

// Checks if the request was sucessful before trying to access data

if (response.IsSuccessStatusCode)

{

// Assigns the response to an instance of the WindModel - Asynchronous

WindModel winddata = await response.Content.ReadAsAsync<WindModel>();

winddata.SetWindData();

return winddata;

}

else

{

throw new Exception(response.ReasonPhrase);

}

}

}

}

}

### WindArrow.cs

using System;

using System.Windows.Controls;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace Optimal\_Route\_Calculator

{

public class WindArrow : Arrow

{

private readonly int height = 40;

private readonly int width = 60;

private readonly int top = 40;

private readonly int left = 40;

public WindArrow(Canvas MyCanvas)

{

uri = ($"pack://application:,,,/Images/Arrow.png");

bitmapImage = new BitmapImage(new Uri(uri));

Skin.ImageSource = bitmapImage;

shape = new Rectangle { Width = width, Height = height, Fill = Skin };

// Sets the centre of rotation

rotate.CenterX = 30;

rotate.CenterY = 20;

GetTop = top;

GetLeft = left;

Canvas.SetLeft(shape, GetLeft);

Canvas.SetTop(shape, GetTop);

MyCanvas.Children.Add(shape);

}

}

}

### WindModel.cs

using System;

using System.Collections.Generic;

namespace Optimal\_Route\_Calculator

{

public class WindModel

{

private const double conversion\_to\_knots = 1.943844;

public Dictionary<string, string> Currently { get; set; }

public string wind\_speed;

public double wind\_bearing;

public void SetWindData()

{

// Converts wind speed in m/s to knots using the constant

double windSpeed = Math.Round(Convert.ToDouble(Currently["windSpeed"]) \* conversion\_to\_knots, 2);

wind\_speed = Convert.ToString(windSpeed);

wind\_bearing = Convert.ToDouble(Currently["windBearing"]);

}

}

}

### HelpWindow.xaml.cs

using System.Windows;

namespace Optimal\_Route\_Calculator

{

/// <summary>

/// Interaction logic for HelpWindow.xaml

/// </summary>

public partial class HelpWindow : Window

{

public HelpWindow()

{

InitializeComponent();

}

}

}

### MainWindow.xaml

<Window x:Class="Optimal\_Route\_Calculator.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Optimal\_Route\_Calculator"

mc:Ignorable="d"

ResizeMode="CanResize"

Title="MainWindow" d:DesignWidth="1242" x:Name="Main\_Window">

<Canvas Name="MyCanvas" Focusable="True" Background="LightGray" x:FieldModifier="public" KeyUp="KeyIsUp" KeyDown="KeyIsDown" MouseLeftButtonUp="LeftMouseIsUp" MouseRightButtonUp="RightMouseIsUp">

<StackPanel x:Name="InputPanel" Canvas.Left="1070">

<Button x:Name="btnHelp" Content="HELP" Click="OnHelp" ClickMode="Release" Margin="5, 5, 5, 5" Width="150" Background="Yellow"/>

<Button x:Name="btnPlot" Content="PLOT" Click="OnPlot" ClickMode="Release" Margin="5, 5, 5, 5" Width="150" Background="Red"/>

<Button x:Name="btnOptimise" Content="OPTIMISE" Click="OnOptimise" ClickMode="Release" Margin="5, 5, 5, 5" Width="150" Background="Red"/>

<Button x:Name="btnSaveLoad" Content="SAVE/LOAD" Click="OnSaveLoad" ClickMode="Release" Margin="5, 5, 5, 5" Width="150" Background="SteelBlue"/>

<Button x:Name="btnResetButton" Content="RESET" Click="OnReset" ClickMode="Release" Margin="5, 5, 5, 5" Width="150" Background="GreenYellow"/>

<TextBlock x:Name="WindAngleInputTextBlock" Text="Angle to wind (Default is 40º):" Margin="5, 5, 5, 0" Width="150" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<TextBox x:Name="WindAngleInput" Margin="5, 0, 5, 5" Width="150"/>

<TextBlock x:Name="MaxHullSpeedInputTextBlock" Text="Max Speed (Default is 6Kts):" Margin="5, 5, 5, 0" Width="150" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<TextBox x:Name="MaxHullSpeedInput" Margin="5, 0, 5, 5" Width="150"/>

<TextBlock x:Name="InputStatusMessageTextBlock" Margin="5, 5, 5, 0" Width="150" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

</StackPanel>

<StackPanel Name="InfoPanel" Canvas.Left="1070" Canvas.Top="400">

<TextBlock x:Name="RouteInfoTextBlock" Text="Map Infomation:" Margin="5, 5, 5, 5" Width="150" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

</StackPanel>

<Rectangle x:Name="LoadingRectangle" Fill="#3F3F41" Width="1250" Height="620"/>

<Image x:Name="LoadingCompass" Source="Images/Loading Compass.jpg" Width="400" Height="400" Canvas.Left="410"/>

<TextBlock x:Name="LoadingText" Text="LOADING" FontSize="60" Foreground="White" Canvas.Left="495" Canvas.Top="400" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

</Canvas>

</Window>

### SaveLoadWindow.xaml

<Window x:Class="Optimal\_Route\_Calculator.SaveLoadWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Optimal\_Route\_Calculator"

mc:Ignorable="d"

Title="SaveLoadWindow" Height="500" Width="450">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="0.3\*"/>

<RowDefinition Height="0.7\*"/>

</Grid.RowDefinitions>

<Border Grid.Row="0" Background="#2f5cb6"/>

<Border Grid.Row="1" Background="#1f3d7a"/>

<StackPanel Grid.Row="0">

<TextBlock Text="SAVE/LOAD WAYPOINTS:" Margin="5, 5, 5, 5" FontSize="20" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<TextBlock Text="SAVING:" FontSize="18" Style="{StaticResource CustomFont}" FontWeight="Thin" Margin="5, 0, 5, 0"/>

<TextBlock Text=" - To save to a text file (.txt) enter the file name then click save" FontSize="12" FontWeight="Thin" Style="{StaticResource CustomFont}" />

<TextBlock Text="LOADING:" FontSize="18" Style="{StaticResource CustomFont}" FontWeight="Thin" Margin="5, 0, 5, 0"/>

<TextBlock Text=" - If there are existing waypoints, they will be deleted on loading a new route&#x0a; - Select a save file from the box below then click load " FontSize="12" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<TextBlock x:Name="StatusMsg" Text="" Margin="5, 5, 5, 5" FontSize="12" Style="{StaticResource CustomFont}" FontWeight="Thin" Foreground="OrangeRed"/>

</StackPanel>

<StackPanel Grid.Row="1">

<TextBlock Text="Enter the save file name:" Margin="5, 5, 5, 5" FontSize="20" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<TextBox x:Name="txtBoxFileName" Margin="5, 5, 5, 5" />

<StackPanel Orientation="Horizontal">

<Button Content="SAVE" Width="150" Margin="5, 5, 5, 5" Click="OnSave" ClickMode="Release"/>

<Button Content="LOAD" Width="150" Margin="5, 5, 5 ,5" Click="OnLoad" ClickMode="Release"/>

</StackPanel>

<TextBlock x:Name="txtBlockExistingFiles" Text="Previous Save Files:" Margin="5, 5, 5, 5" FontSize="12" Style="{StaticResource CustomFont}" FontWeight="Thin"/>

<ListBox x:Name="lstBoxExistingFiles" Margin="5, 5, 5, 5"/>

</StackPanel>

</Grid>

</Window>

### NewTidePointWindow.xaml

<Window x:Class="Optimal\_Route\_Calculator.NewTidalPointWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:oxy="http://oxyplot.org/wpf"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Optimal\_Route\_Calculator"

mc:Ignorable="d"

Title="New Tidal Point" Height="230" Width="480" >

<Window.DataContext>

<local:TidalFlowModel/>

</Window.DataContext>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Column="1">

<oxy:PlotView x:Name="TidalFlowPlotModel" Model="{Binding tideModel}" Margin="10" Height="170" Width="200" />

</StackPanel>

<StackPanel Grid.Column="0">

<TextBlock Text="NEW TIDAL STREAM POINT:" Style="{DynamicResource CustomFont}" FontWeight="Thin" FontSize="20"/>

<StackPanel Orientation="Horizontal">

<TextBlock Text="Minimum Flow:" Style="{DynamicResource CustomFont}" FontWeight="Thin" Margin="5, 5, 5, 5"/>

<TextBox x:Name="txtBoxMinFlow" Margin="5, 5, 5, 5" Width="50"/>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Text="Maximum Flow:" Style="{DynamicResource CustomFont}" FontWeight="Thin" Margin="5, 5, 5, 5"/>

<TextBox x:Name="txtBoxMaxFlow" Margin="2, 5, 5, 5" Width="50" />

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Text="Bearing:" Style="{DynamicResource CustomFont}" FontWeight="Thin" Margin="5, 5, 40, 5"/>

<TextBox x:Name="txtBoxTideBearing" Margin="5, 5, 5, 5" Width="50"/>

</StackPanel>

<TextBlock x:Name="InputStatusMessageTextBlock" Margin="5, 5, 5, 5" Width="150" Style="{DynamicResource CustomFont}" FontWeight="Thin"/>

<Button x:Name="btnNewTidePoint" Content="Create New Tidal Stream Point" Margin="5, 5, 5, 5" Click="BtnNewTidePointClick"/>

<Button x:Name="btnClose" Content="CLOSE" Click="BtnCloseWindow" ClickMode="Release" Margin="5, 5, 5, 5" />

</StackPanel>

</Grid>

</Window>

### HelpWindow.xaml

<Window x:Class="Optimal\_Route\_Calculator.HelpWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Optimal\_Route\_Calculator"

mc:Ignorable="d"

Title="Help Window" Height="500" Width="1000">

<Grid>

<StackPanel>

<TextBlock Text=" WAYPOINTS:" FontSize="20" />

<TextBlock Text=" - Place waypoints on the map by pressing the left mouse button over the sea&#x0a; - Delete a waypoint by pressing on the right mouse button over a waypoint"

FontSize="20" FontWeight="Thin" Style="{StaticResource CustomFont}"/>

<TextBlock Text=" TIDE POINTS:" FontSize="20"/>

<TextBlock Text=" - Place tide points on the map by shift-left clicking over the sea&#x0a; - Input values for the maximum and minimum tidal flow as well as one of the tidal bearings" FontSize="20" FontWeight="Thin" Style="{StaticResource CustomFont}"/>

<TextBlock Text=" PATHFINDING:" FontSize="20" />

<TextBlock Text=" - Pressing the optimise button will run A\* pathfinding between each waypoint"

FontSize="20" FontWeight="Thin" Style="{StaticResource CustomFont}"/>

<TextBlock Text=" PLOTTING:" FontSize="20" />

<TextBlock Text=" - Pressing the plot button will plot the route between each waypoint&#xA; - If travelling upwind this will plot a tacking route, 40º either side of the wind, this value can be changed by typing a&#xA; new value in the &quot;angle to wind&quot; box&#xA; - The tacking algorithm will follow the 40º line until it reaches 10º from the waypoint and the boat's start position&#xA; however if the next route line is less than 10 minutes long it will sail until it hits the layline &#xA; - Pressing F with the mouse over a waypoint will display the approximate arrival time at that waypoint"

FontSize="20" FontWeight="Light" Style="{StaticResource CustomFont}"/>

<TextBlock Text=" SAVING/LOADING:" FontSize="20"/>

<TextBlock Text=" - To save the current waypoints click the Save/Load button&#xA; - Input the save file name"

FontSize="20" FontWeight="Thin" Style="{StaticResource CustomFont}"/>

</StackPanel>

</Grid>

</Window>

## Testing

### Modular Tests:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No. | Window | Purpose | Test Data | Expected Result | Type | Result |
| 1. | Map Window | Check that a waypoint can be placed on the map | Left click over an area of sea | Red Circle should appear | Normal | Pass |
| 2. | Map Window | Check that a waypoint cannot be placed outside the map or on land | Left click over an area of land or outside the map | Nothing should happen | Erroneous | Fail – In some locations a waypoint can be placed on land |
| 3. | Map Window | Check that the plot route button works | Click the plot button | A tacking route should be plotted between waypoints, displayed with a red line. | Normal | Pass |
| 4. | Map Window | Check that the optimise button works | Click the optimise button | The shortest route between waypoints should be plotted, avoiding land | Normal | Pass |
| 5. | Map Window | Check that after the shortest path algorithm is run the redundant nodes will be deleted | Run the shortest path algorithm both with and without the redundant node removal subroutine | Any nodes which a previous node has a line of sight (not blocked by land) on should be removed from the chain | Normal | Partial Pass – See evidence table below |
| 6. | Map Window | Check that the help button works | Click the help button | A new window should appear displaying the help information | Normal | Pass |
| 7. | Map Window | Check that the Save/Load button works | Click the Save/Load button | A new window should appear displaying the save/load window | Normal | Pass |
| 8. | Map Window | Check that the reset button works | Click the reset button | All the items on the map should disappear | Normal | Pass |
| 9. | Map Window | Check that valid “angle to wind” angle is accepted | Enter a non-negative double value less than 45 | Value should be accepted and changed throughout the program | Normal | Pass |
| 10. | Map Window | Check that an invalid “angle to wind” is not accepted | Enter a negative value and a value greater than 45 | Value shouldn’t be changed throughout the program | Erroneous | Pass |
| 11. | Map Window | Check that a valid “hull speed” value is accepted | Enter a non-negative double | Hull speed value should be changed throughout the program | Normal | Pass |
| 12. | Map Window | Check that an invalid “hull speed” value is not accepted | Enter a negative value | Hull speed value shouldn’t be changed throughout the program | Erroneous | Pass |
| 13. | Map Window | Check that the value route distance calculated and displayed is correct | Place two waypoints at specific locations then measure the distance between the same location on <https://www.freemaptools.com/measure-distance.htm> | The value my system calculates should be the same as the online map measure | Normal | Pass – 0.64% Error |
| 14. | Save/Load Window | Check that the load button will load waypoints from a file | Click on an existing save file from the list box and press the load button | The system should load the new waypoints onto the map | Normal | Pass |
| 15. | Save/Load Window | Check that the file name input will not accept erroneous characters. | Enter a file name containing, <, >, :, “, /, \, |, ?, \*. Or a file name that is CON, LPT, COM, PRN, AUX since these are all DOS device files | The system should display a message to the user prompting them to enter a different name | Erroneous | Pass |
| 16. | Save/Load Window | Check that the system will save the waypoint locations to file | Press the save button with a valid file name | The SaveFiles directory should contain a new file with the correct waypoints | Normal | Pass |
| 17. | Save/Load Window | Check that if the file name already exists the program won’t break | Enter an existing file name and press the save button | The system should prompt the user to enter a unique file name | Erroneous | Pass |
| 18. | Map Window | Check that the user can change the current map on the screen | Press the A and D keys to switch between maps | The system should display the next map in sequence and the waypoints on each map should be retained | Normal | Pass |
| 19. | Map Window | Check that the user can delete a waypoint | Right-click with the mouse over an existing waypoint | The red circle should disappear and the previous and subsequent waypoints in the sequence should be linked with a black line | Normal | Pass |
| 20. | Map Window | Check that the user can place a tide point | Shift Left-Click in a sea location on the map | A blue circle should appear with an arrow inside and a “NewTidePointWindow”  Should appear | Normal | Pass |
| 21. | New Tide Point Window | Check that the tide point arrow points towards the bearing submitted by the user | Enter a bearing between 0 and 360 degrees in the bearing input in the “NewTidePointWindow” | The tide arrow should point in the direction of the inputted bearing | Boundary | Pass |
| 22. | New Tide Point Window | Check that the program will not break if the user enters angles greater than 360 or less than 1 | Enter a bearing of, 361 and -1 into the bearing input | A new tide point should not be placed with these inputs | Erroneous | Fail – 0 should be a valid bearing however, a waypoint is not  placed.  EDIT: Fixed |
| 23. | New Tide Point Window | Check that the program will not place a waypoint if the min/max flow inputs are negative or if the min flow > max flow | Test 1:  Max Flow Input = 1  Min Flow Input = 2  Test 2:  Max Flow Input = -1  Min Flow Input = -1 | A new tide point should not be placed with either of these tests | Erroneous | Pass |
| 24. | Map Window | Check that the wind arrow is pointing in the same direction as the real-life wind is and that the wind speed is the same as in real-life | Check the wind arrow bearing is the same as the real wind bearing online. | The wind bearings should be the same. | Normal | Partial Pass – See Evidence table |
| 25. | Map Window | The fastest route algorithm should avoid going through areas of tidal flow that are opposite to the boat’s direction | Place two waypoints, one either side of the isle of weight since there are two routes around the island. Then place a tide point with a high tidal flow at a bearing in the opposite direction that the ship will be travelling in. Run the pathfinding algorithm, then run it without the tide point. | The first test should avoid the area of high flow and the second test should go the normal fastest route if there was no flow. | Normal | Pass |
| 26. | Map  Window | Check that the wind speed indicator displays the correct value. | Use a weather website such as [https://www.windfinder.com/#7/50.9480/0.4724](https://www.windfinder.com/%237/50.9480/0.4724) to see if the wind speed matches. | The value should be the same | Normal | Partial Pass – See evidence table below |
| 27. | Map Window | Check that the tidal flow algorithm outputs the correct velocity that the boat can travel through a node. | I created an excel spreadsheet where I inputted the formula which allowed me to test several inputs, finally I can compare these values with the value shown in the debugger. | The value for the final velocity returned by my program should be the same as the value outputted by excel in all cases | Normal | Pass |
| 28. | Map Window | Check that the route time indicator displays the correct time. | Plot a straight-line route between two nodes divide the route length by the default value for the max hull speed and check if that matches the value in the program | The value should be the same | Normal | Pass |
| 29. | Map Window | Check that the Terrain Map that is generated by the system is accurate and represents the actual land and sea locations on the original image. | Open the terrain map text files the text file should look like the map image since the pixels in the image are arranged in the same pattern in the terrain map file. | The two images should look very similar. | Normal | Partial Pass – see evidence table below |
| 30. | New Tide Point Window | Check that the system will display the correct cos wave representing tidal flow with respect to time. | Enter any values for the min and max flow which describe the amplitude of the wave. | The cos wave should have the amplitude described by the user’s inputs and it should have a period of 6hrs which is the tidal period in real life. | Normal | Pass |
| 31. | Main Window | Check that the system will delete tide points | Shift-Right click over a tide point | The tide point should disappear | Normal | Pass |
| 32. | Main Window | Check that the App won’t throw an error if the user doesn’t have a network connection | Run the program without a network connection | If the program doesn’t throw an error the wind angle should default to 90 degrees and the time until high tide will default to zero. Since without network connection my application cannot access the API data | Normal | Pass |
| 33. | Main Window | Check that the app won’t break if the user tries to plot a route between two inaccessible waypoints | Run the pathfinding algorithm between two inaccessible waypoints. | The program shouldn’t plot anything | Erroneous | Fail – See evidence below |

|  |  |  |
| --- | --- | --- |
| Test No | Evidence | Information |
| 1. |  | A red circle appears at the mouse location when the user left clicks at a sea location. |
| 2. | Before Fix:  After Fix: | A waypoint can be placed on some land locations – a section of land is determined by its colour, by adding a larger range of values that are treated as sea locations these areas are mitigated. |
| 3. |  | These screenshots show the waypoints that the user has placed (red circles), they are each joined by a black line in sequence. After they pressed the ‘plot’ button the tacking algorithm is run, and the red is overlaid onto the map to display the route they would have to take. The first screenshot shows only one leg needing to tack becomes the others were all downwind. The second screenshot shows the wind arrow, the route is plotted clockwise around the Isle of Weight, so the two legs that are within 40 degrees of the wind bearing are plotted with the tacking algorithm. I was able to retest this algorithm with preloaded waypoints once I had programmed the save/load algorithm, which have boundary angles between them of 0 degrees and 360 degrees. |
| 4. |  | The shortest path algorithm plotted the shortest route between the start and end waypoints, a new waypoint is placed every 4 DIPs (Device independent pixels). |
| 5. |  | Plotted from East to West, this algorithm partially works however its clear to see that the first node in the chain has line of sight on a few others that haven’t been deleted, this occurs due to a few pixels of water that are classified as land. After some changes to the range of colours are registered as sea this effect is mitigated. |
| 6. |  | This screenshot shows the help window after pressing the help button. This window explains how to use the program. |
| 7. |  | This screenshot shows the Save Load window which opens after pressing the Save/Load button. |
| 9. |  | This shows the status message for the valid wind angle input. |
| 10. |  | These screenshots show the conditions which are just outside the boundaries, 45 degrees and -1. It also shows that if anything other than an integer is inputted the program won’t break. |
| 11. |  | This shows the status message if the hull speed input is valid. |
| 12. |  | This shows the status message if the hull speed input is negative or not a number. |
| 13. |  | In order to test the route length calculator, I chose a straight line between two easy to find points, the most eastern point to the most western point of the isle of weight. In this case I wanted it to plot through the land so the measurement would be the same using <https://www.freemaptools.com/measure-distance.htm>.  In this case Map tools measured 23.11Nm. |
| 14. |  | The top screenshot shows the map after loading the waypoints from the test2.txt save file which shows that the Load button and the existing save files List Box works. The screenshot below is the corresponding save file. I also tested it with a save file which contains zero waypoints. |
| 15. |  | These screenshots show that the waypoint locations can be saved in a text file after entering a valid file name and pressing the save button, this works with zero waypoints which is the only boundary condition. |
| 16/17. |  | These screenshots show some the testing of the input validation for the save file input box, the file name will not be accepted if it already exists or if it contains one of the windows DOS file names. |
| 18. |  | The top screenshot shows the map before pressing ‘D’ and the bottom screenshot shows the map after pressing ‘D’. Any objects such as waypoints, tide points, waypoint connector lines and the red route lines are contained within the map segment object so they will be redrawn with the map if the user navigates back to the original. If the reset button is ever pressed all tide points and waypoints on all maps will be deleted. |
| 20. |  | This shows how when the user enters normal values in the min/max flow and the angle inputs the tide point will be created at the location that they shift-left clicked. The tidal arrow inside the tide point shows the bearing described by the user. |
| 21. |  | This test shows the tide point status message with normal inputs. |
| 22. |  | This shows the tidal input status message for invalid bearings, bearings greater than 360 and less than 0 will not be accepted. |
| 23. |  | This shows the tide point status message for invalid inputs in the minimum and maximum flow inputs. This will occur when the min flow is greater than the max flow or when either of the min/max flow inputs are negative. |
| 24. |  | This test was to check that my system displays the correct wind direction, my system (above) displays the direction the wind is going. The met office (<https://www.metoffice.gov.uk/public/weather/wind-map/#?tab=map&map=Wind&zoom=9&lon=-1.43&lat=50.73&fcTime=1605160800>) displays the wind as where the wind is coming from meaning the arrows will be offset 180 degrees. In order to fix this, I just had to add 180 degrees to the rotation of the arrow. |
| 25. |  | In the top screenshot there is a tidal point with a maximum of 4kts of tide with a bearing of 270 and 90 degrees, the route was plotted from the West of the map to the East of the map in both cases. In the top screenshot however, the route is automatically plotted to avoid the area of high tide that would slow it down. In the bottom screenshot there is no tide point and the fastest route was the northern route. This shows how the shortest path algorithm takes the tide into account. |
| 26. |  | Using this website I can determine that the actual wind speed around the isle of wight is around 7-8kts: <https://darksky.net/forecast/50.7624,-1.2979/si12/en>  Which is like the value shown by my system, this however can vary greatly depending on the website. |
| 27. |  | These screenshots show that the value calculated by my program was the same as the value calculated by excel using the same inputs. I then preceded to test this for each of the four quadrants that the tidal bearing can be in, and each of the 45-degree entry angles. In all the cases both systems outputted the same value. |
| 28. |  | 4.75 Nautical Miles / 6 Knots = 0.7917  Convert to minutes:  0.7917 \* 60 = 47.4 mins  6 Knots is the default speed, if I change the speed value then replot the route time will also change. As a later addition I could make the program recalculate the route time when the speed value is changed. |
| 29. |  | This is the screenshot of the text file that describes the land and sea locations of the map. The land locations are stored as a 1 and the sea locations are a 0. Since 0s consist of more black pixels than 1s which means that the sea will appear black and the land will appear whiter. The map appears stretched vertically because the characters are rectangles not squares.  The main problem with the algorithm that generates the terrain map array (described above) is that it will show any place names as land since they are not blue. This will mean that any place names which project into areas of sea will be falsely marked as land therefore the user will not be able to place waypoints on place names. At first, I tried to add the colours of the place names into the land detection algorithms range of sea colours but then I realised that this will allows the user to place waypoints in some areas on the land. This is one outstanding issue that I couldn’t figure out a method to fix, other than finding a screenshot of a map that doesn’t contain place names. However, doing this would require reprogramming the land detection algorithm as the land and sea colours would be different. Choosing the original range of colours was a very time-consuming process as it was mainly trial and error, so I didn’t have the time to change it. |
| 30. |  | The screenshot shows the cosine graph with a maximum amplitude of 5 and a minimum of 1. The tidal period can be seen as 6hrs from the graph as well. |
| 32. |  | When the app doesn’t have a network connection the wind speed and direction will use default values. |
| 33, |  | The top screenshot shows the route before applying the pathfinding algorithm between two inaccessible regions of water. The second screenshot displays the route after applying the pathfinding algorithm. A new waypoint is placed at the bottom left most pixel of water on the map, the system doesn’t throw an error however it shows the incorrect result. |

### Full System Test:

|  |  |  |
| --- | --- | --- |
| Test. | Explanation. | Evidence. |
| Place Initial route | Added four manually placed waypoints starting in the South West corner of the map |  |
| Run the pathfinding algorithm | Clicked the optimise button which will run the pathfinding algorithm between the 4 placed waypoints in order |  |
| Run the tacking route plotter | Clicked the plot button which will run the tacking route plotter algorithm |  |
| Save the new waypoints | Clicked the save/load button, entered a valid filename and pressed the save button to save the new waypoints. |  |
| Delete all waypoints on the map then load previous waypoints | Click the reset button, then click the save/load button. Next select the save file and press load. |  |
| Switch Map segments and place some more waypoints | Press the D key to switch to the second map segment, and place some new waypoints by clicking with the mouse |  |
| Add some tide points | Shift-Left click on a sea location and enter valid values for the minimum and maximum velocities as well as the bearing. |  |