william roebuck

Centre number: 48317

Candidate Number: 6773

VisuGraph

General purpose graph visualisation software development project

Software development project

William Roebuck

Contents

[Analysis 3](#_Toc118378917)

[The problem 3](#_Toc118378918)

[Computational methods 3](#_Toc118378919)

[Key features that make the program solvable by computational methods: 3](#_Toc118378920)

[Object oriented programming 3](#_Toc118378921)

[Functional and procedural programming 4](#_Toc118378922)

[Visualisation 4](#_Toc118378923)

[Heuristics and backtracking 4](#_Toc118378924)

[Initial decomposition 4](#_Toc118378925)

[Stakeholders 4](#_Toc118378926)

[Esteban Garcia 4](#_Toc118378927)

[Paula Weston 4](#_Toc118378928)

[Harry Thorne 4](#_Toc118378929)

[Interview 4](#_Toc118378930)

[Research and existing solutions 7](#_Toc118378931)

[CS academy graph editor 7](#_Toc118378932)

[Graph online 8](#_Toc118378933)

[Gephi 8](#_Toc118378934)

[NetworkX 9](#_Toc118378935)

[vis.js 9](#_Toc118378936)

[graph 9](#_Toc118378937)

[Essential Features 10](#_Toc118378938)

[Limitations 10](#_Toc118378939)

[Hardware and software requirements 11](#_Toc118378940)

[Success Criteria 11](#_Toc118378941)

[Design 12](#_Toc118378942)

[Decomposition 12](#_Toc118378943)

[Structure 13](#_Toc118378944)

[Iteration 1 13](#_Toc118378945)

[Iteration 2 13](#_Toc118378946)

[Iteration 3 13](#_Toc118378947)

[Iteration 4 14](#_Toc118378948)

[Algorithms 14](#_Toc118378949)

[Save 14](#_Toc118378950)

[Load 14](#_Toc118378951)

[Autosave 15](#_Toc118378952)

[Add node to graph 15](#_Toc118378953)

[Dikstra’s 15](#_Toc118378954)

[A\* 15](#_Toc118378955)

[Heuristic 16](#_Toc118378956)

[Kruskal’s 16](#_Toc118378957)

[Brute Force Travelling Salesman 17](#_Toc118378958)

[Floyd’s algorithm 18](#_Toc118378959)

[Display graph 18](#_Toc118378960)

[Update graph 19](#_Toc118378961)

[Node constructor 19](#_Toc118378962)

[Join nodes 19](#_Toc118378963)

[Arc constructor 19](#_Toc118378964)

[Get arc weight 19](#_Toc118378965)

[Set arc weight 20](#_Toc118378966)

[Get arc name 20](#_Toc118378967)

[Set arc name 20](#_Toc118378968)

[Node click handler 20](#_Toc118378969)

[Edge click handler 20](#_Toc118378970)

[Dark theme toggle 21](#_Toc118378971)

[Font size toggle 21](#_Toc118378972)

[Text shortener 21](#_Toc118378973)

[Zoom 21](#_Toc118378974)

[Data Structures 22](#_Toc118378975)

[Usability Features 22](#_Toc118378976)

[Layout 22](#_Toc118378977)

[Stakeholder feedback 24](#_Toc118378978)

[Test Data 24](#_Toc118378979)

[Iteration 1 25](#_Toc118378980)

[Iteration 2 26](#_Toc118378981)

[Iteration 3 28](#_Toc118378982)

[Iteration 4 29](#_Toc118378983)

[Post development 30](#_Toc118378984)

[Implementation 31](#_Toc118378985)

[Iteration 1 testing: 31](#_Toc118378986)

[Iteration 1 evidence: 32](#_Toc118378987)

[Code added as a result of error: 33](#_Toc118378988)

[Code added as a result of error: 34](#_Toc118378989)

[Iteration 1 stakeholder feedback and response 35](#_Toc118378990)

[Iteration 2 testing: 36](#_Toc118378991)

[Iteration 2 evidence: 38](#_Toc118378992)

[Code added as a result of error: 42](#_Toc118378993)

[Code added as a result of error: 43](#_Toc118378994)

[Code added as a result of error: 51](#_Toc118378995)

[Code added as a result of error: 57](#_Toc118378996)

[Iteration 2 stakeholder feedback and response 61](#_Toc118378997)

[Iteration 3 testing 62](#_Toc118378998)

[Iteration 3 evidence: 63](#_Toc118378999)

[Iteration 3 stakeholder feedback and response 67](#_Toc118379000)

[Iteration 4 testing 67](#_Toc118379001)

[Iteration 4 evidence: 68](#_Toc118379002)

[Evaluation 76](#_Toc118379003)

[Testing for function 76](#_Toc118379004)

[Testing for robustness 76](#_Toc118379005)

[Testing for usability 76](#_Toc118379006)

[Success Criteria 76](#_Toc118379007)

[Further development 76](#_Toc118379008)

[Maintenance 76](#_Toc118379009)

# Analysis

## The problem

Graphs are useful for a wide variety of applications, including computer networking and urban planning, however there are very few general-purpose node programs that can be used without significant programming experience. The aim of my program is to be easy to set up and intuitive to use, while still maintaining a high level of functionality.

This program will allow the user to create, edit and view weighted node networks, including algorithms such as Dijkstra’s, A\*, Kruskal’s, Floyd’s, and brute force travelling salesman. The program should save graphs to a file and can load graphs from files.

## Computational methods

This problem is suitable for a computational solution mainly to exploit the benefits of abstraction. Since the computation will be done for the user, a high level of expertise is not needed to understand the outputs of the program and visualisation will aid this.

### Key features that make the program solvable by computational methods:

1. Algorithms can be performed on the graph
2. The inputs can be taken from the keyboard and mouse to generate outputs of a visualised graph or algorithm
3. There are points of decision throughout the program, such as what a mouse click should do, affected by its position.

### Object oriented programming

OOP will play a large part in this project’s development as I intend to implement a “Graph” class that will be used for almost all of the program’s features. This has the benefit that the code is reusable if I create a project in future that includes graphs but it is also a useful abstraction, in that it is more intuitive to access the parts of a graph (Graph.Count, Graph.Weight) as attributes rather than as standalone variables (graphCount, graphWeight) which will help with future development as well as making code easier to understand. Furthermore, the class will have many methods, at least one for each graph algorithm that can be used, so that the graph and its data do not have to be passed into a function outside of the class which improves security.

### Functional and procedural programming

Functional and procedural programming will also be important for this project for handling most of the user driven parts of the program, including the UI. These subprograms will allow for event driven code (through windows forms) and make code more human readable.

### Visualisation

Visualisation is essential to the program and is key to making the program user friendly. Graphs will be visualised as soon as they are created/loaded and will update to reflect changes made by the user. It will also be used to show the outputs of algorithms such as Dijkstra’s by highlighting the given route

### Heuristics and backtracking

Heuristics will be used as part of A\* (which improves performance) whereas Dijkstra’s algorithm will use backtracking.

### Initial decomposition

The program will consist of 4 main parts, each of these will be broken down further in the design section

1. User interface with buttons, icons and graph rendering ability
2. Graph data structures that can be saved and loaded to/from files
3. A collection of algorithms that can be performed on the graphs
4. Graph editing capability (click and drag/add nodes/edit edges/edit nodes)

## Stakeholders

### Esteban Garcia

Esteban Garcia is a 57-year-old urban planner who is not fluent in English and requires the program to allow accents such as á, é, í, ñ, ó, and ú as inputs. He is familiar with graph structures but usually uses paper to work with them. He needs the program to be flexible in what it can display and calculate. As he may use the program in his work, he needs it to be robust and save data in the case of a crash, which should be a rare occurrence if it does ever happen.

### Paula Weston

Paula Weston is a partially sighted 17-year-old further maths student who requires the program to have a dark colour palette and an option to increase font size for the whole program. She needs to be able to create graphs quickly and wants a clear output that highlights things such as the shortest path for Dijkstra’s

### Harry Thorne

Harry Thorne is a 34-year-old professional cyclist who requires the program to be easy to use and understand without studying computer science or further maths. He will use the program to plan his routes and needs the nodes and edges to be able to store and display long multi-word names such as “Barton-upon-Humber” or “Cammeringham Hill”. Harry also needs the program to be able to handle large, detailed networks without becoming unusable or unstable.

## Interview

**How will you use the program?**

**Garcia:** I intend to use it to speed up the initial planning phase of my work and reduce the amount of writing materials I go through

**Weston:** I want to use the program to create questions and answers so I can improve my understanding in further maths. For example: I want to be able to create a graph in the program, work out the shortest route by hand and then verify if my answer is correct using the program.

**Thorne:** I spend a lot of time planning long cycles around the UK that often involve visiting a number of key locations like B&Bs or landmarks so I want to use the program to not only speed this up but also make sure that I’m taking the best route

**Do you have any hardware limitations?**

**Garcia:** I have not upgraded my hardware since 2010 and I am used to it being slow. As long as the program is responsive enough so that I can use it, it will be fine.

**(After some discussion, we came to the conclusion that for his PC, with 4GB of RAM, the response time should be no more than 2 seconds for any graph manipulation)**

**Weston: My PC is built for the latest videogames, so it should be able to run most things, but I dislike when the fan starts making a lot of noise, so I hope the program isn’t too intensive**

**Thorne: I have a somewhat modern laptop; it can run many tabs of Google Chrome without issues but it sometimes slows down during intensive physics simulations**

**Do you have any accessibility requirements?**

**Garcia:** My English is not very good, but I will be able to use the program as long as there are icons that show what things do

**Weston:** I am partially sighted and can’t use bright screens for a long time so I need text to be quite large and the interface to be dark

**Thorne:** No

**How experienced are you with graphs?**

**Garcia:** I studied them in university and use them quite often for my job. I can do some of the algorithms by hand, such as Dijkstra’s and Floyd’s.

**Weston:** I’m currently learning about them in further maths so I have a good understanding of them but I’d rather let the computer carry out the algorithms for me.

**Thorne:** I’ve never really used them before but I know that they can be very useful for route planning and optimisation

**How familiar are you with computer interfaces and how do you like to interact with them?**

**Garcia:** I use my company’s software on my computer, and most of it is not very user friendly so it takes a long time to get used to. I would prefer if I could just click buttons and not have to find everything in verbose menus and sub-menus.

**Weston:** I’ve used a lot of different programs for installations and niche applications so I am quite used to confusing user interfaces. I like to use my keyboard to navigate around the interface as its more convenient.

**Thorne:** I mostly use my computer for YouTube and Google but have used some physics software. I prefer visual representations in the user interface since I find icons easier to remember than words.

**At what scale are you likely to use the program?**

**Garcia:** My node networks are often between 25 and 50 nodes however there always far less edges than there could be (more like 300 connections instead of 1225)

**Weston:** I’d only need the program for really small networks; less than 20 nodes and 30 edges

**Thorne:** I don’t think I’d use any more than 15 nodes, but there’d be a lot of connections between them, about 100

**Would you like to name every node and edge?**

**Garcia:** Naming the nodes will be essential but I would only rarely name edges

**Weston:** The names of nodes and edges don’t matter to me; I’d prefer if each node was automatically named A-Z and the edges left blank

**Thorne:** I might not name every edge, but certainly the important ones and each node must have a name

**Which algorithms are the most important to you?**

**Garcia:** It is hard to decide, but I currently use Dijkstra’s the most. Floyd’s will also be very useful and I would use Kruskal’s if it is included

**Weston:** I think I’ll use all of them except maybe A\* and travelling salesman, but I’d say Dijkstra’s is the most important to me

**Thorne:** I’d have to test some of them out, but right now I think travelling salesman is the most important. I’d probably rather use A\* over Dijkstra’s because it doesn’t have to be 100% accurate for me, I’d prefer whichever’s quickest.

**What screen size do you use?**

**Garcia:** 640x480

**Weston:** 2560x1440

**Thorne:** 1280x720

## Research and existing solutions

### CS academy graph editor

<https://csacademy.com/app/graph_editor/> 

This web-based solution is an effective visualisation tool for node graphs, allowing the user to “draw” a graph by clicking and dragging on nodes. Adding nodes is efficient and intuitive as one must click an empty space and then simply click and drag to create a connection between them. Additional details such as weight and direction can be added later by clicking on the connection. The program is somewhat lacking in functionality in that it can only display a graph, not make any calculations using one.

The program also has a range of customisation features, including the ability to change node colour, size, and other options, however there is no way to increase font size.

Graphs can only be saved and loaded by copying the data textbox on the left into a file which makes managing a large group of graphs difficult.

From this I will incorporate the click and drag functionality and the elegant interface that becomes more complex as more information is added

### Graph online

https://graphonline.ru/en/



Graph online is another web-based solution for working with graphs but with a focus less on visualisation and more on algorithms. It offers 20 different useful algorithms that can be performed on your graph, selectable from a dropdown above the graph.

However, the program is hard to use in that to add a new vertex one must first select “Add vertex”, click on the graph, then “Connect vertices” and click on the two vertices. Then, a popup box appears that must be filled in each time a connection is made. In comparison to the CS academy graph editor, it is far slower and more tedious to use.

I will try to avoid using dialogue boxes to add parts to the graph in my program as they disrupt the creation process.

### Gephi

Gephi is a powerful graph visualisation tool that performs well with huge datasets, claiming to support up to 1 million nodes and edges at once. Its primary use is for visual data analysis and it offers a range of tools such as grouping, highlighting and naming to make this more effective. It also offers a zoom feature so that large networks can be closely examined

The UI makes use of icons for most tools, most of these make the function quite clear however some are more vague, such as the aeroplane icon. As shown in this screenshot, there is an option to change the font size, however this is only for the node names and not the UI controls.

As for algorithms, the program offers multiple layout algorithms to improve readability, a range of data analysis tools (Betweenness Centrality, Closeness, Diameter, Clustering Coefficient, and PageRank) and an unspecified shortest path algorithm.

Gephi can import the majority of graph file formats and can save and load projects to files. One drawback of this is that there are no tools inside the program to construct a graph visually by placing individual nodes.

### NetworkX

NetworkX is a collection of python modules that can be used to create, manipulate and analyse networks. It is not primarily a graph visualisation software but can be used with Matplotlib to draw its networks

Here is an example of how to construct a graph and find a shortest path between two nodes. While intuitive for a programmer, this text-based interface is not suitable for somebody with no experience. Furthermore, for very networks typing out each bracket, quotation mark, and comma would not be efficient.

A unique benefit of this solution is that “Nodes can be anything”, for example images, text or even XML records. Edges can also hold data such as weights

### vis.js

vis.js is a visualisation library for JavaScript, one of its many applications being network visualisation.

As shown here, it is quite versatile in that nodes can be any shape/colour and edges can have weight (indicated by their thickness), direction and labels. It also claims to support up to 1000 nodes and 1000 edges

Furthermore, as it is a JavaScript library, it can be implemented in such a way that allows the creation of networks without code and, while it does not include graph algorithms, the data structures it offers could be utilised to implement as many as the programmer desired.

### graph

graph is a graph visualisation library for Go and, while offering many of the features of solutions shown above, it also offers a few unique ones. It offers breadth and depth first traversal algorithms as well as an option to prevent cycles during graph creation.

## Essential Features

From my research and discussions with stakeholders, I have decided on these essential features:

* UI accessible algorithms to be performed on user’s graphs. These are important to the functionality of the program so that it can be used in the wide range of applications desired by my stakeholders. I have chosen to include Dijkstra’s **and** A\* because for some applications such as city planning, a large number of nodes will be required and Dijkstra’s may be too slow to remain effective at these scales.
  + Dijkstra’s – For finding the shortest path between two nodes, useful for planning and route finding
  + A\* – Same function as Dijkstra’s but faster at the cost of accuracy
  + Kruskal’s – For finding the minimum spanning tree, can be useful in planning and finding an upper and lower bound for travelling salesman
  + Brute force travelling salesman – For visiting every node in the network using the minimum weight.
  + Floyd’s – For constructing a minimum distances table, useful for route planning
* Ability to create, save, name, rename, and delete graphs (storing this data on file(s)) – This is important so that users can work on multiple projects concurrently
* Graph manipulation capability: Adding, removing, moving, and naming nodes; connecting nodes, naming, removing, and automatic/manual weighting connections. – Important for visualisation and the function of the program
* Accessibility settings: Font size, colour scheme, keyboard controls

## Limitations

Some solutions that I researched offered a “gravity” or automatic layout feature, where the nodes in the network automatically arrange themselves in a geometric way so that they are evenly spaced out. While this is useful for abstract applications where the physical layout of the nodes is not important, my program is primarily concerned with problems in the real world so will not include this.

Another feature provided by some solutions was directional edges, which, while useful for some applications are not necessary for any of my stakeholders’ use cases. In future, this feature could be implemented with some changes to the node class

One accessibility option that I have not accounted for is a screen reader to explain what the program is doing for blind users. I have decided not to implement this because one of the main features of the program is visualisation and there are other solutions which would be better suited to a purely numeric implementation. Furthermore, Windows 10, the operating system the program is designed for, comes with a screen reader so an additional one is not necessary.

Another potential feature would be language options; however, these should not be necessary as the program’s main purpose is visualisation so there will be more symbols than English language. Furthermore, all the algorithms have names or are named after somebody so will not need to be localised. The program will support user inputs from other languages, so the user can type with their language if they wish

Given the time provided, a step-by-step explanation of each algorithm’s process cannot be implemented. While this would be invaluable in an educational environment, this is not important for most applications. Using the program to teach graph theory is not any of my stakeholder’s intended use cases.

Added as a result of stakeholder feedback from design section: Unfortunately, there is an issue with some graphs in that overlapping edges can create ambiguity as for which arc is named/weighted which. While a solution for this could be implemented in the future, such as more advanced layout algorithms or links between the text and arc, this is outside the scope of the program in the time provided. In the meantime, this can be mitigated by slightly adjusting node positions so the label is more clearly linked

## Hardware and software requirements

To compile the program, I will need:

* Microsoft Visual Studio (which requires windows 10+)

The user will need:

* Windows 10[[1]](#footnote-1)
* 4 GB or more of RAM[[2]](#footnote-2)
* 1 GHz or faster processor[[3]](#footnote-3)
* 16 GB for 32-bit OS or 20 GB for 64-bit OS[[4]](#footnote-4)
* Built in microsoft libraries
* A keyboard, 4:3 or 16:9 display[[5]](#footnote-5), and mouse

## Success Criteria

Note to self: Specific, measurable, achievable, realistic, timely

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Justification |
| 1 | The user can create and manipulate graphs with GUI controls | This is an essential feature for the stakeholders’ use of the program |
| 2 | The user can run algorithms on their graphs that display an output |
| 2.1 | Dijkstra’s algorithm |
| 2.2 | A\* algorithm |
| 2.3 | Kruskal’s algorithm |
| 2.4 | Travelling salesman |
| 2.5 | Floyd’s algorithm |
| 3 | Initially, nodes should be named A-Z but edges should be left blank |
| 4 | The user can save and load graphs to/from files | A stakeholder has specified that the program must be suitable for long term, multi-project use |
| 5 | The user can name edges and nodes with titles up to 30 characters in length | Stakeholders require this feature as it is important for the program’s use to model real situations |
| 6 | Users can input Unicode which will be displayed with the same font, colour, size and spacing | A stakeholder has specified that the program must support languages with extended alphabets such as Spanish |
| 7 | If an error occurs, the user should be shown what has caused it | A stakeholder has said that the program may not be perfect, but that the user should be able to see what has gone wrong to avoid future errors |
| 8 | The program must have accessibility options to ensure suitability for partially sighted users | A stakeholder requires these features to use the program |
| 9 | In the event of a crash, the program should be able to recover a recent copy of the most recent graph if it has not been saved | This is important to stakeholders as the program must be reliable so that it can be used in industry |
| 10 | The program must be sufficiently stable and responsive (no more than 2 second delay for manipulation) with large networks up to 50 nodes and 300 edges in size. | Stakeholders may use the program for very large projects so it must be functional as scale grows |
| 11 | The program must be usable without any English language, (excluding algorithm names). I will ask my stakeholders to evaluate whether this criterion is met | Not all of my stakeholders are fluent in English, so I should ensure that the program can be used without relying on English |
| 12 | The program must have at least 3 zoom levels | My stakeholders will manage large networks so zooming is important for visualisation |
| 13 | If the user tries to perform an algorithm with invalid parameters (such as Dijkstra’s on a graph with loose nodes) they should be shown a popup explaining this | My users may make mistakes if they are not already familiar with the algorithms, the program should be robust in this case |
| 14 | All parts of the program accessible with the mouse should be accessible with the keyboard | Some people, including a stakeholder prefer to navigate programs using a keyboard |
| 15 | The program should be resizable to fit on a 4:3 or 16:9 screen | This will make the program more versatile and suit the stakeholders’ hardware |

# Design

## Decomposition

This diagram will be useful to me during development as it is arranged in such a way that each component is linked to what must be implemented before it and which must be implemented after. For example, to complete the graph data structure I must implement graph algorithms, which also require error handling and validation. I will also show this diagram to my stakeholders so they can understand the order in which I will create the program.

## Structure

To develop my program, I will use the spiral methodology and I have chosen it because it allows flexibility when my stakeholders requirements change. My stakeholders do not know exactly what they want the final product to look like so they will see it evolve as I develop it and will be able to suggest potential features that I can add in the next iteration. Furthermore, with this SDLC the final program will have excellent usability because my stakeholders can have an impact during development, giving me the flexibility to adapt features to their needs

Evaluation

Analysis phase

Implementation

Design

Start of project

1st prototype

2nd prototype

3rd prototype

Final prototype

### Iteration 1

The first iteration will focus on the user interface which I can then show to my stakeholders and ask whether they have any suggestions

Prototype 1.0 will only have the controls and no functionality

Prototype 1.1 will have a non-functional options menu that is shown when the options button is pressed

Prototype 1.2 will be able to toggle a dark colour palette and text size

### Iteration 2

The next iteration will implement and display the graphs

Prototype 2.0 will be able to display graphs input by the programmer

Prototype 2.1 will allow the user to rename nodes and edges and add weights

Prototype 2.2 will allow the user to move, add and delete parts of the graph

Prototype 2.3 will validate user input

Prototype 2.4 will implement the zoom feature

When this is complete, I will ask the stakeholders to create some graphs to ensure they are able to create what they desire

### Iteration 3

Iteration 3 implements the file IO which the stakeholders will test with their own graphs

Prototype 3.0 will allow the user to save and load graphs to/from a file

Prototype 3.1 will autosave and ask the user to recover data in the case of a crash

### Iteration 4

The final iteration implements the algorithms, which will once again be tested by the stakeholders on their graphs

Protype 4.0 will allow the user to perform and see the output of Dijkstra’s and A\*

Protype 4.1 will allow the user to perform and see the output of Kruskal’s and Floyd’s

Prototype 4.2 will allow the user to perform and see the output of the travelling salesman algorithm

Prototype 4.3 will validate user input to each algorithm

## Algorithms

### Save

Public Procedure Save(string dir)

string data = “GRAPH FILE\n”

// This will be used to detect whether the file is of the correct format

data += minWeight + “\n” + nodeID + “\n”

foreach (Node n in nodes)

data += n.name + “:”

foreach (Arc a in n.connections)

data += a.GetName() + “,” + a.destination.name + “,” + a.GetWeight()

data += “\n”

nextfor

try

CreateFile(dir, data)

wasSaved = True

catch (Exception e)

MsgBox(e)

End Procedure

### Load

Public Function Load(string dir)

string data = “”

try

data = LoadFile(dir)

catch (Exception e)

MsgBox(e)

Return new Graph()

string[] dataLines = data.Split(“\n”)

if (dataLines[0] != “GRAPH FILE”)

MsgBox(“This is not a graph file”)

Return new Graph()

endif

loadedGraph = new Graph()

loadedGraph.minWeight = dataLines[1]

loadedGraph.nodeID = dataLines[2]

for (i from 3 to dataLines.Length)

nameLength = dataLines[i].FindIndex(‘:’)

Node n = new Node(dataLines[i].Substring(nameLength))

loadedGraph.nodes.Add(n)

nextfor

for (i from 3 to dataLines.Length)

string connectionsLine = dataLines[i].Substring(nameLength, dataLines[i].length)

string[] connections = connectionsLine.Split(‘,’)

Node connectedNode

foreach (Node n in loadedGraph.Nodes)

if (n.name = connections[1])

connectedNode = n

endif

nextfor

Arc a = new Arc(connections[0], n, connections[2])

loadedGraph.nodes[i – 3].connections.Add(a)

nextfor

Return loadedGraph

End Function

### Autosave

Public Procedure Autosave()

//Called every 5 minutes

Save(“..\graphAutosave.txt”)

End Procedure

### Add node to graph

Public Procedure AddNode(ref Node n)

if (n.name = “”)

// Automatically assigns nodes with names A-Z, but if there are more than 26 they are named A1-Z1, then A2-Z2 and so on

if (nodeID > 26)

n.name = Asc((nodeID MOD 26) + 65) + (nodeID / 26).ToString()

else

n.name = Asc(nodeID + 65)

endif

endif

nodeID++

nodes.Add(n)

End Procedure

### Dikstra’s

Source: <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Pseudocode>

function Dijkstra(Graph, source)

for each vertex v in Graph.Vertices:

dist[v] = INFINITY

prev[v] = null

add v to Q

dist[source] = 0

while Q is not empty:

u = vertex in Q with min dist[u]

remove u from Q

for each neighbor v of u still in Q:

alt = dist[u] + Graph.Edges(u, v)

if alt < dist[v]:

dist[v] = alt

prev[v] = u

return dist[], prev[]

### A\*

Source: [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

function reconstruct\_path(cameFrom, current)

total\_path := {current}

while current in cameFrom.Keys:

current := cameFrom[current]

total\_path.prepend(current)

return total\_path

// A\* finds a path from start to goal.

// h is the heuristic function. h(n) estimates the cost to reach goal from node n.

function A\_Star(start, goal, h)

// The set of discovered nodes that may need to be (re-)expanded.

// Initially, only the start node is known.

// This is usually implemented as a min-heap or priority queue rather than a hash-set.

openSet := {start}

// For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start

// to n currently known.

cameFrom := an empty map

// For node n, gScore[n] is the cost of the cheapest path from start to n currently known.

gScore := map with default value of Infinity

gScore[start] := 0

// For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to

// how cheap a path could be from start to finish if it goes through n.

fScore := map with default value of Infinity

fScore[start] := h(start)

while openSet is not empty

// This operation can occur in O(Log(N)) time if openSet is a min-heap or a priority queue

current := the node in openSet having the lowest fScore[] value

if current = goal

return reconstruct\_path(cameFrom, current)

openSet.Remove(current)

for each neighbor of current

// d(current,neighbor) is the weight of the edge from current to neighbor

// tentative\_gScore is the distance from start to the neighbor through current

tentative\_gScore := gScore[current] + d(current, neighbor)

if tentative\_gScore < gScore[neighbor]

// This path to neighbor is better than any previous one. Record it!

cameFrom[neighbor] := current

gScore[neighbor] := tentative\_gScore

fScore[neighbor] := tentative\_gScore + h(neighbor)

if neighbor not in openSet

openSet.add(neighbor)

// Open set is empty but goal was never reached

return failure

### Heuristic

Public Function Heuristic(Node start, Node end)

// This function can be improved later, for now it returns the minimum weight of the network

Return minWeight

End Function

### Kruskal’s

Public Function Kruskal()

List<(Node start, Arc connection)> sortedList = new List<(Node start, Arc connection)>()

foreach (Node n in nodes)

foreach (Arc a in n)

sortedList.Add(n,a)

nextfor

nextfor

sortedList.Sort(Ascending)

Graph MST = new Graph()

int i = 0

MST.nodes.Add(DeleteArcs(ref sortedList[0].start))

MST.nodes.Add(DeleteArcs(ref sortedList[0].connection.destination))

MST.nodes[0].JoinTo(MST.nodes[1], sortedList[0].GetWeight())

while(MST.nodes.length < this.nodes.length)

if (MST.nodes.Contains(sortedList[i].start) = false)

MST.nodes.Add(DeleteArcs(ref sortedList[i].start))

if (MST.nodes.Contains(sortedList[i].connection.destination) = false)

MST.nodes.Add(DeleteArcs(ref sortedList[i].connection.destination))

MST.nodes.Last.JoinTo(MST.nodes[MST.nodes.length - 2], sortedList[i].connection.GetWeight())

else

MST.nodes.Last.JoinTo(MST.nodes.Find(sortedList[i].start), sortedList[i].connection.GetWeight())

endif

endif

i++

endwhile

Return MST

End Function

Private Function RemoveArcs(ref Node n)

n.connections = new List<Arc>()

return n

End Function

### Brute Force Travelling Salesman

Adapted from <https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/>

Public Function T\_Salesman(Node start)

List<Node> path = new List<Node>

foreach (Node n in nodes)

if (n != start)

path.Add(n)

endif

next for

double shortest = INFINITY

Do {

double currentPathWeight = 0

//Find current path weight

foreach (i from 0 to path.length - 2)

currentPathWeight += ArcBetween(path[i], path[i + 1])

nextfor

currentPathWeight += ArcBetween(path[0], path[path.length - 1])

// Update shortest length

List<Node> minPath = new List<Node>()

if (currentPathWeight < shortest)

shortest = currentPathWeight

minPath = path

endif

} while (FindNextPath(path))

if (shortest = INFINITY)

// This means that there is no possible route

Return new List<Node>()

else

Return minPath

endif

End Function

Private Function ArcBetween(Node start, Node end)

foreach (Arc a in start.connections)

if (a.destination = end)

return a.weight

endif

nextfor

return INFINITY

End Function

// Find the next permutation of the array

Private Function FindNextPath(List<Node> data)

// If the given dataset is empty or contains only one element next\_permutation is not possible

if (data.Count <= 1)

return false

endif

int last = data.Count - 2

// find the longest non-increasing suffix and find the pivot

while (last >= 0)

if (data[last] < data[last + 1])

break

endif

last--

endwhile

// If there is no increasing pair there is no higher order permutation

if (last < 0)

return false

endif

int nextGreater = data.Count - 1

// Find the rightmost successor to the pivot

for (int i = data.Count - 1; i > last; i--)

if (data[i] > data[last])

nextGreater = i

break

endif

nextfor

// Swap the successor and the pivot

data = swap(data, nextGreater, last);

// Reverse the suffix

data = reverse(data, last + 1, data.Count - 1)

// Return true as the next permutation is done

Return true

End function

### Floyd’s algorithm

Source: <https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm#Pseudocode_[11>]

let dist be a |V| times |V| array of minimum distances initialized to infinity

let next be a |V| times |V| array of vertex indices initialized to null

procedure FloydWarshallWithPathReconstruction() is

for each edge (u, v) do

dist[u][v] = w(u, v) // The weight of the edge (u, v)

next[u][v] = v

for each vertex v do

dist[v][v] = 0

next[v][v] = v

for k from 1 to |V| do // standard Floyd-Warshall implementation

for i from 1 to |V|

for j from 1 to |V|

if dist[i][j] > dist[i][k] + dist[k][j] then

dist[i][j] = dist[i][k] + dist[k][j]

next[i][j] = next[i][k]

End Procedure

### Display graph

Public Procedure DisplayGraph(Graph input)

foreach (Node n in input.nodes)

Control btn = AddControl(Button)

btn.text = n.name

btn.tags.Add(GraphPart)

nextfor

foreach (Node n in input.nodes)

Control nodeBtn = FindControlWithText(n.name)

foreach (Arc a in n.connections)

DrawLine(nodeBtn, FindControlWithText(a.destination.name))

Control arcLbl = AddControl(Label)

arcLbl.text = a.name + “ weight: ” + a.weight

arcLbl.position = (nodeBtn.position + FindControlWithText(a.destination.name).position)/2

arcLbl.tags.Add(EdgeLabel)

arcLbl.tags.Add(GraphPart)

nextfor

nextfor

End Procedure

### Update graph

Public Procedure UpdateGraph(Graph input)

DeleteAllControlsWithTag(EdgeLabel)

// Check for new nodes and redraw edges

foreach (Node n in input)

if (FindControlWithText(n.name) = null)

Control btn = AddControl(Button)

btn.text = n.name

btn.position = mouse.position

btn.tags.Add(GraphPart)

endif

foreach (Arc a in n.connections)

DrawLine(nodeBtn, FindControlWithText(a.destination.name))

Control arcLbl = AddControl(Label)

arcLbl.text = a.name + “ weight: ” + a.weight

arcLbl.position = (nodeBtn.position + FindControlWithText(a.destination.name).position)/2

arcLbl.tags.Add(EdgeLabel)

arcLbl.tags.Add(GraphPart)

nextfor

nextfor

// Check for deleted nodes

foreach (Button b in Form)

if (input.nodes.Contains(b.text) = false)

b.delete()

endif

nextfor

End Procedure

### Node constructor

Public Constructor Node(string name)

int copies = 0

if (Graph.nodes.Contains(name)

copies++

endif

while (Graph.nodes.Contains(name + “ (” + copies + “)”

copies++

endwhile

if (copies > 0)

name += “ (” + copies + “)”

endif

this.name = name

End Constructor

### Join nodes

Public Procedure JoinTo(Node destination, double weight)

Arc outgoing = new Arc(“”, destination, weight)

Arc incoming = new Arc(“”, This, weight)

connections.Add(outgoing)

destination.connections.Add(incoming)

End Procedure

### Arc constructor

Public Constructor Arc(string name, Node destination, double weight)

This.name = name

This.destination = destination

This.weight = weight

End Constructor

### Get arc weight

Public Function GetWeight()

Return weight

End Function

### Set arc weight

Public Procedure SetWeight(double val)

if (val >= 0)

weight = val

Graph.minWeight = Min(val, Graph.minWeight)

endif

End Procedure

### Get arc name

Public Function GetName()

Return name

End Function

### Set arc name

Public Procedure SetName(string name)

if (name.length < 31)

this.name = name

endif

End Procedure

### Node click handler

Public Procedure NodeClick(sender userMouse)

Switch selectedRdBtn:

case: 0

// This means “Create mode”

if (statusStrip.text.Contains = “Select the node to connect to:”)

string[] parsedText = statusStrip.text.Split(‘:’)

node.JoinTo(parsedText[1], 1)

else

statusStrip.text = “Select the node to connect to:” + node.name

endif

break

case: 1

// This means “Edit mode”

OpenDlg(nodeEditDlg)

break

case: 2

// This means “Delete mode”

foreach (Arc a in node.connections)

a.destination.connections.Remove(a)

node.connections.Remove(a)

nextfor

graph.nodes.Remove(node)

This.delete()

break

End Switch

End Procedure

### Edge click handler

Public Procedure EdgeClick(sender userMouse)

Switch selectedRdBtn:

case: 0

// This means “Create mode”

break;

case: 1

// This means “Edit mode”

OpenDlg(edgeEditDlg)

break;

case: 2

// This means “Delete mode”)

arc.destination.connections.Remove(arc)

node.connections.Remove(arc)

break;

End Switch

End Procedure

### Dark theme toggle

Public Procedure DarkTheme()

if (themeIsDark = false)

color1 = Color.Black

color2 = Color.SlateGray

color3 = Color.DimGray

textCol = Color.White

else

color1 = Color.ControlDark

color2 = Color.Control

color3 = Color.ControlLight

textCol = Color.Black

endif

foreach(Control c in Form)

c.backColor = color1

c.foreColor = color3

c.text.color = textCol

nextfor

Form.backColor = color2

Form.foreColor = color3

themeIsDark = !themeIsDark

End Procedure

### Font size toggle

Public Procedure ToggleFont()

if (fontIsLarge = false)

font = “Arial Black, 20pts”

else

font = “Helvetica, 12pts”

endif

foreach(Control c in Form)

c.text.font = font

nextfor

fontIsLarge = !fontIsLarge

End Procedure

### Text shortener

Public Procedure shortenText(string text)

string[] words = text.Split[“ ”]

string output = “”

foreach (string word in words)

output+= word[0].ToUpper()

nextfor

Return output

End Procedure

### Zoom

Public Procedure Zoom(float scale)

foreach (Control c in Form)

if (c.tags.Contains(GraphPart))

Point centre = new Point(form.width/2, form.height/2)

c.location = c.location \* scale – (centre\*(scale – 1))

// The formula for scaling a point around the centre, where S is the scale factor, P is the point and C is the centre is P’ = sP – (1-s)C

endif

nextfor

UpdateGraph(Graph)

End Procedure

## Data Structures



These data structures are useful for abstraction in that they simplify access to the relevant parts of the graph. It is more human readable to access, for example, graph.nodes[0].connections[0] than an enormous 2D array. They are also important for validation so that access to attributes is controlled through methods so validation will always take place.

## Usability Features

### Layout

Here is a mock-up for the UI. The final program will look similar to this, however some icons and shapes may change





This shows how the program may look with the large text mode and dark theme (both separately togglable). Note: I was not able to change the colour of “Barton-upon-Humber” from orange



### Stakeholder feedback

**Garcia**: The overall layout looks fantastic, however I am concerned about the text cluttering up the interface if lots of nodes are near to each other.

**Weston**: I think it looks good but it is not easy to tell apart the weights of the two middle arcs shown

**Thorne**: Looks great in my opinion

After some discussion with Esteban, we decided that this issue could be overcome with an algorithm that abbreviates the text as the zoom level increases. Unfortunately, I was not able to come to a solution for the issue raised by Paula

## Test Data

**Graph A:**

Riga

Dreilini

Mārupe

Valdlauči

Kekava

Saurieši

**Graph B:**

8

1

6

3

4

2

5

Cammeringham Hill

Barton-upon-Humber

Álta

B

5

2

1

3

8

9



Dallas

Austin

St Louis

Memphis

Houston

New Orleans

San Antonio

**Graph C:**

4

105

53

34

### Iteration 1

Prototype 1.0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | Description | Test data | Type | Expected result | Linked success criteria |
| 1.0 | Attempting to load the form | None | None | The form loads and displays controls | All |

Protype 1.1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.1 | Attempt to open options | Click on options button | Valid | Options dialogue box is shown | 8 |

Protype 1.2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.2.0 | Dark theme set | Dark theme option chosen | Valid | All form and control colours are changed to a darker colour whereas text is made brighter | 8 |
| 1.2.1 | Dark theme unset | Dark them option unset after being chosen | Valid | The effect of the previous test is reverted | 8 |
| 1.2.2 | Large text enabled | Large text option chosen | Valid | Text in the status bar and dropdown box has its size and font changed | 8 |
| 1.2.3 | Large text disabled | Large text option unset after being set | Valid | Text in the status bar and dropdown box has its size and font changed | 8 |

### Iteration 2

Protype 2.0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2.0.0a | Testing various graphs | A single node named “Cottonshopeburnfoot” | Valid | A box is drawn with the text “Cottonshopeburnfoot” inside, the text may be outside the bounds of the shape | 5 |
| 2.0.0b | This node has a node joined to it named “A” automatically and the edge is assigned weight 5 and name “M2” | Valid | Alongside the box, a straight line is drawn to another equally sized box with the text “A”. Text above the line reads: “M2, weight 5” | 3,5 |
| 2.0.0c | Another node is joined with an unnamed and unweighted edge. The node is named “B” automatically | Valid | Another circle is joined with a line to node A and contains the text “B” | 3,5 |
| 2.0.1a | Testing different manipulations of the graphs | The graph described above will be arranged so that at least one edge overlaps with another | Valid | The lines will not change other than position | 1 |
| 2.0.1b | Two of the nodes will be made to overlap | Valid | The selected node will display on top of the one beneath | 1 |

Protype 2.1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2.1.0 | A test of editing the nodes and edges | With edit mode selected, the user clicks on a node and names it “áéíóúñ”, then an edge and names it “Спасибо” | Valid | Unicode is displayed with the same font, colour, size and spacing | 1,5,6 |
| 2.1.1 | Weight testing | The “Спасибо” edge is assigned weight 1.5 | Valid | The edge now reads: “Спасибо, weight: 1.5” | 5 |

Protype 2.2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2.2.0 | No movement | A click and drag on anywhere other than the nodes | Invalid | No movement | 1 |
| 2.2.1 | Normal movement | A node is clicked and dragged from some point to another | Valid | The nodes and edges and related text update their positions | 1 |
| 2.2.2a | Unusual movement | A node is clicked and dragged off the window | Boundary | When the mouse returns to the screen, the node resumes following it | 1 |
| 2.2.2b | A node is placed on top of the node it is connected to | Boundary | The last moved node is drawn on top of the one beneath and the line cannot be seen | 1 |
| 2.2.3 | Node deletion | With delete mode selected, all nodes are clicked on | Valid | The node and associated edges are removed | 1 |
| 2.2.4 | Node addition | With add mode selected, 3 clicks are made in the graph area | Valid | 3 nodes are created and automatically named sequentially | 1 |
| 2.2.5 | Edge deletion | With delete mode selected, 3 edges are clicked at their centre | Valid | The edges and their text are removed | 1 |
| 2.2.6 | Valid edge creation | With add mode selected, two nodes are clicked | Valid | A line is drawn between the two nodes with no text | 1 |

Protype 2.3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2.3.0 | Invalid edge creation | In create mode, the same node is clicked twice | Invalid | A message is shown in the status bar stating that edge creation was cancelled and why | 1,7 |
| 2.3.1 | Boundary node name | A node is named “Cuartero Cerámica y Mosaico SL” (30) | Valid Boundary | The node’s name and text is successfully updated | 1,5 |
| 2.3.2 | Invalid node name | A node is named “Universidad CE Cardenal Herrera” (31) | Invalid Boundary | A message pops up telling the user that the max length is 30 and the node name is changed to “Universidad CE Cardenal Herrer” | 1,5 |
| 2.3.4 | Boundary weight | An edge is assigned weight 0 | Valid Boundary | The weight is changed successfully to 0 | 1 |
| 2.3.5 | Invalid weight | An edge is assigned weight -1 | Invalid | The user is shown a popup that the minimum weight must be 0 and the edge is unchanged | 1 |
| 2.3.6 | Multiple nodes with the same name | A second node is created called “Cuartero Cerámica y Mosaico SL” | Boundary | The node is renamed “Cuartero Cerámica y Mosaico SL (1)” | 1 |

Prototype 2.4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2.4.0 | Furthest out zoom | The zoom level is changed to its minimum on **Graph A, B, and C** separately | Valid | The size of the circles representing nodes and the width of edges is reduced and the names are shortened | 12 |
| 2.4.1 | Closest zoom in | The zoom level is changed to its maximum on **Graph A, B, and C** separately | Valid | The size of the circles representing nodes and the width of edges is increased and the names are returned to their originals | 12 |

### Iteration 3

Protype 3.0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3.0.0 | Testing saving | The save button is pressed while each graph is displayed (separately) | Valid | A file dialogue is opened and the user can save a file with a name and location | 4 |
| 3.0.1a | Testing loading on a blank graph | The load button is pressed while no graph is displayed | Valid | A file dialogue allows the user to open a file that is of the required file type | 4 |
| 3.0.1b | Testing loading on an existing graph | Load button is pressed while **graph C** is displayed | Valid | Before the file dialogue is opened, a message asks the user whether they would like to cancel the operation | 4 |
| 3.0.2 | A graph with the invalid file type is loaded | The user selects this document as the graph to load | Invalid | The operation is cancelled and a message in the status bar explains why | 4,7 |

Protype 3.1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3.1.0 | Unexpected close with **graph B** loaded | **Graph B** is loaded from a file, altered, and then after 5 minutes (autosave time) the program is closed using Task Manager | Invalid | When the program is next opened, a message tells the user what happened and where to find the autosave, which will be named \_NAME\_+autosave | 7,9 |

### Iteration 4

Protype 4.0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.0.0 | Dijkstra’s on various graphs | The first node shall be the leftmost node and the destination the rightmost valid node for each graph (A, B, and C) tested | Valid | For graph A: Barton-upon-Humber, Álta, B, length: 3  For B: Mārupe, Valdlauči, Dreilini, Saurieši, length: 9  For C: Austin, Dallas, St Louis, length 59 | 2.1 |
| 4.0.1 | A\* on various graphs | The first node shall be the leftmost node and the destination the rightmost valid node for each graph (A, B, and C) tested | Valid | For graph A: Barton-upon-Humber, Álta, B, length: 3  For B: Mārupe, Valdlauči, Dreilini, Saurieši, length: 9  For C: Austin, Dallas, St Louis, length 59 | 2.2 |

Protype 4.1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.1.0 | Kruskal’s on various graphs | Graphs A, B, and C shall be tested separately | Valid | The minimum spanning tree will be highlighted in red | 2.3 |
| 4.1.1 | Floyd’s on various graphs | Graphs A, B, and C shall be tested separately | Valid | A table of least distances will be shown for each graph | 2.5 |

Protype 4.2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.2.0 | Travelling salesman on a valid graph | Graphs A shall be tested separately starting from Barton-Upon-Humber | Valid | **Route for A:** Barton-upon-Humber, Cammeringham Hill, B, Alta, Barton-upon-Humber, length: 11 | 2.4 |

Protype 4.3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.3.0a | Dijkstra’s where no route is possible | Dijkstra’s is run on **graph C** starting at Dallas with a destination of Houston | Invalid | An error message is displayed stating that there are no routes between the two nodes | 2.1, 7, 13 |
| 4.3.0b | Dijkstra’s starting and ending at the same node | Dijkstra’s is run on **graph B** starting at Riga with a destination of Riga | Valid Boundary | The shortest path is given as “Riga, length 0” | 2.1 |
| 4.3.0c | Dijkstra’s where the start node is invalid | The start node for **graph A** is given as “Australia” | Invalid | An error message is displayed stating that “Australia” does not exist | 2.1,7,13 |
| 4.3.1a | A\* where no route is possible | A\* is run on **graph C** starting at Dallas with a destination of Houston | Invalid | An error message is displayed stating that there are no routes between the two nodes | 2.2, 7, 13 |
| 4.3.1b | A\* starting and ending at the same node | A\* is run on **graph B** starting at Riga with a destination of Riga | Valid Boundary | The shortest path is given as “Riga, length 0” | 2.2 |
| 4.3.1c | A\* where the start node is invalid | The start node for **graph B** is given as “Álta” | Invalid | An error message is displayed stating that “Álta” does not exist | 2.2,7,13 |
| 4.3.2 | Travelling salesman on a graph with no Hamiltonian cycles | Travelling salesman is run on **Graph B** and then **C** | Invalid | An error message is displayed explaining that no routes exist for the problem | 2.4, 7, 13 |

## Post development

Once development is complete, I will use performance modelling to determine whether success criteria 10 has been achieved.

After that, I will give my stakeholders this questionnaire to assess how well the program was suited to their needs and the rest of my success criteria

**What was the most useful part of the program for you?**

**Were you able to access every part of the program?**

**Did you encounter any errors? If so, were you able to solve the problem(s) or did they reoccur?**

**Are there any features you felt were missing?**

**Which parts of the program did you spend the most time using? Could these be sped up?**

**How responsive was it on your machine?**

**Rate out of 10 each of the algorithms for their usefulness**

**Rate out of 10 how intuitive using the program was**

**Rate out of 10 the robustness and stability of the program**

**Out of 10, how likely are you to use the program again? Was it useful to you?**

**(For Esteban) Were you able to understand what the program did from icons alone?**

**(For Paula) Was the program accessible to you?**

**(For Harry) Did you know what to expect from each graph algorithm before testing them?**

# Implementation

### Iteration 1 testing:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Type | Expected result | Linked success criteria | Result | Action taken |
| 1.0 | Attempting to load the form | None | None | The form loads and displays controls | All | Success  (Screenshot 1.0) | None |
| 1.1 | Attempt to open options | Click on options button | Valid | Options dialogue box is shown | 8 | Success  (Screenshot 1.1) | None |
| 1.2.0 | Dark theme set | Dark theme option chosen | Valid | All form and control colours are changed to a darker colour whereas text is made brighter | 8 | Failure, not all controls were changed (Screenshot 1.2.0)  Code is below screenshot | Additional loop added to catch controls inside controls |
| 1.2.1 | Dark theme unset | Dark them option unset after being chosen | Valid | The effect of the previous test is reverted | 8 | Failure, the colour of some controls was not reverted to its original (Screenshot 1.2.1) | If statement specifying the controls to have a different colour |
| 1.2.2 | Large text enabled | Large text option chosen | Valid | Text in the status bar and dropdown box has its size and font changed | 8 | Success (Screenshot 1.2.2) | None |
| 1.2.3 | Large text disabled | Large text option unset after being set | Valid | Text in the status bar and dropdown box has its size and font changed | 8 | Success (Screenshot 1.2.3) | None |

### Iteration 1 evidence:



Screenshot 1.0 The duck images act as placeholders for the icons I will draw later.

These are the icons I created which will provide a visual explanation for most of the features



Screenshot 1.1



Screenshot 1.2.0, the dropdown box has not had its colours changed.

private void itemChecked(object sender, ItemCheckEventArgs e)

{

// Called before the item is actually checked/unchecked

switch (optionsBox.SelectedIndex)

{

case 1:

// Dark theme

// Default colours:

Color backColour = SystemColors.Control;

Color textColour = SystemColors.ControlText;

if (e.CurrentValue == CheckState.Unchecked)

{

// Dark colours:

backColour = SystemColors.ControlDarkDark;

textColour = SystemColors.ControlLight;

}

foreach (Form f in Application.OpenForms)

{

f.BackColor = backColour;

foreach (Control c in f.Controls)

{

if (c.Name == "darkThemeIndicator")

{

c.BackColor = textColour;

}

else

{

c.BackColor = backColour;

c.ForeColor = textColour;

}

}

}

break;

}

### Code added as a result of error:

foreach (Control nestedControl in c.Controls)

{

nestedControl.BackColor = backColour;

nestedControl.ForeColor = textColour;

}



Screenshot 1.2.1, the dropdown box was originally white

### Code added as a result of error:

private void SetControlColours(Control control, Color backColour, Color textColour)

{

// This is a list of all controls that must have a white back colour

if ((control.Name == "cbxAlgorithmSelect" || control.Name == "optionsBox") && control.BackColor != SystemColors.Window)

{

control.BackColor = SystemColors.Window;

}

else

{

control.BackColor = backColour;

}

control.ForeColor = textColour;

}

\*This if statement was later changed to the below as the error had reocurred

if ((control.Name == "cbxAlgorithmSelect" || control.Name == "optionsBox" || control.Name == "") && !themeIsDark)



Screenshot 1.2.2

case 0:

// Large text

// Default font

Font font = new Font("Segoe UI", 9, FontStyle.Regular);

textIsLarge = false;

if (e.CurrentValue == CheckState.Unchecked)

{

// Large font

font = new Font("Arial", 16, FontStyle.Bold);

textIsLarge = true;

}

foreach (Form f in Application.OpenForms)

{

// Checks if any open form is a mainForm, if so, change its fonts

try

{

((mainForm)f).ChangeFonts(font);

}

catch (Exception error)

{ }

}

break;

/// <summary>

/// Changes the fonts of some controls

/// </summary>

/// <param name="font">The font to be changed to</param>

public void ChangeFonts(Font font)

{

cbxAlgorithmSelect.Font = font;

statusLabel.Font = font;

}



Screenshot 1.2.3, same code as above

### Iteration 1 stakeholder feedback and response

My stakeholders’ feedback was positive overall; however, one complaint was that options were not saved when the program was closed. To address this, I added the following procedure that is called when the CheckState for any option is changed

/// <summary>

/// Saves options to a config file

/// </summary>

/// <param name="cfgFilePath">The location of the file</param>

private void WriteOpsToFile(string cfgFilePath)

{

string data = "";

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

{

// The condition after the '^' (XOR operator) is necessary due to the fact this is called before the item is actually checked/unchecked

if (optionsBox.GetItemCheckState(i) == CheckState.Checked ^ optionsBox.SelectedIndex == i)

{

data += "1";

}

else

{

data += "0";

}

data += "\n";

}

File.WriteAllText(cfgFilePath, data);

}

The next section of code applies the options found in the file

private void OptionsLoaded(object sender, EventArgs e)

{

// When this form opens, check the boxes for the options that were saved in the file

string[] storedOps = File.ReadAllLines(cfgFilePath);

for (int i = 0; i < storedOps.Length; i++)

{

if (storedOps[i] == "1")

{

// Have to mimic a user clicking the boxes so that changes are handled correctly

optionsBox.SelectedIndex = i;

ItemChecked(optionsBox, new ItemCheckEventArgs(i, CheckState.Checked, CheckState.Unchecked));

optionsBox.SetItemCheckState(i, CheckState.Checked);

}

}

}

### Iteration 2 testing:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Type | Expected result | Linked success criteria | Result | Action taken |
| 2.0.0a | Testing various graphs | A single node named “Cottonshopeburnfoot” | Valid | A box is drawn with the text “Cottonshopeburnfoot” inside, the text may be outside the bounds of the shape | 5 | Success  (Screenshot 2.0.0a) | None |
| 2.0.0b |  | This node has a node joined to it named “A” automatically and the edge is assigned weight 5 and name “M2” | Valid | Alongside the box, a straight line is drawn to another equally sized box with the text “A”. Text above the line reads: “M2, weight 5” | 3,5 | Failure (Screenshot 2.0.0b) | Added space and comma to display code |
| 2.0.0c | Another node is joined with an unnamed and unweighted edge. The node is named “B” automatically | Valid | Another box is joined with a line to node A and contains the text “B” | 3,5 | Failure (Screenshot 2.0.0c) | Added missing increment code |
| 2.0.1a | Testing different manipulations of the graphs | The graph described above will be arranged so that at least one edge overlaps with another | Valid | The lines will not change other than position | 1 | Success (Screenshot 2.0.1a) | None |
| 2.0.1b | Two of the nodes will be made to overlap | Valid | The selected node will display on top of the one beneath | 1 | Success (Screenshot 2.0.1b) | None |
| 2.1.0 | A test of editing the nodes and edges | With edit mode selected, the user clicks on a node and names it “áéíóúñ”, then an edge and names it “Спасибо” | Valid | Unicode is displayed with the same font, colour, size and spacing | 1,5,6 | Success (Screenshot 2.1.0) | None |
| 2.1.1 | Weight testing | The “Спасибо” edge is assigned weight 1.5 | Valid | The edge now reads: “Спасибо, weight: 1.5” | 5 | Success (Screenshot 2.1.1) | None |
| 2.2.0 | No movement | A click and drag on anywhere other than the nodes | Invalid | No movement | 1 | Success (No screenshot) | None |
| 2.2.1 | Normal movement | A node is clicked and dragged from some point to another | Valid | The nodes and edges and related text update their positions | 1 | Success (No screenshot) | None |
| 2.2.2a | Unusual movement | A node is clicked and dragged off the window | Boundary | When the mouse returns to the screen, the node resumes following it | 1 | Success (No screenshot) | None |
| 2.2.2b | A node is placed on top of the node it is connected to | Boundary | The last moved node is drawn on top of the one beneath and the line cannot be seen | 1 | Success (Screenshot 2.2.2b) | None |
| 2.2.3 | Node deletion | With delete mode selected, all nodes are clicked on | Valid | The node and associated edges are removed | 1 | Failure (Screenshot 2.2.3) | For loop added to delete each connection |
| 2.2.4 | Node addition | With add mode selected, 3 clicks are made in the graph area | Valid | 3 nodes are created and automatically named sequentially | 1 | Success (Screenshot 2.2.4) | None |
| 2.2.5 | Edge deletion | With delete mode selected, 3 edges are clicked at their centre | Valid | The edges and their text are removed | 1 | Success (Screenshot 2.2.5) | None |
| 2.2.6 | Valid edge creation | With add mode selected, two nodes are clicked | Valid | A line is drawn between the two nodes with no text | 1 | Success (Screenshot 2.2.6) | None |
| 2.3.0 | Invalid edge creation | In create mode, the same node is clicked twice | Invalid | A message is shown in the status bar stating that edge creation was cancelled and why | 1,7 | Success (Screenshot 2.3.0) | None |
| 2.3.1 | Boundary node name | A node is named “Cuartero Cerámica y Mosaico SL” (30) | Valid Boundary | The node’s name and text is successfully updated | 1,5 | Success (Screenshot 2.3.1) | None |
| 2.3.2 | Invalid node name | A node is named “Universidad CE Cardenal Herrera” (31) | Invalid Boundary | A message pops up telling the user that the max length is 30 and the node name is changed to “Universidad CE Cardenal Herrer” | 1,5 | Success (Screenshot 2.3.2) | None |
| 2.3.4 | Boundary weight | An edge is assigned weight 0 | Valid Boundary | The weight is changed successfully to 0 | 1 | Success (Screenshot 2.3.4) | None |
| 2.3.5 | Invalid weight | An edge is assigned weight -1 | Invalid | The user is shown a popup that the minimum weight must be 0 and the edge is unchanged | 1 | Failure (No screenshot, the weight was set to 0) | Validation added inside EditEdge |
| 2.3.6 | Multiple nodes with the same name | A second node is created called “Cuartero Cerámica y Mosaico SL” | Boundary | The node is renamed “Cuartero Cerámica y Mosaico SL (1)” | 1 | Failure (Screenshot 2.3.6) | EditNode redone to take advantage of constructor validation |
| 2.4.0 | Furthest out zoom | The zoom level is changed to its minimum on **Graph A, B, and C** separately | Valid | The size of the circles representing nodes and the width of edges is reduced and the names are shortened | 12 | Success (Screenshot 2.4.0) | None |
| 2.4.1 | Closest zoom in | The zoom level is changed to its maximum on **Graph A, B, and C** separately | Valid | The size of the circles representing nodes and the width of edges is increased and the names are returned to their originals | 12 | Success (Screenshot 2.4.1) | None |

### Iteration 2 evidence:

Initial code for classes:

namespace GraphManager

{

public class Arc

{

private string name;

private double weight;

public Node destination;

public Arc(string name, Node destination, double weight)

{

this.name = name;

this.destination = destination;

this.weight = weight;

}

/// <summary>

/// Validates string given as name, then sets it if it passes

/// </summary>

/// <param name="newName">Desired name</param>

public void SetName(string newName)

{

// Length check

if (newName.Length < 31)

{

name = newName;

}

}

/// <summary>

/// Returns the name of the arc as a string

/// </summary>

/// <returns></returns>

public string GetName()

{

return name;

}

/// <summary>

/// Validates double given as weight, then sets it if it passes

/// </summary>

/// <param name="newWeight">Desired weight</param>

public void SetWeight(double val)

{

// Range check

if (val >= 0)

{

weight = val;

}

}

/// <summary>

/// Returns the weight of the arc as a double

/// </summary>

/// <returns></returns>

public double GetWeight()

{

return weight;

}

}

}

namespace GraphManager

{

public class Node

{

public string name;

public List<Arc> connections = new List<Arc>();

public System.Drawing.Point location;

public Node(Graph parentGraph, string name, System.Drawing.Point location)

{

if (name == "")

{

// Automatically assigns nodes with names A-Z, but if there are more than 26 they are named A1-Z1, then A2-Z2 and so on

if (parentGraph.nodeID > 25)

{

this.name = (char)((parentGraph.nodeID % 26) + 65) + (parentGraph.nodeID / 26).ToString();

}

else

{

this.name = ((char)(parentGraph.nodeID + 65)).ToString();

}

}

else

{

int copies = 0;

string copyName = name;

this.name = copyName;

while (parentGraph.nodes.Contains(this))

{

copies++;

copyName = name + " (" + copies + ")";

this.name = copyName;

}

}

// Range check for position (short hand for if location < 0, location = 0)

this.location.X = Math.Max(0, location.X);

this.location.Y = Math.Max(0, location.Y);

}

// I have overriden this function to change the behaviour of .Contains so that it returns true if a node is found with the same name as the parameter

// This post guided me to this solution: https://stackoverflow.com/questions/1076350/modify-list-contains-behavior

public override bool Equals(object obj)

{

if (obj is Node)

{

return ((Node)obj).name == name;

}

else

{

return base.Equals(obj);

}

}

/// <summary>

/// Join this node to another, updating the other node. This procedure is not validated

/// </summary>

/// <param name="n">Destination node</param>

/// <param name="weight">Connection weight</param>

public void JoinTo(Node n, double weight)

{

Arc outgoing = new Arc("", n, weight);

Arc incoming = new Arc("", this, weight);

connections.Add(outgoing);

n.connections.Add(incoming);

}

}

}

namespace GraphManager

{

public class Graph

{

private bool wasSaved;

public int nodeID;

public double minWeight;

public List<Node> nodes = new List<Node>();

}

}



Screenshot 2.0.0a

// !!Testing code!!

Graph graph = new Graph();

graph.nodes.Add(new Node(graph, "Cottonshopeburnfoot", new Point(500, 500)));

DisplayGraph(graph);

public void DisplayGraph(Graph input)

{

//Draws lines between each node

activeGraph = input;

this.Invalidate();

foreach (Node n in input.nodes)

{

// Create button to represent node

Button btnNode = new Button

{

Location = n.location,

Text = n.name,

Tag = "Graph Part",

Size = new Size(234, 45)

};

Controls.Add(btnNode);

// Draw arcs to each connected node

// Each will be drawn twice due to the to/from nature of connections

foreach (Arc a in n.connections)

{

Button btnArc = new Button

{

Text = a.GetName() + "weight = " + a.GetWeight(),

// Centerpoint between nodes

Location = new Point((btnNode.Location.X + a.destination.location.X) / 2,

(btnNode.Location.Y + a.destination.location.Y) / 2),

Tag = "Edge Label",

Size = new Size(50,45)

};

Controls.Add(btnArc);

}

}

}

// Called on "this.Invalidate()", clears the canvas each time

protected override void OnPaint(PaintEventArgs e)

{

// Source: https://stackoverflow.com/questions/47239557/c-sharp-how-do-i-draw-a-line-between-two-objects-on-a-windows-form

base.OnPaint(e);

using (var blackPen = new Pen(Color.Black, 3))

{

foreach (Node n in activeGraph.nodes)

{

foreach (Arc a in n.connections)

{

e.Graphics.DrawLine(blackPen, n.location, a.destination.location);

}

}

}

}



Screenshot 2.0.0b, the edge label was incorrectly formatted

The JoinTo() procedure had to be changed to add a “name” argument to name the arc

### Code added as a result of error:

Inside DisplayGraph():

Text = a.GetName() + " weight = " + a.GetWeight(),

A comma and a space were added before the “w”

However, this resulted in the text being cut off, so I increased the size of the generated button



Screenshot 2.0.0c, the node was also named ‘A’

Code added:

if (a.GetWeight() == 0)

{

btnArc.Text = "\_";

}

else if (a.GetName() == "")

{

btnArc.Text = "Weight = " + a.GetWeight();

}

else

{

btnArc.Text = a.GetName() + ", weight = " + a.GetWeight();

}

### Code added as a result of error:

I had forgotten to add this line inside the node constructor:

parentGraph.nodeID++;



Screenshot 2.0.1a



Screenshot 2.0.1b

Prototype 2.1

It was at this point that I realised I would need to make some changes to my design for the Arc class to allow for manipulation capability. One major change was that arcs will no longer be two per connection, which makes future development of directional arcs slightly more difficult, but vastly simplifies UI and management code. Another change is that arcs now have a unique ID, set by the constructor and maintained by the mainForm object. This is so that buttons on the UI can be associated with the arcs in code.

Changes to arc class and resultant changes to node constructor:





+some small syntax changes elsewhere



Screenshot 2.1.0

Code used:

(In mainform)

private void HandleNodeClick(object sender, EventArgs e)

{

Button btnSender = sender as Button;

activeNode = FindNodeWithName(activeGraph.nodes, btnSender.Text);

if (rdbCreate.Checked)

{

// Create mode

if (statusLabel.Text.Contains("Select the node to connect to:"))

{

string[] parsedText = statusLabel.Text.Split(':');

activeNode.JoinTo(FindNodeWithName(activeGraph.nodes, parsedText[1]), "", 0, ref IDCount);

statusLabel.Text = "";

DisplayGraph(activeGraph);

}

else

{

statusLabel.Text = "Select the node to connect to:" + activeNode.name;

}

}

else if (rdbEdit.Checked)

{

// Edit mode

NodeEditDialogue editWindow = new NodeEditDialogue();

editWindow.ShowDialog(this);

}

else if (rdbDelete.Checked)

{

// Delete mode

}

else

{

// This should never happen, but if it does, ignore it.

}

}

/// <summary>

/// Returns a node from a list of nodes that has the name specified. Returns null and displays a message if not found

/// </summary>

/// <param name="nodes">The list to be searched</param>

/// <param name="name">The name the node should have</param>

/// <returns></returns>

public Node FindNodeWithName(List<Node> nodes, string name)

{

foreach (Node node in nodes)

{

if (node.name == name)

{

return node;

}

}

MessageBox.Show("NODE NOT FOUND");

return null;

}

private void HandleEdgeClick(object sender, EventArgs e)

{

Button btnSender = sender as Button;

if (rdbCreate.Checked)

{

// Create mode

}

else if (rdbEdit.Checked)

{

// Edit mode

activeEdge = FindArcFromID(Convert.ToInt32(btnSender.Name));

EdgeEditDialogue editWindow = new EdgeEditDialogue();

editWindow.ShowDialog(this);

}

else if (rdbDelete.Checked)

{

// Delete mode

activeEdge = FindArcFromID(Convert.ToInt32(btnSender.Name));

activeEdge.between[0].connections.Remove(activeEdge);

activeEdge.between[0] = null;

activeEdge.between[1].connections.Remove(activeEdge);

activeEdge.between[1] = null;

// Hopefully won't result in a memory leak

Controls.Remove(btnSender);

activeEdge = null;

DisplayGraph(activeGraph);

}

else

{

// This should never happen, but if it does, ignore it.

}

}

/// <summary>

/// Takes an integer input and returns the arc with that ID

/// </summary>

/// <param name="id">The ID the arc should have</param>

/// <returns></returns>

private Arc FindArcFromID(int id)

{

// O(N^2) complexity here is bad, should be improved in future, perhaps with indexing

foreach (Node n in activeGraph.nodes)

{

foreach (Arc a in n.connections)

{

if (a.ID == id)

{

return a;

}

}

}

return null;

}

/// <summary>

/// Clears all graph related controls and draws a new graph using paint and buttons

/// </summary>

/// <param name="input">The graph that should be drawn</param>

public void DisplayGraph(Graph input)

{

// Clear existing graph

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

{

Control c = this.Controls[i];

if ((string)c.Tag == "Graph Part" || (string)c.Tag == "Edge Label")

{

this.Controls.Remove(c);

i--;

}

}

// Draws lines between each node

activeGraph = input;

this.Invalidate();

foreach (Node n in input.nodes)

{

// Create button to represent node

Button btnNode = new Button

{

Location = n.location,

Text = n.name,

Tag = "Graph Part",

Size = new Size(234, 45)

};

btnNode.Click += new EventHandler(HandleNodeClick);

Controls.Add(btnNode);

// Draw arcs to each connected node

foreach (Arc a in n.connections)

{

// Prevents two buttons per arc by checking if one already exists

// (ContainsKey checks for a control with the given name)

if (!Controls.ContainsKey(a.ID.ToString()))

{

Button btnArc = new Button

{

// Centerpoint between nodes

Location = new Point((a.between[0].location.X + a.between[1].location.X) / 2,

(a.between[0].location.Y + a.between[1].location.Y) / 2),

Tag = "Edge Label",

Size = new Size(75, 75),

Name = a.ID.ToString()

};

// When this button is clicked, call the HandleEdgeClick procedure

btnArc.Click += new EventHandler(HandleEdgeClick);

if (a.GetWeight() == 0)

{

if (a.GetName() == "")

{

btnArc.Text = "\_";

}

else

{

btnArc.Text = a.GetName();

}

}

else if (a.GetName() == "")

{

btnArc.Text = "Weight = " + a.GetWeight();

}

else

{

btnArc.Text = a.GetName() + ", weight = " + a.GetWeight();

}

Controls.Add(btnArc);

}

}

}

}

/// <summary>

/// Changes various properties of an the active edge

/// </summary>

/// <param name="mode">0 = Set name and weight, 1 = Set name, 2 = Set weight, 3/other = None</param>

/// <param name="newName">The name that should be set or not</param>

/// <param name="newWeight">The weight that should be set or not</param>

public void EditEdge(int mode, string newName, double newWeight)

{

switch (mode)

{

case 0:

// Change both

activeEdge.SetName(newName);

activeEdge.SetWeight(newWeight);

break;

case 1:

// Change name

activeEdge.SetName(newName);

break;

case 2:

// Change weight

activeEdge.SetWeight(newWeight);

break;

case 3:

// Do nothing

break;

}

DisplayGraph(activeGraph);

activeEdge = null;

}

/// <summary>

/// Sets the active node's name to the string

/// </summary>

/// <param name="newName">The name that should be set</param>

public void EditNode(string newName)

{

activeNode.name = newName;

DisplayGraph(activeGraph);

activeNode = null;

}

(In NodeEditDialogue)

namespace GraphManager

{

public partial class NodeEditDialogue : Form

{

public NodeEditDialogue()

{

InitializeComponent();

}

private void CloseDialogue(object sender, EventArgs e)

{

if (tbxName.Text == "")

{

((mainForm)Owner).activeEdge = null;

Close();

}

else

{

((mainForm)Owner).EditNode(tbxName.Text);

Close();

}

}

}

}

(In EdgeEditDialogue)

namespace GraphManager

{

public partial class EdgeEditDialogue : Form

{

public EdgeEditDialogue()

{

InitializeComponent();

}

private void CloseDialogue(object sender, EventArgs e)

{

if (tbxName.Text == "" && tbxWeight.Text == "")

{

((mainForm)Owner).activeEdge = null;

Close();

}

else if(tbxName.Text == "")

{

// Type check

try

{

double weight = Convert.ToDouble(tbxWeight.Text);

((mainForm)Owner).EditEdge(2, "", weight);

Close();

}

catch

{

MessageBox.Show("Weight must be a number");

}

}

else if(tbxWeight.Text == "")

{

((mainForm)Owner).EditEdge(1, tbxName.Text, -1);

Close();

}

else

{

// Type check

try

{

double weight = Convert.ToDouble(tbxWeight.Text);

((mainForm)Owner).EditEdge(0, tbxName.Text, weight);

Close();

}

catch

{

MessageBox.Show("Weight must be a number");

}

}

}

}

}



Screenshot 2.1.1, same code as above

Code for tests 2.2.0 to 2.2.2:

(Inside MainForm)

private void HandleNodeDrag(object sender, EventArgs e)

{

Button btn = sender as Button;

// Keeps the location inside the bounds of the form

// 79 is the height of the two toolbars at the top

// 36 is the height of the flow layout panel at the bottom

// Location is the top left of the button, so I need to subtract its width/height from the min limits

btn.Location = new Point(Math.Min(this.Width - btn.Width, Math.Max(0, btn.Location.X)),

Math.Min(this.Height - 36 - statusStrip.Size.Height - btn.Height, Math.Max(79, btn.Location.Y)));

btn.BringToFront();

activeNode = FindNodeWithName(activeGraph.nodes, btn.Text);

activeNode.location = btn.Location;

this.Invalidate();

}

private void Main\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.G)

{

Cursor = Cursors.SizeAll;

foreach (Control c in this.Controls)

{

// Remove all click events temporarily

if ((string)c.Tag == "Graph Part" && c is Button)

{

c.Click -= new EventHandler(HandleNodeClick);

// Using the Control.Draggable NuGet package

c.Draggable(true);

}

}

}

}

private void Main\_KeyUp(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.G)

{

Cursor = Cursors.Default;

foreach (Control c in this.Controls)

{

// Restore event handlers

if ((string)c.Tag == "Graph Part" && c is Button)

{

c.Click += new EventHandler(HandleNodeClick);

c.Draggable(false);

}

}

// Refresh graph after movement is done

DisplayGraph(activeGraph);

}

}



Screenshot 2.2.2b



Screenshot 2.2.3, the node has been deleted but the edge remains

else if (rdbDelete.Checked)

{

// Delete mode

activeNode = FindNodeWithName(activeGraph.nodes, btnSender.Text);

// Delete all connections to this node

activeNode.connections = new List<Arc>();

activeGraph.nodes.Remove(activeNode);

// Refresh display

DisplayGraph(activeGraph);

}

The issue here was that the connections aren’t deleted, they are just removed from the deleted node’s list.

### Code added as a result of error:

This replaced the line above starting activeNode.connections

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

{

// Gets the node at the other end of the arc from the deleted node

Node destination = activeNode.connections[i].GetDestination(activeNode);

// Removes the arc from the destination node's list

destination.connections.Remove(activeNode.connections[i]);

}



Screenshot 2.2.4

private void BackClicked(object sender, MouseEventArgs e)

{

if (rdbCreate.Checked)

{

activeGraph.nodes.Add(new Node(activeGraph, "", e.Location));

DisplayGraph(activeGraph);

}

}



Screenshot 2.2.5

(Inside HandleEdgeClick)

else if (rdbDelete.Checked)

{

// Delete mode

activeEdge = FindArcFromID(Convert.ToInt32(btnSender.Name));

activeEdge.between[0].connections.Remove(activeEdge);

activeEdge.between[0] = null;

activeEdge.between[1].connections.Remove(activeEdge);

activeEdge.between[1] = null;

// Hopefully won't result in a memory leak

Controls.Remove(btnSender);

activeEdge = null;

DisplayGraph(activeGraph);

}



Screenshot 2.2.6

if (rdbCreate.Checked)

{

// Create mode

if (statusLabel.Text.Contains("Select the node to connect to:"))

{

string[] parsedText = statusLabel.Text.Split(':');

activeNode.JoinTo(FindNodeWithName(activeGraph.nodes, parsedText[1]), "", 0, ref IDCount);

statusLabel.Text = "";

DisplayGraph(activeGraph);

}

else

{

statusLabel.Text = "Select the node to connect to:" + activeNode.name;

}

}



Screenshot 2.3.0

if (rdbCreate.Checked)

{

// Create mode

if (statusLabel.Text.Contains("Select the node to connect to:"))

{

string[] parsedText = statusLabel.Text.Split(':');

Node destination = FindNodeWithName(activeGraph.nodes, parsedText[1]);

if (activeNode == destination)

{

statusLabel.Text = "Edge creation cancelled, you cannot join a node to itself";

}

else

{

activeNode.JoinTo(destination, "", 0, ref IDCount);

statusLabel.Text = "";

DisplayGraph(activeGraph);

}

}

else

{

statusLabel.Text = "Select the node to connect to:" + activeNode.name;

}



Screenshot 2.3.1



Screenshot 2.3.2

(Inside NodeEditDialogue)

private void Name\_TextChanged(object sender, EventArgs e)

{

if (tbxName.Text.Length == 30)

{

MessageBox.Show("Max name length is 30");

}

}



Screenshot 2.2.4, the weight is not shown, as intended

(Inside EdgeEditDialogue)

private void CloseDialogue(object sender, EventArgs e)

{

if (tbxName.Text == "" && tbxWeight.Text == "")

{

((mainForm)Owner).activeEdge = null;

Close();

}

else if(tbxName.Text == "")

{

// Type check

try

{

double weight = Convert.ToDouble(tbxWeight.Text);

((mainForm)Owner).EditEdge(2, "", weight);

Close();

}

catch

{

MessageBox.Show("Weight must be a number");

}

}

else if(tbxWeight.Text == "")

{

((mainForm)Owner).EditEdge(1, tbxName.Text, -1);

Close();

}

else

{

// Type check

try

{

double weight = Convert.ToDouble(tbxWeight.Text);

((mainForm)Owner).EditEdge(0, tbxName.Text, weight);

Close();

}

catch

{

MessageBox.Show("Weight must be a number");

}

}

}

/// <summary>

/// Changes various properties of an the active edge

/// </summary>

/// <param name="mode">0 = Set name and weight, 1 = Set name, 2 = Set weight, 3/other = None</param>

/// <param name="newName">The name that should be set or not</param>

/// <param name="newWeight">The weight that should be set or not</param>

public void EditEdge(int mode, string newName, double newWeight)

{

switch (mode)

{

case 0:

// Change both

activeEdge.SetName(newName);

activeEdge.SetWeight(newWeight);

break;

case 1:

// Change name

activeEdge.SetName(newName);

break;

case 2:

// Change weight

activeEdge.SetWeight(newWeight);

break;

case 3:

// Do nothing

break;

}

DisplayGraph(activeGraph);

activeEdge = null;

}

Change as of test 2.3.5:

// Change weight

if (newWeight < 0)

{

MessageBox.Show("Weight must be positive, value not changed");

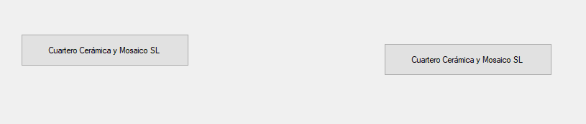
}

else

{

activeEdge.SetWeight(newWeight);

}



Screenshot 2.3.6, both nodes were assigned the same name!

### Code added as a result of error:

public void EditNode(string newName)

{

// To ensure no identical names are created, we must destroy the existing node and replace it with one with the new name

// This way, the validation inside the node constructor can be carried out

// First, store all attributes of the node (except name)

List<Arc> oldConnections = activeNode.connections;

Point oldLocation = activeNode.location;

activeGraph.nodes.Remove(activeNode);

activeGraph.nodes.Add(new Node(activeGraph, newName, oldLocation));

activeGraph.nodes.Last().connections = oldConnections;

DisplayGraph(activeGraph);

activeNode = null;

}



Screenshot 2.4.0, working solution shown on **Graph C**



Screenshot 2.4.1, working solution shown on **Graph C**

Code used:

private void ZoomLvlChanged(object sender, EventArgs e)

{

zoomLevel = (225-trbZoom.Value) / 100f;

foreach (Control c in this.Controls)

{

if ((string)c.Tag == "Graph Part" || (string)c.Tag == "Edge Label")

{

SizeF ratio = new SizeF(zoomLevel / prevZoomLevel, zoomLevel / prevZoomLevel);

Point oldLocation = c.Location;

c.Scale(ratio);

// Because for some reason this scales the controls about (0,0)

c.Location = oldLocation;

if (zoomLevel <= 0.5 && prevZoomLevel > 0.5)

{

if ((string)c.Tag == "Graph Part")

{

c.Text = ShortenText(c.Text);

}

else

{

activeEdge = FindArcFromID(Convert.ToInt32(c.Name));

if (activeEdge.GetWeight() == 0)

{

if (activeEdge.GetName() == "")

{

c.Text = "\_";

}

else

{

c.Text = activeEdge.GetName();

}

}

else if (activeEdge.GetName() == "")

{

c.Text = "W: " + activeEdge.GetWeight();

}

else

{

c.Text = ShortenText(activeEdge.GetName()) + ", W: " + activeEdge.GetWeight();

}

}

}

else if (zoomLevel > 0.5 && prevZoomLevel <= 0.5)

{

if ((string)c.Tag == "Graph Part")

{

c.Text = c.Name;

}

else

{

activeEdge = FindArcFromID(Convert.ToInt32(c.Name));

if (activeEdge.GetWeight() == 0)

{

if (activeEdge.GetName() == "")

{

c.Text = "\_";

}

else

{

c.Text = activeEdge.GetName();

}

}

else if (activeEdge.GetName() == "")

{

c.Text = "Weight = " + activeEdge.GetWeight();

}

else

{

c.Text = activeEdge.GetName() + ", weight = " + activeEdge.GetWeight();

}

}

}

}

}

prevZoomLevel = zoomLevel;

private string ShortenText(string input)

{

string[] words = input.Split(' ','-');

string output = "";

foreach (string word in words)

{

if (word.Length > 0)

{

output += word[0].ToString().ToUpper();

}

}

return output;

}

DisplayGraph() and OnPaint() were also updated:

// Create button to represent node

Button btnNode = new Button

{

Location = n.location,

Name = n.name,

Tag = "Graph Part",

Size = new Size((int)(234f \* zoomLevel), (int)(45f \* zoomLevel)),

};

if (zoomLevel <= 0.5)

{

btnNode.Text = ShortenText(btnNode.Name);

}

else

{

btnNode.Text = btnNode.Name;

}

btnNode.Click += new EventHandler(HandleNodeClick);

btnNode.LocationChanged += new EventHandler(HandleNodeDrag);

Controls.Add(btnNode);

// Draw arcs to each connected node

foreach (Arc a in n.connections)

{

// Prevents two buttons per arc by checking if one already exists

// (ContainsKey checks for a control with the given name)

if (!Controls.ContainsKey(a.ID.ToString()))

{

Button btnArc = new Button

{

// Centerpoint between nodes

Location = new Point((a.between[0].location.X + a.between[1].location.X) / 2,

(a.between[0].location.Y + a.between[1].location.Y) / 2),

Tag = "Edge Label",

Size = new Size((int)(75f \* zoomLevel), (int)(75f \* zoomLevel)),

Name = a.ID.ToString()

};

// When this button is clicked, call the HandleEdgeClick procedure

btnArc.Click += new EventHandler(HandleEdgeClick);

if (zoomLevel > 0.5)

{

if (a.GetWeight() == 0)

{

if (a.GetName() == "")

{

btnArc.Text = "\_";

}

else

{

btnArc.Text = a.GetName();

}

}

else if (a.GetName() == "")

{

btnArc.Text = "Weight = " + a.GetWeight();

}

else

{

btnArc.Text = a.GetName() + ", weight = " + a.GetWeight();

}

}

else

{

if (a.GetWeight() == 0)

{

if (a.GetName() == "")

{

btnArc.Text = "\_";

}

else

{

btnArc.Text = ShortenText(a.GetName());

}

}

else if (a.GetName() == "")

{

btnArc.Text = "W: " + a.GetWeight();

}

else

{

btnArc.Text = ShortenText(a.GetName()) + ", W: " + a.GetWeight();

}

}

Controls.Add(btnArc);

}

}

}

}

// Called on "this.Invalidate()", clears the canvas each time

protected override void OnPaint(PaintEventArgs e)

{

// Source: https://stackoverflow.com/questions/47239557/c-sharp-how-do-i-draw-a-line-between-two-objects-on-a-windows-form

base.OnPaint(e);

using (var blackPen = new Pen(Color.Black, 3 \* zoomLevel \* zoomLevel))

{

foreach (Node n in activeGraph.nodes)

{

foreach (Arc a in n.connections)

{

e.Graphics.DrawLine(blackPen, a.between[0].location, a.between[1].location);

}

}

}

}

### Iteration 2 stakeholder feedback and response

Here is a summary of the feedback given:

* Graph creation is very intuitive, however it does not state anywhere that “g” can be used to move parts of the graph
* The dimensions of nodes and edges are not ideal
* Dark theme and large text mode are not correctly implemented for the graph controls

Response:

* Added icon to explain movement



* Gave graph buttons AutoSize, MinimumSize and MaximumSize properties (Changes below)
* Added code to GraphDisplay() to take options into account (Changes below)





### Iteration 3 testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Type | Expected result | Linked success criteria | Result | Action taken |
| 3.0.0 | Testing saving | The save button is pressed while each graph is displayed (separately) | Valid | A file dialogue is opened and the user can save a file with a name and location | 4 | Success (Screenshot 3.0.0) | None |
| 3.0.1a | Testing loading on a blank graph | The load button is pressed while no graph is displayed | Valid | A file dialogue allows the user to open a file that is of the required file type | 4 | Success (Screenshot 3.0.1a) | None |
| 3.0.1b | Testing loading on an existing graph | Load button is pressed while **graph C** is displayed | Valid | Before the file dialogue is opened, a message asks the user whether they would like to cancel the operation | 4 | Success (Screenshot 3.0.1b) | None |
| 3.0.2 | A graph with the invalid file type is loaded | The user selects this document as the graph to load | Invalid | The operation is cancelled and a message in the status bar explains why | 4,7 | Success (Screenshot 3.0.2) | None |
| 3.1.0 | Unexpected close with **graph B** loaded | **Graph B** is loaded from a file, altered, and then after 5 minutes (autosave time) the program is closed using Task Manager | Invalid | When the program is next opened, a message tells the user what happened and where to find the autosave, which will be named \_NAME\_+autosave | 7,9 | Success (Screenshot 3.1.0) | None |

### Iteration 3 evidence:

 Screenshot 3.0.0

private void SaveClicked(object sender, EventArgs e)

{

SaveFileDialog fileDialogue = new SaveFileDialog();

fileDialogue.Filter = "WR Graph File|\*.wrgf";

fileDialogue.Title = "Save a graph";

fileDialogue.AddExtension = true;

DialogResult dialogueResult = fileDialogue.ShowDialog();

if (dialogueResult == DialogResult.OK)

{

string path = fileDialogue.FileName;

activeGraph.Save(path, false);

}

}

/// <summary>

/// Saves a wrgf file containing the graph at the location specified

/// </summary>

/// <param name="path">Location to save the file at</param>

/// <param name="autosave">If this is set to true, the graph's saved property is not updated</param>

public void Save(string path, bool autosave=false)

{

// This will be used to detect whether the file is of the correct format

string data = "GRAPH FILE\n";

data += minWeight + "\n" + nodeID + "\n";

foreach (Node n in nodes)

{

data += n.location.ToString() + n.name + ":";

foreach (Arc a in n.connections)

{

data += a.ID + "," + a.GetName() + "," + a.GetDestination(n).name + "," + a.GetWeight();

data += "\n";

}

}

data += "END";

try

{

File.WriteAllText(path, data);

if (!autosave)

{

wasSaved = true;

}

}

catch (Exception e)

{

System.Windows.Forms.MessageBox.Show("Graph not saved, reason: " + e.Message);

}

}



Screenshot 3.0.1a

private void LoadClicked(object sender, EventArgs e)

{

if (activeGraph.nodes.Count > 0 && !activeGraph.wasSaved)

{

DialogResult userAnswer = MessageBox.Show("Are you sure you want to overwrite the current graph?","Overwrite?",MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (userAnswer == DialogResult.No)

{

return;

}

}

OpenFileDialog fileDialogue = new OpenFileDialog();

fileDialogue.Filter = "WR Graph File|\*.wrgf";

fileDialogue.Title = "Load a graph";

DialogResult dialogueResult = fileDialogue.ShowDialog();

if (dialogueResult == DialogResult.OK)

{

string path = fileDialogue.FileName;

LoadGraph(path);

}

}

/// <summary>

/// Parses and validates wrgf files then displays them

/// </summary>

/// <param name="path">Location of the file</param>

private void LoadGraph(string path)

{

Graph graph = new Graph();

string[] fileLines = File.ReadAllLines(path);

string wholeFile = File.ReadAllText(path);

if (fileLines[0] != "GRAPH FILE")

{

statusLabel.Text = "The file specified is not in the required format or is corrupted";

return;

}

else

{

graph.minWeight = Convert.ToDouble(fileLines[1]);

graph.nodeID = Convert.ToInt32(fileLines[2]);

// Regex to capture nodes

string regex = @"{X=([0-9]+),Y=([0-9]+)}([^:]+)";

foreach (Match m in Regex.Matches(wholeFile, regex, RegexOptions.Multiline))

{

Point location = new Point(Convert.ToInt32(m.Groups[1].Value), Convert.ToInt32(m.Groups[2].Value));

graph.nodes.Add(new Node(graph, m.Groups[3].Value, location));

}

// Now to find the arcs

List<int> createdArcs = new List<int>();

string[] colonSplit = wholeFile.Split(':');

for (int i = 1; i < colonSplit.Length; i++)

{

string[] arcs = colonSplit[i].Split('\n');

// The last item in this list will be a node or end of file, so we don't parse it here

for (int j = 0; j < arcs.Length - 1; j++)

{

string[] data = arcs[j].Split(',');

int tempId = Convert.ToInt32(data[0]);

if (!createdArcs.Contains(tempId))

{

createdArcs.Add(tempId);

Node destination = FindNodeWithName(graph.nodes, data[2]);

graph.nodes[i - 1].JoinTo(destination, data[1], Convert.ToDouble(data[3]), ref tempId);

IDCount = Math.Max(tempId, IDCount);

}

}

}

activeGraph = graph;

graph.wasSaved = true;

DisplayGraph(activeGraph);

}

}



Screenshot 3.0.1b

Same code as above



Screenshot 3.0.2

Same code as above



Screenshot 3.1.0

private void TimeTick(object sender, EventArgs e)

{

// Called every 5 minutes

activeGraph.Save(Application.StartupPath + "\\autosave.wrgf", true);

}

private void ProgramClosing(object sender, FormClosingEventArgs e)

{

File.WriteAllText(Application.StartupPath + "\\AutosaveData.txt", activeGraph.wasSaved.ToString());

}

private void ProgramLoaded(object sender, EventArgs e)

{

// While this code seems to do nothing, it actually has the effect of applying the saved options to the main form when the program starts

OptionsForm optionsForm = new OptionsForm();

optionsForm.Show();

optionsForm.Close();

// Check what happened last time

try

{

string lastData = File.ReadAllText(Application.StartupPath + "\\AutosaveData.txt");

if (lastData == "False")

{

MessageBox.Show("The last graph loaded was not saved when the program closed. To recover lost work, the autosave can be found at: " + Application.StartupPath + "\\autosave.wrgf");

}

}

catch { }

autosaveTimer.Start();

}

### Iteration 3 stakeholder feedback and response

Feedback was very positive, however one stakeholder requested a feature where, if the graph is unsaved when the program is closed, a message comes up asking the user to save it.

I implemented this as a solution:

(Inside ProgramClosing)

if (activeGraph.nodes.Count > 0 && !activeGraph.wasSaved)

{

DialogResult userAnswer = MessageBox.Show("Current graph is not saved, would you like to save?", "Save graph?", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (userAnswer == DialogResult.Yes)

{

File.WriteAllText(Application.StartupPath + "\\AutosaveData.txt", "True");

SaveClicked(new object(), new EventArgs());

}

else if (userAnswer == DialogResult.No)

{

File.WriteAllText(Application.StartupPath + "\\AutosaveData.txt", "True");

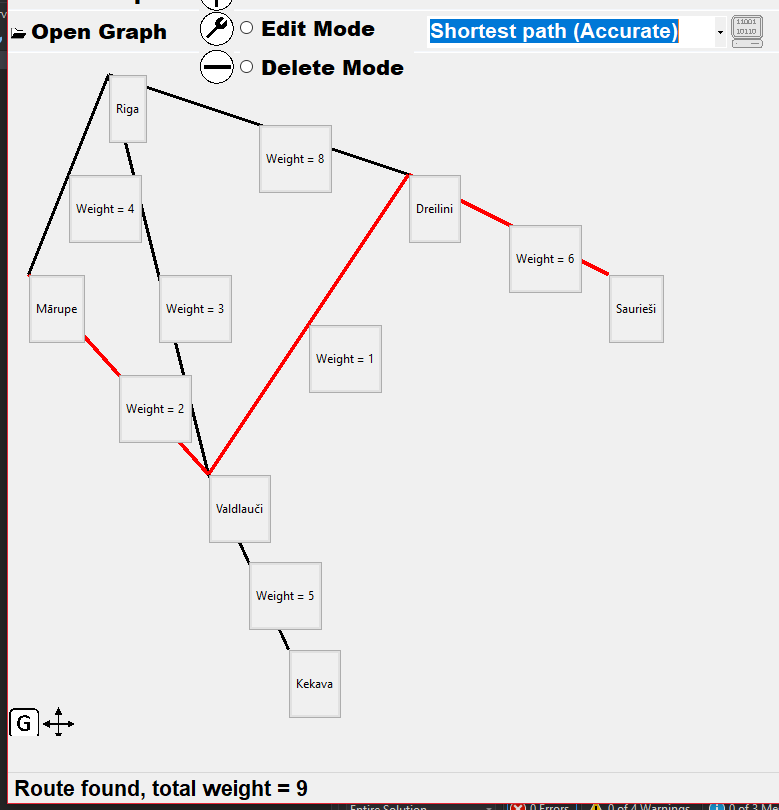
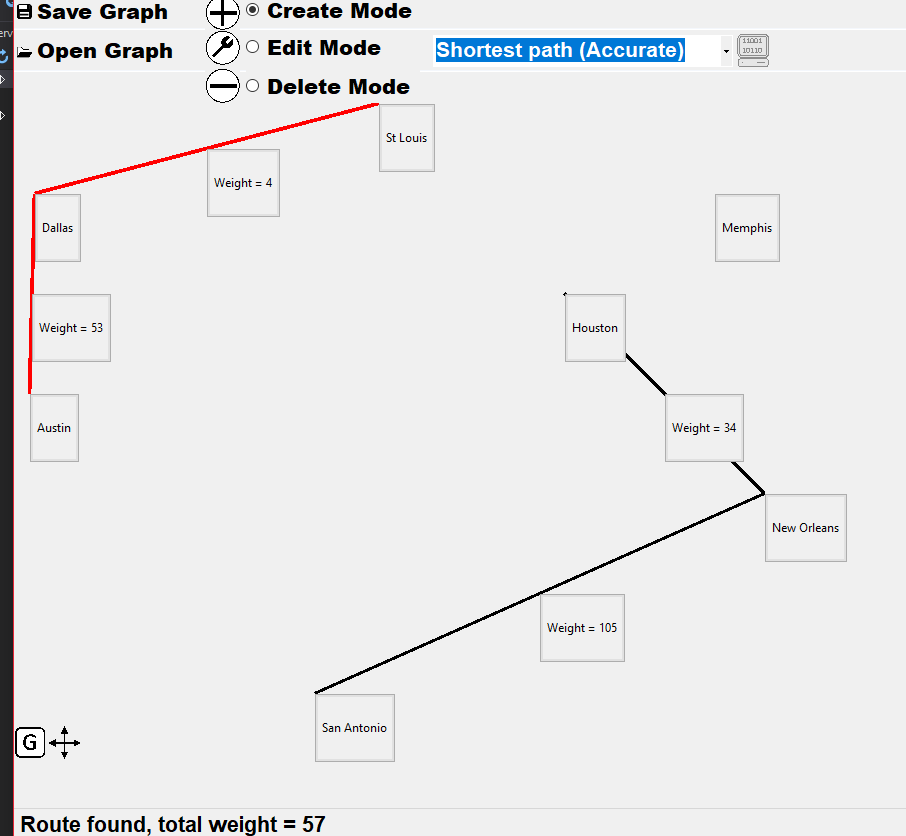
}

}

### Iteration 4 testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Type | Expected result | Linked success criteria | Result | Action taken |
| 4.0.0 | Dijkstra’s on various graphs | The first node shall be the leftmost node and the destination the rightmost valid node for each graph (A, B, and C) tested | Valid | For graph A: Barton-upon-Humber, Álta, B, length: 3  For B: Mārupe, Valdlauči, Dreilini, Saurieši, length: 9  For C: Austin, Dallas, St Louis, length 57 | 2.1 | Success (Screenshots 4.0.0a, 4.0.0b, and 4.0.0c) | None |
| 4.0.1 | A\* on various graphs | The first node shall be the leftmost node and the destination the rightmost valid node for each graph (A, B, and C) tested | Valid | For graph A: Barton-upon-Humber, Álta, B, length: 3  For B: Mārupe, Valdlauči, Dreilini, Saurieši, length: 9  For C: Austin, Dallas, St Louis, length 59 | 2.2 | Success (Screenshots 4.0.1a, 4.0.1b, and 4.0.1c) | None |
| 4.1.0 | Kruskal’s on various graphs | Graphs A, B, and C shall be tested separately | Valid | The minimum spanning tree will be highlighted in red | 2.3 | Success (Screenshots 4.1.0a, 4.1.0b, and 4.1.0c) | None |
| 4.1.1 | Floyd’s on various graphs | Graphs A, B, and C shall be tested separately | Valid | A table of least distances will be shown for each graph | 2.5 | Success (Screenshots 4.1.1a, 4.1.1b, and 4.1.1c) | None |
| 4.2.0 | Travelling salesman on a valid graph | Graphs A shall be tested separately starting from Barton-Upon-Humber | Valid | **Route for A:** Barton-upon-Humber, Cammeringham Hill, B, Alta, Barton-upon-Humber, length: 11 | 2.4 | Failure (Crash, null reference exception) | Passed list by value instead of by reference |
| 4.3.0a | Dijkstra’s where no route is possible | Dijkstra’s is run on **graph C** starting at Dallas with a destination of Houston | Invalid | An error message is displayed stating that there are no routes between the two nodes | 2.1, 7, 13 | Success (Screenshot 4.3.0a) | None |
| 4.3.0b | Dijkstra’s starting and ending at the same node | Dijkstra’s is run on **graph B** starting at Riga with a destination of Riga | Valid Boundary | The shortest path is given as “Riga, length 0” | 2.1 | Success (Screenshot 4.3.0b) | None |
| 4.3.0c | Dijkstra’s where the start node is invalid | The start node for **graph A** is given as “Australia” | Invalid | An error message is displayed stating that “Australia” does not exist | 2.1,7,13 | Success (Screenshot 4.3.0c) | None |
| 4.3.1a | A\* where no route is possible | A\* is run on **graph C** starting at Dallas with a destination of Houston | Invalid | An error message is displayed stating that there are no routes between the two nodes | 2.2, 7, 13 | Success (Screenshot 4.3.1a) | None |
| 4.3.1b | A\* starting and ending at the same node | A\* is run on **graph B** starting at Riga with a destination of Riga | Valid Boundary | The shortest path is given as “Riga, length 0” | 2.2 | Success (Screenshot 4.3.1b) | None |
| 4.3.1c | A\* where the start node is invalid | The start node for **graph B** is given as “Álta” | Invalid | An error message is displayed stating that “Álta” does not exist | 2.2,7,13 | Success (Screenshot 4.3.1c) | None |
| 4.3.2 | Travelling salesman on a graph with no Hamiltonian cycles | Travelling salesman is run on **Graph B** and then **C** | Invalid | An error message is displayed explaining that no routes exist for the problem | 2.4, 7, 13 | Failure (Screenshot 4.3.2) | Instead of checking for a null return value, check for a list of length 0 |

### Iteration 4 evidence:

Screenshots 4.0.0a, 4.0.0b, and 4.0.0c respectively

(Inside MainForm)

private void AlgorithmChosen(object sender, EventArgs e)

{

if (cbxAlgorithmSelect.Text == "Shortest path (Accurate)" || cbxAlgorithmSelect.Text == "Shortest path (Fast)")

{

AlgorithmInputDialogue algorithmInput = new AlgorithmInputDialogue();

algorithmInput.selectedAlgorithm = cbxAlgorithmSelect.Text;

algorithmInput.ShowDialog(this);

}

else

{

}

}

OnPaint was updated to highlight arcs

foreach (Arc a in n.connections)

{

if (a.highlighted)

{

blackPen.Color = Color.Red;

blackPen.Width \*= 1.25f;

}

e.Graphics.DrawLine(blackPen, a.between[0].location, a.between[1].location);

blackPen.Color = Color.Black;

blackPen.Width = 3 \* zoomLevel \* zoomLevel;

}

/// <summary>

/// Called from AlgorithmInputDialogue, parses, validates and carries out algorithms then displays the output

/// </summary>

/// <param name="selectedAlgorithm">The algorithm's name in the combo box</param>

/// <param name="start">The name of the start node (Where applicable)</param>

/// <param name="end">The name of the destination node (Where applicable)</param>

public void RunAlgorithm(string selectedAlgorithm, string start, string end)

{

ClearHighlight(activeGraph);

Node from = null;

Node to = null;

switch (selectedAlgorithm)

{

case "Shortest path (Accurate)":

from = FindNodeWithName(activeGraph.nodes, start);

to = FindNodeWithName(activeGraph.nodes, end);

if (from == null)

{

MessageBox.Show("Error, \"" + start + "\" does not exist on this graph");

}

else if (to == null)

{

MessageBox.Show("Error, \"" + end + "\" does not exist on this graph");

}

else

{

List<Node> route = activeGraph.Dijkstra(from, to);

if (route != null)

{

double sum = 0;

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

{

Arc arc = route[i].GetArcBetween(route[i + 1]);

arc.highlighted = true;

sum += arc.GetWeight();

}

DisplayGraph(activeGraph);

statusLabel.Text = "Route found, total weight = " + sum;

}

else

{

statusLabel.Text = "No possible route found";

}

}

break;

case "Shortest path (Fast)":

from = FindNodeWithName(activeGraph.nodes, start);

to = FindNodeWithName(activeGraph.nodes, end);

if (from == null)

{

MessageBox.Show("Error, \"" + start + "\" does not exist on this graph");

}

else if (to == null)

{

MessageBox.Show("Error, \"" + end + "\" does not exist on this graph");

}

else

{

List<Node> route = activeGraph.AStar(from, to);

if (route != null)

{

double sum = 0;

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

{

Arc arc = route[i].GetArcBetween(route[i + 1]);

arc.highlighted = true;

sum += arc.GetWeight();

}

DisplayGraph(activeGraph);

statusLabel.Text = "Route found, total weight = " + sum;

}

else

{

statusLabel.Text = "No possible route found";

}

}

break;

}

}

public void ClearHighlight(Graph graph)

{

foreach (Node n in graph.nodes)

{

foreach (Arc a in n.connections)

{

a.highlighted = false;

}

}

}

(Inside AlgorithmInputDialogue)

private void CloseDialogue(object sender, EventArgs e)

{

((mainForm)Owner).RunAlgorithm(selectedAlgorithm, tbxStart.Text, tbxEnd.Text);

this.Close();

}

private void FormOpened(object sender, EventArgs e)

{

if (selectedAlgorithm == "Find shortest tour of all nodes")

{

tbxEnd.Enabled = false;

}

}

(Inside Graph)

public List<Node> Dijkstra(Node start, Node end)

{

int count = nodes.Count;

// Distance is the distance from the start

double[] distance = new double[count];

// Store the node before the one at the index position on the shortest route

Node[] previous = new Node[count];

// Set to true when all routes to neighbours have been checked

bool[] explored = new bool[count];

int i = 0;

foreach (Node n in nodes)

{

if (n == start)

{

distance[i] = 0;

}

else

{

distance[i] = double.PositiveInfinity;

}

// Previous is already null for all values so no need to set it here

explored[i] = false;

i++;

}

// Do this while there are still unexplored nodes

while (explored.Contains(false))

{

int smallestIndex = -1;

double smallestValue = double.PositiveInfinity;

for (int j = 0; j < count; j++)

{

if (explored[j] == false && distance[j] < smallestValue)

{

smallestIndex = j;

smallestValue = distance[j];

}

}

// This means we have some nodes which are unreachable from the start point, and we have explored all reachable nodes

if (smallestIndex == -1)

{

// First, check if the end is reachable

if (distance[nodes.IndexOf(end)] == double.PositiveInfinity)

{

// No possible route

return null;

}

else

{

for (int j = 0; j < count; j++)

{

return Backtrack(previous, end);

}

}

}

explored[smallestIndex] = true;

foreach (Arc a in nodes[smallestIndex].connections)

{

Node neighbour = a.GetDestination(nodes[smallestIndex]);

int indexOfNeighbour = nodes.IndexOf(neighbour);

if (explored[indexOfNeighbour] == false)

{

double newDist = distance[smallestIndex] + a.GetWeight();

if (newDist < distance[indexOfNeighbour])

{

distance[indexOfNeighbour] = newDist;

previous[indexOfNeighbour] = nodes[smallestIndex];

}

}

}

}

return Backtrack(previous, end);

}

private List<Node> Backtrack(Node[] previous, Node end)

{

List<Node> route = new List<Node>();

int pos = nodes.IndexOf(end);

route.Add(end);

while (previous[pos] != null)

{

route.Add(previous[pos]);

pos = nodes.IndexOf(previous[pos]);

}

// Reverse returns nothing, but alters the list.

route.Reverse();

return route;

}

Screenshots 4.0.1a, 4.0.1b, and 4.0.1c respectively

The only additional code is inside Graph

public List<Node> AStar(Node start, Node end)

{

// For any node in the open set, cameFrom at the same index is the node before it on the shortest path

var cameFrom = new List<Node>();

// gScore is the cost of the cheapest path from start to the node

var gScore = new List<double>();

// fScore is our current best guess at how cheap a path could be

var fScore = new List<double>();

List<bool> explored = new List<bool>();

foreach (Node n in nodes)

{

cameFrom.Add(null);

gScore.Add(double.PositiveInfinity);

fScore.Add(double.PositiveInfinity);

explored.Add(false);

}

int startPos = nodes.IndexOf(start);

gScore[startPos] = 0f;

fScore[startPos] = Heuristic(start, end);

while (explored.Contains(false))

{

// The node with the lowest fScore

int currentPos = -1;

double min = double.PositiveInfinity;

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

{

if (fScore[i] < min && explored[i] == false)

{

currentPos = i;

min = fScore[i];

}

}

// This means we have some nodes which are unreachable from the start point, and we have explored all reachable nodes

if (currentPos == -1)

{

// First, check if the end is reachable

if (gScore[nodes.IndexOf(end)] == double.PositiveInfinity)

{

// No possible route

return null;

}

else

{

for (int j = 0; j < fScore.Count; j++)

{

return Backtrack(cameFrom.ToArray(), end);

}

}

}

Node currentNode = nodes[currentPos];

if (currentNode == end)

{

List<Node> path = new List<Node>();

while (currentNode != null)

{

path.Add(currentNode);

currentNode = cameFrom[nodes.IndexOf(currentNode)];

}

path.Reverse();

return path;

}

else

{

explored[currentPos] = true;

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

{

int neighbourPos = nodes.IndexOf(currentNode.connections[i].GetDestination(currentNode));

double temp\_gScore = gScore[currentPos] + currentNode.connections[i].GetWeight();

if (temp\_gScore < gScore[neighbourPos])

{

// This path is better than any previous one

cameFrom[neighbourPos] = currentNode;

gScore[neighbourPos] = temp\_gScore;

fScore[neighbourPos] = temp\_gScore + Heuristic(nodes[neighbourPos], end);

if (explored[neighbourPos] == true)

{

explored[neighbourPos] = false;

}

}

}

}

}

// Failure

return null;

}

For prototype 4.1, although I had initially planned to use Kruskal’s algorithm, I had not realised the complexity involved in checking for cycles when adding arcs, so I have instead opted to use Prim’s algorithm, which will output the same results in a similar time frame for most graphs, but is vastly simpler to implement.

public double Heuristic(Node current, Node goal)

{

// Good heuristic goes here.

// Using min edge weight for now

return minWeight;

}



Screenshots, 4.1.0a, 4.1.0b, and 4.1.0c. The algorithm fails for graph C, as it should since MSTs are not possible on unconnected graphs, which I had overlooked in my test tables.

public List<Arc> Prims()

{

List<Arc> MST = new List<Arc>();

List<Node> joinedNodes = new List<Node>();

// Arbitrary start node

Node start = nodes[0];

joinedNodes.Add(start);

while (joinedNodes.Count < nodes.Count)

{

double min = double.PositiveInfinity;

Arc minArc = null;

Node minNode = null;

foreach (Node n in joinedNodes)

{

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

{

// If the connection is the smallest we've seen so far, and it does not connect to a node already on the tree, select it

if (n.connections[i].GetWeight() < min && !joinedNodes.Contains(n.connections[i].GetDestination(n)))

{

min = n.connections[i].GetWeight();

minArc = n.connections[i];

minNode = n;

}

}

}

if (minArc == null)

{

// Graph has unconnected nodes, so a MST is not possible

return null;

}

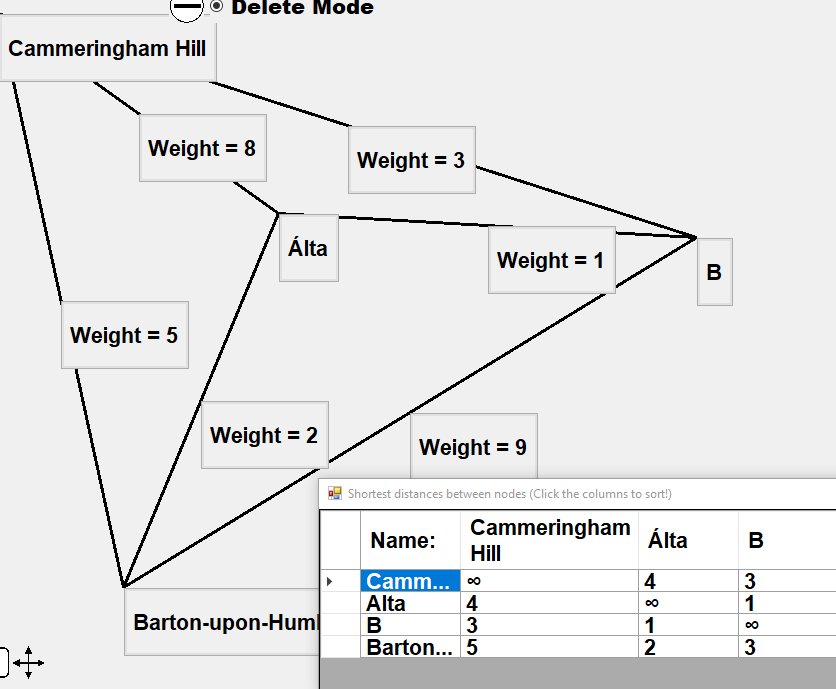
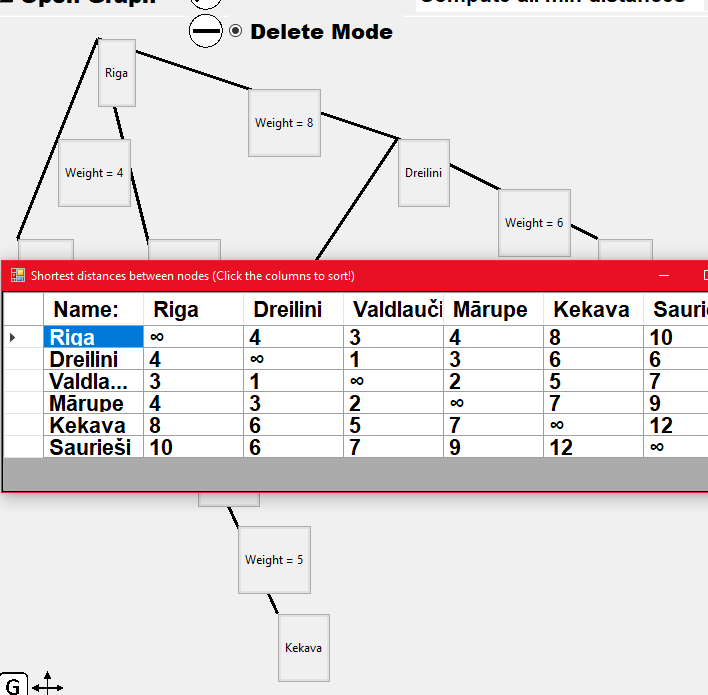
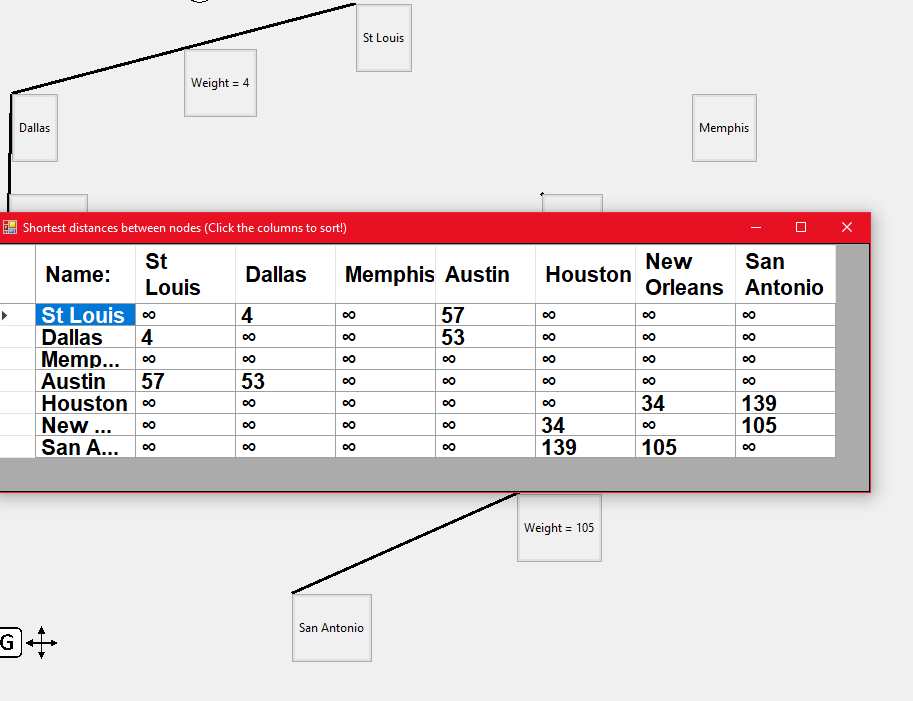
MST.Add(minArc);

joinedNodes.Add(minArc.GetDestination(minNode));

}

return MST;

}

Screenshots 4.1.1a, 4.1.1b, and 4.1.1c respectively

public double[,] Floyds()

{

double[,] leastDistances = new double[nodes.Count, nodes.Count];

// Initialise values to infinity

for (int i = 0; i < nodes.Count \* nodes.Count; i++)

{

leastDistances[i/nodes.Count, i % nodes.Count] = double.PositiveInfinity;

}

// Set up immediate distances (directly connected nodes)

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

{

foreach (Arc a in nodes[i].connections)

{

int gridX = i;

int gridY = nodes.IndexOf(a.GetDestination(nodes[i]));

if (a.GetWeight() < leastDistances[gridX, gridY])

{

// Not using Math.Min here for future development so that a routing table can be added in future more easily

leastDistances[gridX, gridY] = a.GetWeight();

}

}

}

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

{

for (int x = 0; x < nodes.Count; x++)

{

for (int y = 0; y < nodes.Count; y++)

{

// Check connections to each node, excluding node i and the diagonal (node connecting to itself)

if (x != y && x != i && y != i)

{

if (leastDistances[y, i] + leastDistances[i, x] < leastDistances[y, x])

{

leastDistances[y, x] = leastDistances[y, i] + leastDistances[i, x];

}

}

}

}

}

return leastDistances;

}

(Inside a new form, LeastDistanceDisplay)

public partial class LeastDistancesDisplay : Form

{

public double[,] distances;

public List<Node> nodes;

public LeastDistancesDisplay()

{

InitializeComponent();

}

private void DisplayOpened(object sender, EventArgs e)

{

DataTable dataTable = new DataTable("Min Distances");

dataTable.Columns.Add(new DataColumn("Name:"));

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

{

dataTable.Columns.Add(new DataColumn(nodes[i].name));

}

for (int j = 0; j < distances.GetLength(0); j++)

{

DataRow row = dataTable.NewRow();

row["Name:"] = nodes[j].name;

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

{

row[nodes[i].name] = distances[i, j];

}

dataTable.Rows.Add(row);

}

dataGridView.DataSource = dataTable;

dataGridView.Font = new Font("Arial", 16, FontStyle.Bold);

dataGridView.AutoSize = true;

dataGridView.Refresh();

this.TopMost = true;

}

}

Code for 4.2.0 (that caused an error)

(In Graph)

public List<Node> TravellingSalesman(Node start)

{

List<Node> minPath = new List<Node>();

List<Node> path = new List<Node>();

List<int> nodeIDs = new List<int>();

int count = 0;

foreach (Node n in nodes)

{

if (n != start)

{

path.Add(n);

nodeIDs.Add(count);

count++;

}

}

double shortest = double.PositiveInfinity;

do

{

int numTested = 0;

double currentPathWeight = -1;

//Find current path weight

try

{

for (int i = 0; i <= path.Count - 2; i++)

{

currentPathWeight += path[i].GetArcBetween(path[i + 1]).GetWeight();

}

currentPathWeight += path[0].GetArcBetween(start).GetWeight();

currentPathWeight += path[path.Count - 1].GetArcBetween(start).GetWeight();

// Update shortest length

if (currentPathWeight < shortest)

{

shortest = currentPathWeight;

minPath = path;

}

}

catch

{

}

numTested++;

} while (FindNextPermutation(ref nodeIDs, ref path));

if (shortest == double.PositiveInfinity)

{

// This means that there is no possible route

return new List<Node>();

}

else

{

return minPath;

}

}

// Find the next permutation of the int array. Adapted from https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/

public static bool FindNextPermutation(ref List<int> data, ref List<Node> nodes)

{

// If the given dataset is empty or contains only one element next\_permutation is not possible

if (data.Count <= 1)

return false;

int last = data.Count - 2;

// find the longest non-increasing suffix and find the pivot

while (last >= 0)

{

if (data[last] < data[last + 1])

break;

last--;

}

// If there is no increasing pair there is no higher order permutation

if (last < 0)

return false;

int nextGreater = data.Count - 1;

// Find the rightmost successor to the pivot

for (int i = data.Count - 1; i > last; i--)

{

if (data[i] > data[last])

{

nextGreater = i;

break;

}

}

// Swap the successor and the pivot

int temp = data[nextGreater];

Node tempN = nodes[nextGreater];

data[nextGreater] = data[last];

nodes[nextGreater] = nodes[last];

data[last] = temp;

nodes[last] = tempN;

// Reverse the suffix

int left = last + 1;

int right = data.Count - 1;

while (left < right)

{

temp = data[left];

tempN = nodes[left];

data[left] = data[right];

// ++ updates the variable AFTER it has been used

nodes[left++] = nodes[right];

data[right] = temp;

nodes[right--] = tempN;

}

// Return true as the next\_permutation is done

return true;

}

(In MainForm)

case "Find shortest tour of all nodes":

from = FindNodeWithName(activeGraph.nodes, start);

if (from != null)

{

List<Node> cycle = activeGraph.TravellingSalesman(from);

if (cycle != null)

{

// To create a closed loop

cycle.Add(from);

cycle.Add(cycle[0]);

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

{

Arc arc = cycle[i].GetArcBetween(cycle[i + 1]);

arc.highlighted = true;

sum += arc.GetWeight();

}

DisplayGraph(activeGraph);

statusLabel.Text = "Route found, total weight = " + sum;

}

}

break;

### Code added as a result of error:

The line that had caused the error was this line, inside the travelling salesman algorithm:

minPath = path;

This was a logic error, because I had expected that line to set the values inside minPath to a copy of the values inside path, whereas in reality minPath was set to the reference of path, meaning it was updated every time path was, so the final checked route (which turned out to be impossible) was returned instead of the best possible route

I solved this by changing the line as follows and adding a comment to explain.

// .ToList sets the list by value rather than by reference

minPath = path.ToList();

Screenshot 4.3.0a, code unchanged

 Screenshot 4.3.0b, code unchanged



Screenshot 4.3.0c, code unchanged

 Screenshot 4.3.1a, code unchanged

 Screenshot 4.3.1b, code unchanged



Screenshot 4.3.1c, code unchanged



Screenshot 4.3.2, this should display an error message, but instead shows an invalid solution

case "Find shortest tour of all nodes":

from = FindNodeWithName(activeGraph.nodes, start);

if (from != null)

{

List<Node> cycle = activeGraph.TravellingSalesman(from);

if (cycle != null)

{

// To create a closed loop

cycle.Add(from);

cycle.Add(cycle[0]);

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

{

Arc arc = cycle[i].GetArcBetween(cycle[i + 1]);

arc.highlighted = true;

sum += arc.GetWeight();

}

DisplayGraph(activeGraph);

statusLabel.Text = "Route found, total weight = " + sum;

}

else

{

statusLabel.Text = "No possible routes found";

}

}

else

{

statusLabel.Text = "Error, \"" + start + "\" does not exist on this graph";

}

break;

### Code added as a result of error:

Instead of if (cycle != null), I used if (cycle.Count > 0)

### Iteration 4 stakeholder feedback and response

// note to self: Warning for TSP and better tab controls

# Evaluation

## Testing for function

## Testing for robustness

## Testing for usability

## Success Criteria

## Further development

## Maintenance

1. The program will be tested on Windows 10 so although the program will most likely work from Windows XP and up I cannot be sure without testing for each operating system [↑](#footnote-ref-1)
2. This is the smallest amount of total RAM out of my stakeholders [↑](#footnote-ref-2)
3. Source: <https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715> [↑](#footnote-ref-3)
4. Source: <https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715> [↑](#footnote-ref-4)
5. My stakeholders use both 4:3 and 16:9 displays [↑](#footnote-ref-5)