CST3990 Undergraduate Individual Project
Mind Map Software

Benjamin Manuel M00666131
Supervisor: Dr Barry D. Nichols

I hereby confirm that the work presented here in this report and in all other associated material is wholly my own work

Print Name: BENJAMIN MANUEL Signature: B Manuel Date: 30/04/2021

Abstract

In this report I will be telling the story behind my individual project.

For my project, I decided I wanted to attempt to make software for making mind maps. I usually have trouble using mind map software and thought it would be a good incentive for the project if I tried to make it so I understood the software better than professional software.

In order to create said software, I had to read about mind maps in books and articles to learn what diagrams are defined as mind maps, how you make mind maps, what the software would need to be able to do in order to make the maps and why people make mind maps as additional explanation behind my idea. I also installed free mind map software that I found online and tested them out to see what they could and could not do, what did they have in common and what features that I wanted my prototype to have did these software also have.

I used this information to draw up a design for the user interface of my project as well as a list of capabilities and features it should have and a UML diagram of the Java classes I would need.

The design was used to create the code for the project.

The prototype was evaluated by using the design and specification as a checklist and checking if each element and feature that was in the design was in the final prototype. I also kept track of features or elements that were only partially implemented or unimplemented for specific reasons in order to evaluate the suitability of the language and frameworks used

I can conclude that while I have achieved a functional prototype and every core feature was implemented, the prototype has shortcomings and missing features. My approach to programming it and choice of technologies might not have been optimal due to inexperience, but I used the most suitable language and methodology from my point of view.

Contents

Introduction 5
Background and Literature Review 6
  Core Features of a Mind Map 6
  Core Purpose of Mind Mapping and Its Uses 7
  Common Features of Mind Map Software 8
  Popular Features of Mind Map Software 9
  Requirements Specification 16
Analysis and Design 17
Implementation 21
Testing and Evaluation 28
  Testing 29
  Test Details 29
  Evaluation 35
    Limitations and Incomplete Features 37
    Unimplemented Features 38
Conclusion 40
Appendices 45
  A - Project Supervisor Meeting Logs 45
  B Running Guide 50
  C Controls 51
    Interface Button Controls 51
    Mouse/Keyboard Controls 52
References 53

Introduction

In this project my goal is to create a software prototype for creating mind maps on computers. I planned for said prototype to have all the basic features common to software that I took as examples I research, such as the ability to draw boxes and lines on the screen, connect them up, edit them, change the colours of objects and save and load the maps created to file. I also planned for the software to contain as many advanced features or quality of life features as I could manage to figure out how to implement.

The sections of the report are as follows:

Firstly, the background and literature review which will cover my reasoning as to why I picked this problem to solve and discuss what mind maps are, what they are for, how they are made and what features are needed for a mind map to be considered a mind map. It will also contain the requirements specification based on the research done.

Next section contains my design for the software and the reasoning and justification for the design with reference to the literature review and specification as well as diagrams to support the design.

After this, the next section explains how I implemented the software using the design from the last section.

Next comes the results of my tests of the capabilities of the software and my evaluation of how successful the software was. Also, a list and explanation of all missing features and the reason why they were not implemented.

Then I conclude the report by summarising the achievements, the overall limitations of the prototype and how I would take the lessons learnt and limitations into account if I was to do the project again.

Finally, the appendices and references.

Background and Literature Review

I have often struggled to record my ideas and thoughts and have tended towards creating collections of written notes whenever asked to take notes for something despite being recommended to use a mind map or similar system. I was not able to use the mind-map properly because the way my thoughts and the software worked were not compatible. This in recent years has resulted in lots of thoughts which I could express and store better with appropriate software.

Core Features of a Mind Map

A traditional mind map starts with a central image, keyword or idea. It is considered more effective to use an image or drawing here, rather than just text. This is because we naturally find it easier to process colours and pictures than pure text (Hiranabe, 2007; Tsinakos and Balafoutis, 2009)*.* From the central image, labelled branches branch off in a recursive fashion, such that the further the branch is from the centre, the more refined and specific the concept it represents becomes. Some people choose to have the branch get physically thinner to represent this and some choose to have the labels (or picture) inside a little box or bubble at the end of the branch instead of on the branch.

Less traditional mind maps can have multiple central images or no explicitly defined centre in the event the concept the user is mapping out is unsuited to the traditional approach.

From *Agile Modeling with Mind Map and UML* (Hiranabe, 2007), the author lists what he believes to be important features of mind maps. These features can be summarised as the use of associated keywords at a high-level view in a semi-structured and easily expandable fashion. Such a map will be easy to use and will remind people of the context in which it was created.

Readers of *The Mind Map Book* (Buzan and Buzan, 2010) are strongly recommended to use variable size components in their mind maps. For example, if you make a mind map on animals and begin categorising them by size, you might label them ‘Big’ in large font, ‘Medium’ in regular font and ‘Small’ in small font or alternatively some picture that signals the same thing to you. Where relevant, this also extends to using 3D shapes and drawings to add emphasis.

The core problem with mind maps is that people maintain the opinion that mind maps are flexible and for everyone and the opinion that they must have a central image at the same time. (mostly through tradition and expectations). If the mind map is to be truly flexible and omni-purpose, it should be able to handle multiple central images. I believe there are improvements that can be made over existing software.

Core Purpose of Mind Mapping and Its Uses

Mind maps are used to convert information either from memory or from written form into a visual form. (Hiranabe, 2007, p.1) Mind maps should also ideally be made by the person who is going to be using the map. This is because they work through concept associations and each person will have different associations. The person with the greatest affinity for a map is the creator of the map. (Tsinakos and Balafoutis, 2009) Obviously, if the associations are quite broad and evidence-reinforced, the map will be simple to grasp. If the associations are more subjective and opinion/feeling based, they will be harder to interpret by people who aren't its creator. This raises some challenges which I will incorporate into the software requirements.

According to *Mind Map Generator Software* (Kudelić, Maleković, and Lovrenčić, 2012), uses for mind maps include:

- As a learning resource
- For structuring data
- As a search engine
- For planning
- For qualitative data analysis in a time constrained environment
- For idea generation
- For knowledge management

Another use can be seen in the article *English2Mindmap: an Automated System for MindMap Generation from English Text* (Elhoseiny and Elgammel, 2012), which explains the authors' theory, efforts and testing of their software designed to analyse passages of text and automatically generate mind maps based on the content. Their goal was to enable summarization of text via mind map. While there were several paragraphs in the article explaining the details about the semantic analysis and other methods used by the software, I only understand that the software they made takes the sentences, breaks them down into parse trees and then reassembles the trees into the mind map based on information about how the English language is structured.

According to *Agile Modeling with Mind Map and UML*, the most frequent uses for mind maps in business are to-do lists, presentation preparation and note taking (Hiranabe, 2007, p.3).

For uses of mind maps in special education, it is important to note that “the Mind Map frees the ‘learning disabled’ brain from semantic restrictions” (Buzan and Buzan, 2010, pg. 195), improving learning and expression.

Common Features of Mind Map Software

All mind map software shares common features that must therefore define the base standard.

The software I had originally chosen to study are iMindMap, Obsidian, Scapple and Milanote. There is also a table within the literature I am analysing comparing and contrasting Freemind, iMindMap, NovaMind, OpenMind and Xmind.

Based on my sources, the common features are therefore

- Intuitive interface
- Create nodes
- Edit and style existing nodes
- Image support
- Add hyperlinks or file links to nodes
- Drag and drop nodes on overview
- Export to external file
- Keyword search

- Windows OS support

Unfortunately, some of my sources are out of date and the software previously known as iMindMap has been absorbed into a product branded *AYOA* by a company also called *AYOA.* The company website contains a page which gives the impression that *AYOA* is the natural successor of iMindMap and includes all its best features. My main reason for continuing to study this software is that *The Mind Map Book* (Buzan and Buzan, 2010) directly references and encourages the use of iMindMap, quoted as “the official Buzan mind mapping software” (pg. 171).

A. A. Tsinakos also says that the software “is the only software product that fully duplicates T. Buzan‘s renowned mind mapping techniques” in *A Comparative Survey on Mind Mapping Tools* (Tsinakos and Balafoutis, 2009, pg. 59).

Popular Features of Mind Map Software

The Mind Map Book (Buzan and Buzan, 2010) says that mind mapping software allows for digital mind maps to be more flexible than their paper alternatives so at the very least, mind map software in general should be made to leverage those advantages as much as possible.

The software that I have studied also include several features which appear non-essential for mind map creation but leverage the advantages of computers. Some features augment the software's base features.

- Free-hand branches are supported by AYOA, Freemind, NovaMInd, OpenMind and XMind.
- AYOA reportedly supports touch screens according to *A Comparative Survey on Mind Mapping Tools* but due to the article being 11 years old and the above rebranding issue, I cannot confirm this claim.
- A right click menu customised to contain pictorial icons rather than the default text.

- “Brainstorming mode” (Tsinakos and Balafoutis, 2009, pg. 61) This appears to enable faster creation of nodes by automatically connecting the newest node to its most likely direct parent. It also makes the button for submitting a finished node also create a new node so the user never needs to to click on additional buttons.

Other features improve compatibility with other software, either by direct cross-platform support (AYOA, Obsidian, Milanote, Scapple, FreeMind, NovaMind and Xmind) or by being able to natively convert the mind map into a timeline,calender or Gantt chart (AYOA, Milanote, OpenMind) inside the software.

From *English2Mindmap: an Automated System for MindMap Generation from English Text* (Elhoseiny and Elgammel, 2012), I learnt that not all digital mind maps must be created by hand, nor will they necessarily have compatible styles or formats as each software uses whatever suits them best.

All the software that I have looked at is missing one or more potentially essential features as can be seen in the table on the next page. Any other features not on this table either are not deemed important or are available in all software listed. For example, all software listed can create nodes.

Feature	Ayoa	Obsidian	Milanote	Scapple
Movable nodes	Yes	No	Yes	Yes
Pictures	Only if you subscribe (£7.50 per user per month)	Not in graph view (can see pictures when viewing a single node)	Yes	No
Colour	Yes	Only if you want to highlight text yellow	Yes	Yes
Quick linking nodes	In quick mode	No	No	No
Freehand branches	Yes	No	Yes	No
Recursive nodes	No	No	Partially Not universal	No
Use of 3D space	No	No	No	No
Infinite workspace	Yes	Invisible boundary crops around nodes but stretches to fit	No	Yes, although it's a pain to traverse
Links	No	Yes	Yes	Yes

Windows support	Browser	Yes	Browser	Yes
Keyword Search	Yes	Yes	Subscription only ($9.99/month)	Uses an inbuilt tool called Find and Replace that is functionally similar
Export/Save	Only if you subscribe	Auto saves as Markdown	Export as PDF or PNG Save as Word, TXT or Markdown	It is possible to print the diagrams so a user could theoretically export by printing and scanning the printout as a new file

Table 1. Comparison table of common features

	Freemind	iMindMap	NovaMind	OpenMind	XMind
Operating System					
Windows	✓	✓	✓	✓	✓
Mac OS X	✓	✓	✓		✓
Linux	✓	✓			✓
Map Format Supported					
Traditional	✓	✓	✓	✓	✓
Directional		✓	✓	✓	✓
Organizational Charts				✓	✓
Fishbone					✓
Topic Features					
Map markers	✓	✓	✓	✓	✓
Customisable Symbols		✓	✓	✓	✓
Task related info		✓	✓	✓	✓
Priority related info		✓	✓	✓	✓
Call out topics			✓	✓	
Notes	✓	✓	✓	✓	✓
Hyperlinks	✓	✓	✓	✓	✓
Multiple links per branch			✓		✓
Embedded attachments		✓	✓	✓	✓
Topic numbering	✓	✓	✓	✓	✓

Topic boundaries	✓	✓	✓	✓	✓
Map Features					
Floating topics	✓	✓	✓	✓	✓
Background image	✓	✓	✓	✓	✓
Multi-page document			✓		✓
Customisable branch shape	✓	✓	✓	✓	✓
Freehand drawing tools		✓	✓		
Spell check		✓	✓	✓	
Wizards	✓	✓	✓	✓	✓
Relationship lines		✓	✓	✓	✓
Import images to maps	✓	✓	✓	✓	✓
Advanced Features					
Review map with others				✓	
Search topics		✓	✓	✓	✓
Integration with MS Office					
Word				✓	✓
Excel				✓	
PowerPoint				✓	✓
Outlook				✓	
MS Project				✓	
Internet Explorer				✓	

Export Formats					
PDF		✓	✓	✓	✓
Word				✓	✓
PowerPoint				✓	✓
Outlook				✓	
MS Project				✓	
HTML web page	✓	✓	✓	✓	✓
Image formats	✓	✓	✓	✓	✓
XML	✓	✓	✓	✓	✓

Table 2. Comparison table of five mind map software transcribed from source (Tsinakos and Balafoutis, 2009, p. 65)

Requirements Specification

The software will need to implement all the basic features that I have observed other software to include:

- Intuitive interface – Essentially, it needs to be designed so button, keyboard and mouse presses do the most likely action when pressed so as to minimise unnecessary actions and streamline the creation of maps.
- Create nodes – If the software cannot create nodes, it is impossible to use the software for its intended purpose.
- Edit and style existing nodes – If the software cannot edit nodes then no information can be stored. Also, the software would fail to allow the use of colour that is vital according to my research.
- Image support – Mind maps apparently need to be able to have pictures as we process the information better that way.
- Add hyperlinks or file links to nodes – This feature would not be possible on a paper map and tries to leverage the advantages of using a computer.
- Drag and drop nodes on overview – Just to allow the user to edit the map to make it easier to use and possible to change the links (associations) as you go along.
- Export and/or save to external file – In the event that the software needs a custom data structure to store maps that is not easily read by other programs, it seems wise to add the conversion feature to enable maps made by the software to be read by other software. Also, users may want to swap software or device and if a map is restricted to the copy of the software it was made on, the portability is low.
- Keyword search – Since the user is not restrained to a certain page size or even two dimensions, the map can get quite large and complicated. This allows the user to find certain nodes in the map.

There are other less vital features that the software needs to include.

The software must be very flexible to allow the user to make best fit associations often. That is difficult to quantify so I will be ensuring that features added to the software add functionality rather than restricting existing functionality.

It needs to support seamless switching between single core image, multiple core image and no core image for ease of use.

It must include recursive nodes allowing nodes to contain other nodes inside themselves. This serves to mentally partition the map into chunks as well as adding an extra dimension of space on the map that a paper map cannot support. It also lets the user add child nodes tangentially related to the parent node without compromising the main map flow.

The area on the user interface for creating maps should be practically infinite in size and of landscape orientation. Portrait orientation artificially stunts the map size. Nodes, lines and text should be scalable to help with associations. It is not possible with a paper mind map to have dynamically expanding space or scaling elements and adding such features helps to take advantage of computers rather than just cloning the capabilities of paper. (Buzan and Buzan, 2010)

Finally, it should include a brainstorming mode (Tsinakos and Balafoutis, 2009) to make the creation of simple maps faster and more intuitive. It would enable the creation of a multi-node chain without having to manually link them together.

Analysis and Design

I plan to make the mind map creation program as desktop software as this works nicely with Java.

For the next few paragraphs and diagrams, I will refer to a Board. This is being used because map is a function name in Java which is the language I am most likely to create this in.

This software will visually look like the design of other mind map software as the majority of the screen has to be used for map creation and so starts off blank to enable full freedom and flexibility for the user as seen in the UI mock-up below. For each of the functions listed above, I will explain how the design will implement it if relevant.

Creating nodes will be bound to a keyboard or mouse shortcut and to a UI button as a backup so users cannot get stuck at the beginning with no nodes on screen and no way to create one.

I am aiming for node styling options to appear in a popup next to the node it targets for ease of use, but it may end up being in an entirely separate popup window.

Images are already supported by a Node class attribute and links can go in the content attribute. Both will need to be detected and handled in the code.

Saving to an external file will be an integral part of the software and can be performed by saving the contents of the Board's content attribute to a file with the same name as the Board instance. Exporting can be achieved by writing an additional program to parse the content ArrayList and building a file in another format. I will bind this to a UI button.

Keyword searching can be achieved by searching the name, content and label attributes of Nodes and Connectors. I will bind this to a UI button.

The ability for maps to have either one, multiple or no core image is supported by the centre attribute of Nodes. If one node has that set to true, it is the centre node. If many nodes have it, the software should attempt to put them all as close together as possible. If all nodes have it set to false, no changes in display are made.

Recursive nodes are supported in the Node class. Nodes can contain an ArrayList of Nodes and Connectors. Buttons will be in the UI to support moving up and down levels.

The landscape orientation can be seen in the mock-up. I am not sure how to implement or mock-up the need for the infinite plane for space, but I do intend to add the feature.

The final feature on the list is 'Brainstorming mode'. This is a quality-of-life feature and will probably be one of the last features that I will implement as it relies on the implementation of several of the earlier features such as the ability to create and style nodes.

Then it will need a toolbar of actions that you can perform.

The toolbar shall be horizontal and collapsible to ensure it does not intrude on map space.

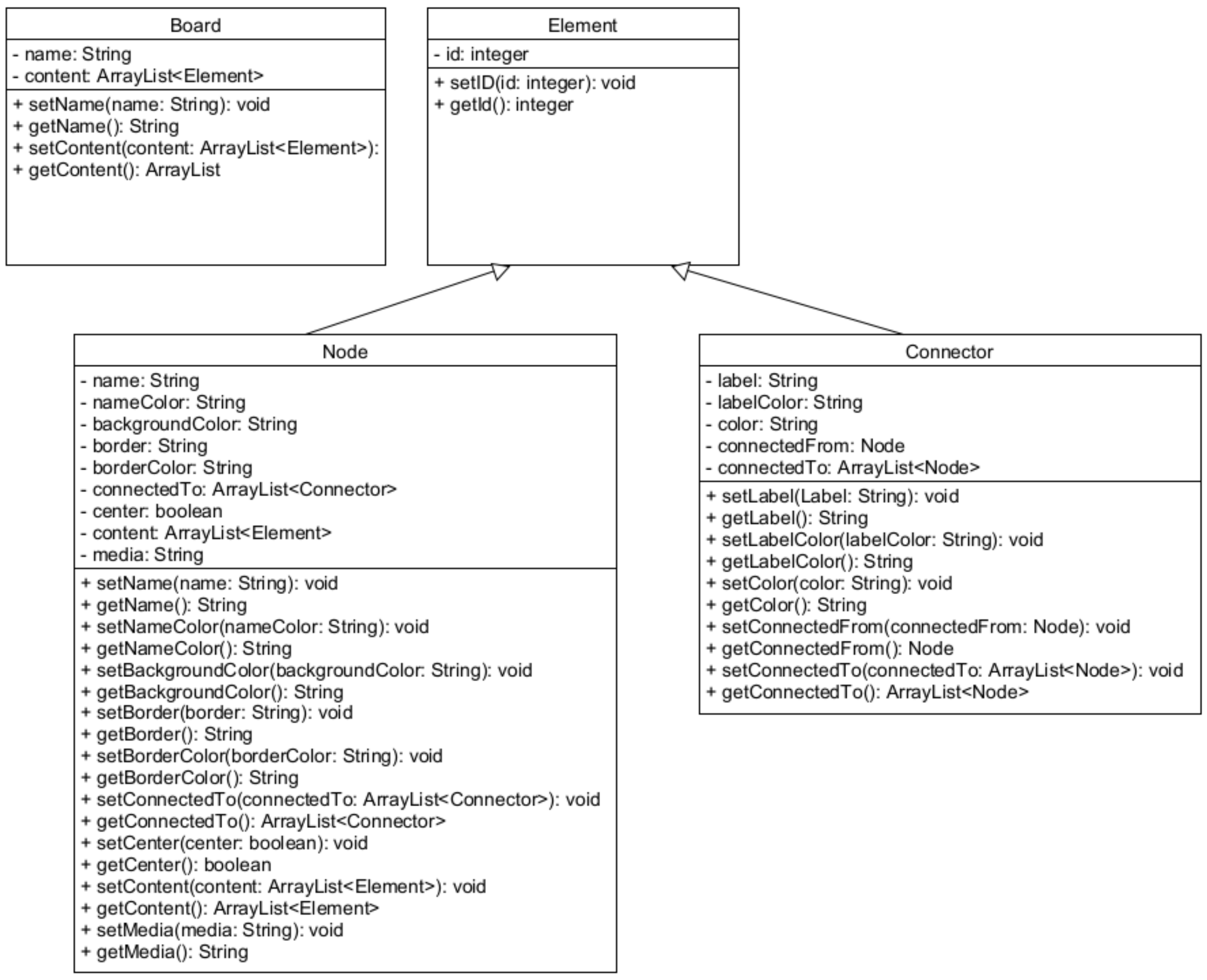

Board
- name: String
- content: ArrayList<Element>
+ setName(name: String): void
+ getName(): String
+ setContent(content: ArrayList<Element>):
+ getContent(): ArrayList
Element
- id: integer
+ setID(id: integer): void
+ getId(): integer
Node
- name: String
- nameColor: String
- backgroundColor: String
- border: String
- borderColor: String
- connectedTo: ArrayList<Connector>
- center: boolean
- content: ArrayList<Element>
- media: String
+ setName(name: String): void
+ getName(): String
+ setNameColor(nameColor: String): void
+ getNameColor(): String
+ setBackgroundColor(backgroundColor: String): void
+ getBackgroundColor(): String
+ setBorder(border: String): void
+ getBorder(): String
+ setBorderColor(borderColor: String): void
+ getBorderColor(): String
+ setConnectedTo(connectedTo: ArrayList<Connector>): void
+ getConnectedTo(): ArrayList<Connector>
+ setCenter(center: boolean): void
+ getCenter(): boolean
+ setContent(content: ArrayList<Element>): void
+ getContent(): ArrayList<Element>
+ setMedia(media: String): void
+ getMedia(): String
Connector
- label: String
- labelColor: String
- color: String
- connectedFrom: Node
- connectedTo: ArrayList<Node>
+ setLabel(Label: String): void
+ getLabel(): String
+ setLabelColor(labelColor: String): void
+ getLabelColor(): String
+ setColor(color: String): void
+ getColor(): String
+ setConnectedFrom(connectedFrom: Node): void
+ getConnectedFrom(): Node
+ setConnectedTo(connectedTo: ArrayList<Node>): void
+ getConnectedTo(): ArrayList<Node>

Figure 1. Core class diagrams for storing the maps as data structures.

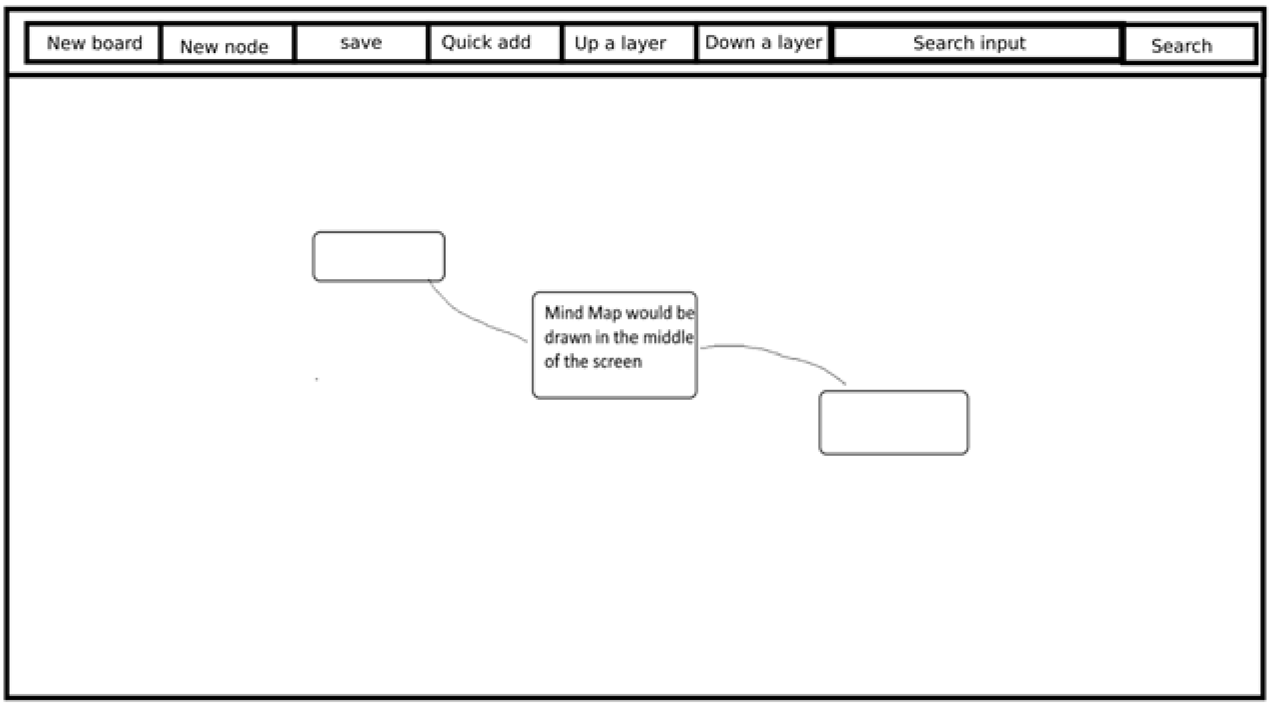


Figure 2. Software UI mockup

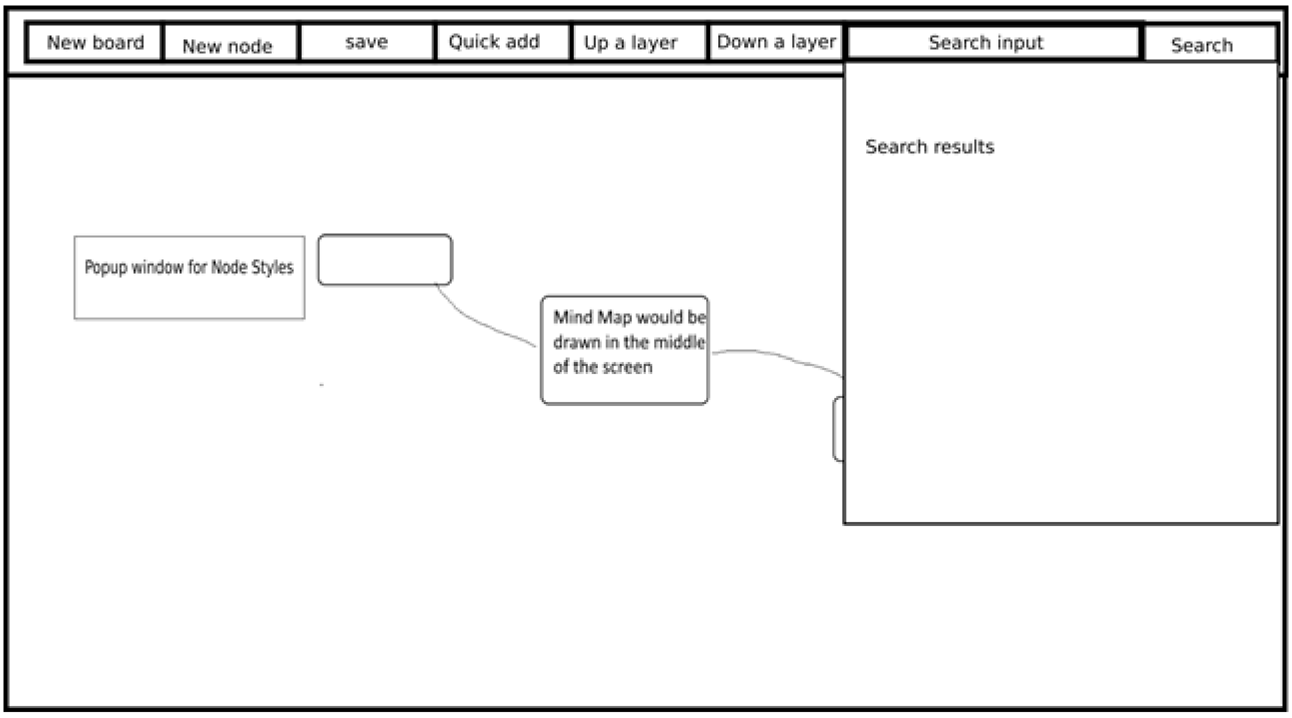


Figure 3. Software UI mockup with extra information

Implementation

My first step in implementing this software in code was to create the Board, Element, Node and Connector classes, as defined in the design phase earlier. This included overriding the toString method on Board, Node and Connector so that Node and Connector instances returned an array of their attribute values in order and Board instances returned an array of the arrays returned from Node and Connector instances. (Essentially allowing the content of the instances to be converted to array).

Later, other attributes were also added to the classes that were not initially planned. For Node, the content attribute was converted into two attributes called nodeContent and connectorContent, ArrayLists for their respective class instances so that it was easier to iterate over them and act on the contents without having to iterate over an ArrayList of the parent class and forcibly cast them down to the subclass based on what class the instance was originally made from. I also added a boolean called isRendered which checks if the instance had been rendered to the screen so when the program adds a new connector or node to the board and then iterates over the board content all instances that are already on screen are skipped over to avoid double rendering. Finally, I added doubles xCoord and yCoord which stored the current coordinates of the instance’s location on screen, so the position of instances was not reset upon saving and loading.

Connector had connectedTo and connectedFrom replaced with 2 attributes called node1 and node2 so they were reversible and the boolean isRendered added to avoid double rendering.

Board had the content attribute split into 2 ArrayLists nodeContent and connectorContent to avoid having to create ArrayList<Element> and then check the type of each instance and force cast down each time I needed to read the content attribute.

Then I began to implement some basic features such as the function to convert links from the text format to the HTML <a/> format (named linkfinder) as it was a simple string formatting function and the start of the search function. Initially linkfinder took a string as input and iterated over it, storing the index of each ‘[‘in the string in an array. Then for each index in the array, if the text after the ‘[‘contained any text followed by a ‘]’ then a ‘(‘then any text then ‘)’ it would attempt to make it a hyperlink. If the resulting link were invalid it would then try to make it a file link. The search function was coded to take a string to search for and an array of class instances to search in as input. It would then iterate over the instances, checking if their text matched the text supplied. If it did, they got added to a result array which was then returned These would later be updated as time went on.

Next, I implemented code to save the class instances making up a board to external files and to load them from external files. I then tested that saved class instances could be loaded without data loss. To save boards, an FileOutputStream is created using the name of the board given during board creation (and stored in the relevant Board class instance) as input. The FileOutputStream is then used to save the Board class instance to file. To load saved boards, a FileInputStream is created then the file is read into memory. Next, the footer is set to the loaded board’s name, then the screen is wiped clean and the contents are rendered to the screen. If the user is loading a file immediately after opening the software, the screen will already be blank anyway but to ensure no issues arise, the screen is cleared irrespective of that fact.

In order to facilitate saving and loading, the data classes were adapted to implement the Serializable interface.

The next major update involved developing the user interface (UI), using a VBox for the root node, a TitledPane for the toolbar and a HBox inside the toolbar to force the toolbar buttons to lay horizontally on the UI to match the design. Using a TitledPane allowed the toolbar to collapse again to match the design above. The new board button was programmed to trigger the creation of a temporary popup window that would allow for the resetting of the contents of the board on screen and the naming of the Board class instance this created.

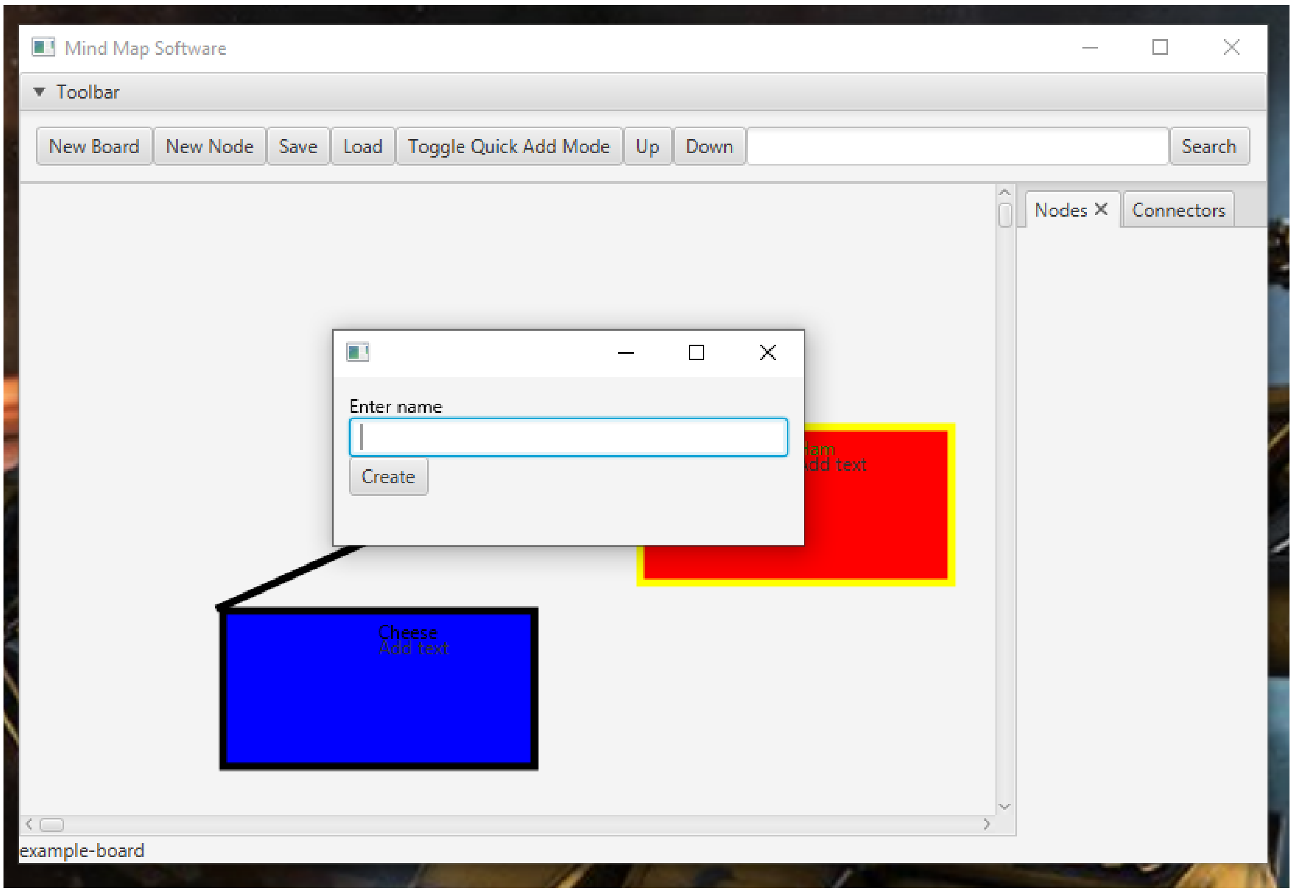
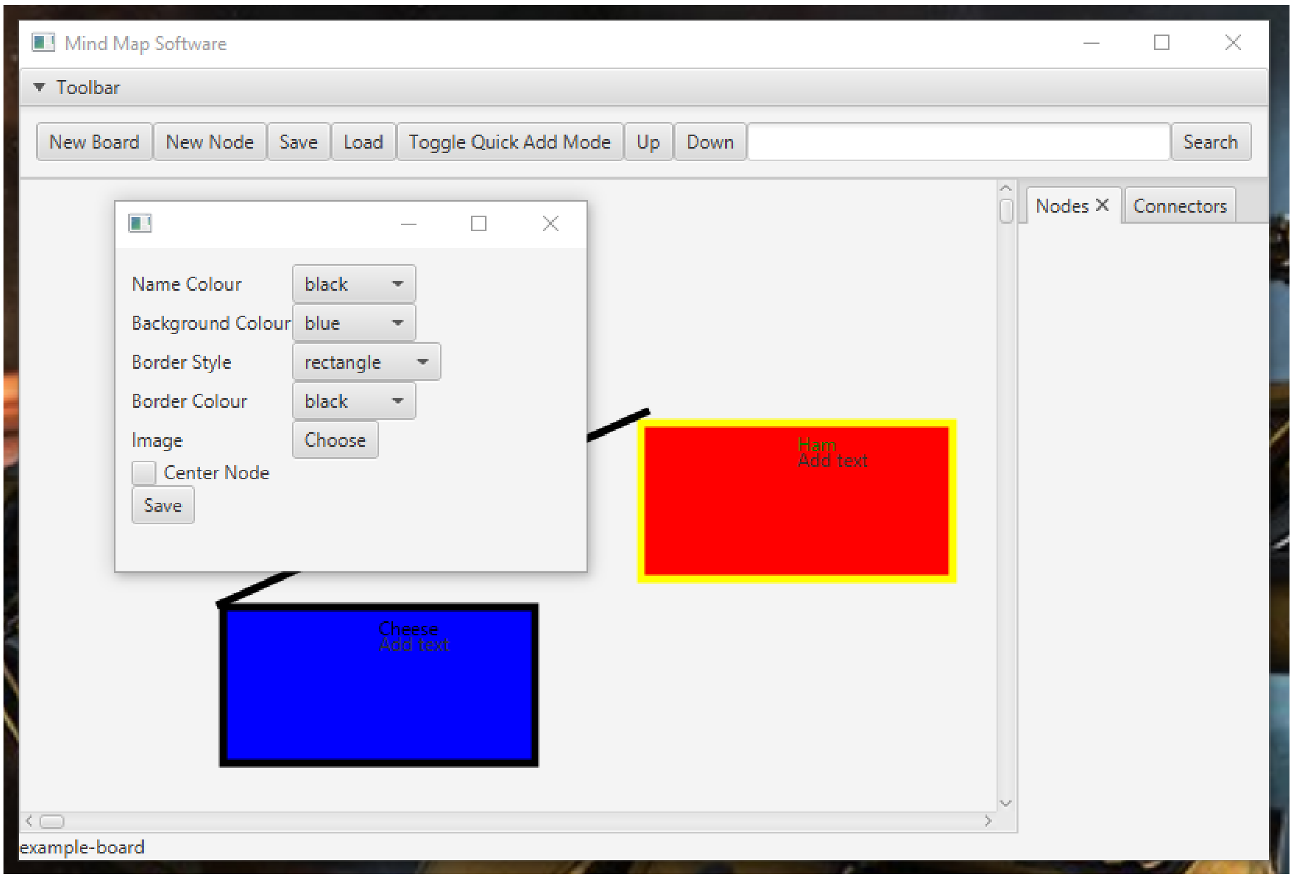


Figure 4. Screenshot of final UI showing the Board Creation popup window over a map

The new node button was programmed to create a new CustomNode instance, add the instance to the Board instance and then refresh the screen so the change was visible to the user. The save and load buttons were programmed to invoke the save and load functions from earlier with their required parameters. The quick add toggle button is programmed to switch colours between red and green when clicked. It was intended to signify if the ‘Brainstorming Mode’ was enabled but since the feature was not implemented it has no practical function. The up and down layer buttons were intended to change between layers of nested boards and nodes but again they have no function. The search button was programmed to take the input from the adjacent search box, run the search function on the active board with that input and insert the results into the UI sidebar. The buttons were placed in that order partly to match the initial design and partly for ease of use. The first button to press (New Board) is on the far left, where people will look first, followed immediately by the New Node button. Save and Load come next as they are quite common actions after board and node creation. They are also adjacent as they are related, as are the up and down layer buttons. The search input and button are on the far right so as to line up with the search results sidebar and all unmentioned buttons just fill in the middle.

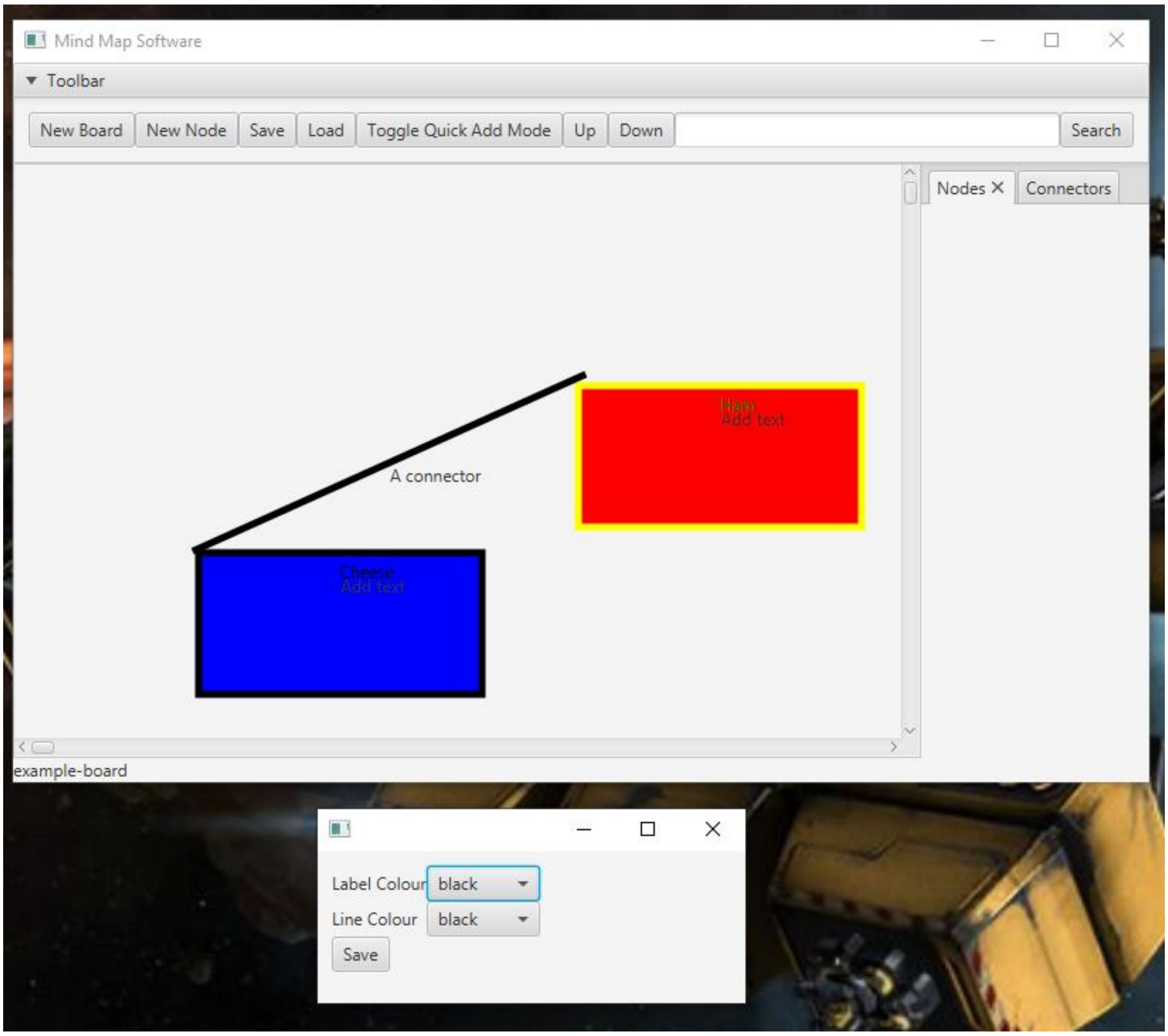
I then added the popup windows for styles and board creation. The style windows are separate Stages containing a GridPane with some Buttons, Labels and ComboBoxes. They contain a save button which collects the current state of the ComboBoxes and updates the class instance attributes to match.

Next, I linked together the created node and the node style window by setting up an onClick function for the graphical representation of a node. This meant that clicking on the node opened the style window.



Name Colour: black
Background Colour: blue
Border Style: rectangle
Border Colour: black
Image
Center Node
example-board

Figure 5. Screenshot of final UI showing the Node Style popup window over a map



Label Colour black
Line Colour black

Figure 6. Screenshot of final UI showing the Connector Style popup window over a map

At the same time, I changed how the style screen saved changes when the Save button was pressed so that it changed the value of the attributes in the class instance linked to the UI object, rather than the UI object's attributes. So, for example if mycolour = "blue" and myshape.setColor(mycolour) then rather than simply writing myshape.setColor(newcolour), it would instead set mycolour to, for example, red so that the value could be saved to memory with the save function. The ability to edit the text on nodes was added around this time, making use of an extra class that extended Label. This class, which I found online on StackOverflow written by a user by the name of Autumn_Wind (StackOverflow, 2021) wraps around the Label class and produces a JavaFX UI node comprised of a paired Label and TextField. By default, the TextField is invisible and the Label contains the text that was passed to it upon instantiation. When double clicked, the text in the TextField is changed to match the Label and the TextField is moved to the front. The user may then edit the text in the TextField. Then by pressing Enter, the Label is moved to the front and its contents set to match the TextField. Alternatively, by pressing Esc, the Label is moved to the front with no changes.

The next major update resulted in the creation of two new classes, previously not mentioned in the design, named NodePane and Connector Pane. These classes had attributes related to the UI representations of nodes and connectors and an attribute that linked them back to instances of the CustomNode and Connector classes. They ensured that front-end and back-end attribute values were synced by providing a function to sync them. This was an important feature as before now, the relationship between backend data and frontend UI elements was only one way (from back to front) and there was no way to know which backend class instance had resulted in which frontend element after creation. This I then had to refactor the whole software to make use of the new system, including switching from having one list of elements to two lists, one of connectors and one of nodes. This rendered all the previous test save files unusable.

After this, I made it possible to delete nodes on the UI. This is done by adding an EventHandler for mouse clicks and filtering it to ensure the click was a left click and was done while the user is also pressing the shift key to ensure they are not trying to move the node around. Once the program has ascertained that a deletion is wanted, the relevant NodePane instance is deleted from the UI and the connected CustomNode instance is deleted from the backend data storage arrays in the currently active board class. This also works for connectors. I also added ability to change the image shown on a node utilising a FileChooser to allow selection of a file then changing the values in the attributes related to images on both front and back end and finally creating a new Image instance from the image file given and adding it to the empty ImageView instance on the node. While I was working on this section, I also updated the style screens for both nodes and connectors to have default values and added some cosmetic padding for the images, so they did not accidentally overlap the node's border.

Later, I added labels for connectors which are EditableLabel instances positioned at the midpoint of the Line instance that makes up the connectors.

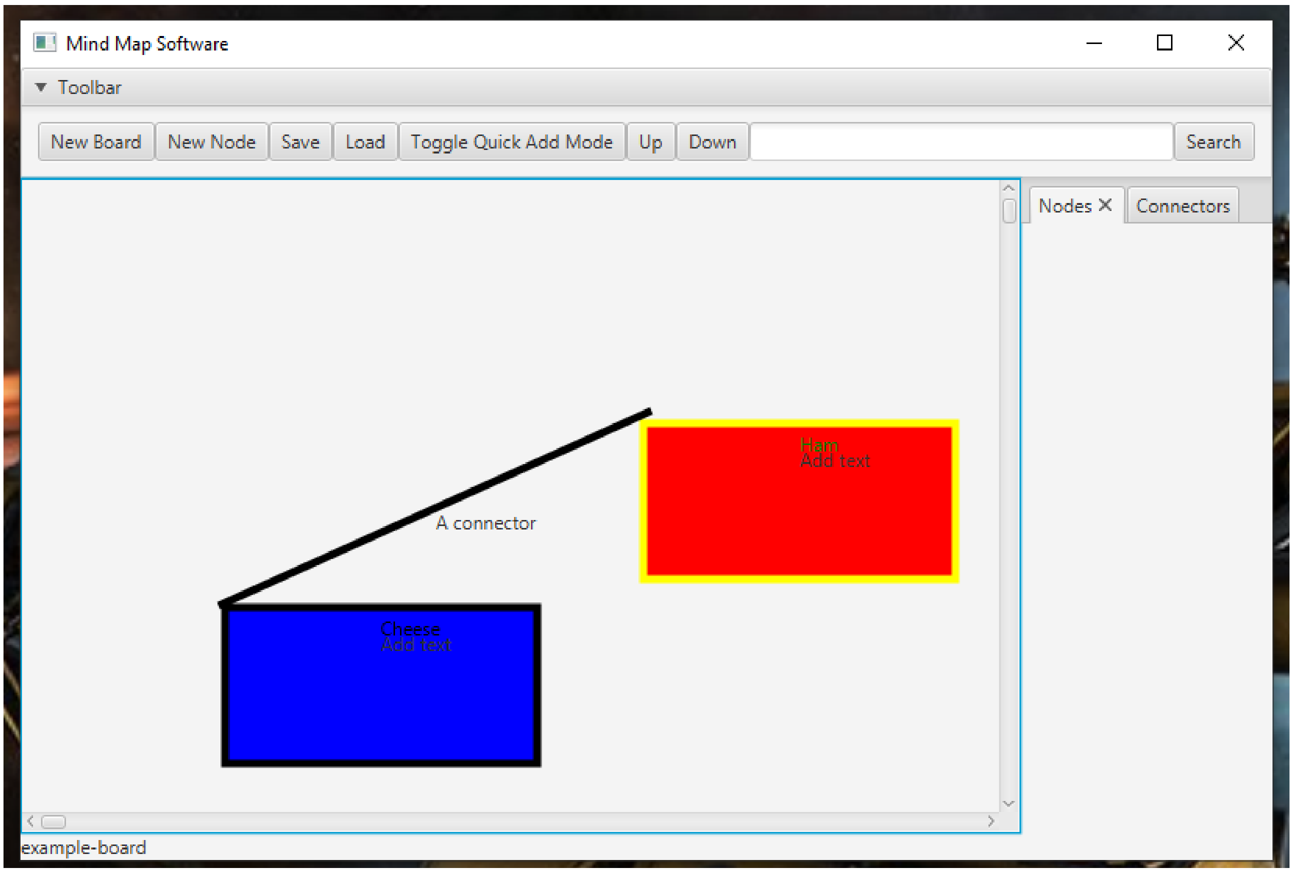


Figure 7. Screenshot of final UI with 2 different colour nodes with different titles and a labelled connector connecting them

At this point in implementing the software, I decided to create a set of instructions so when it came to writing this report, I would not forget the keyboard and mouse controls that I had previously set up.

Finally, I attempted to add dynamic re-rendering of connectors so that connectors would reposition themselves on screen when their respective nodes moves to maintain the illusion of connection. The code that I wrote to detect when nodes are moved and in response to the completion of a dragging movement, fetch all connectors connected to that node and move the end that touches the node to the node’s new location did not seem to have any positive effect and sometimes had a detrimental effect, so I deleted the code. I can only assume it was doing the wrong task the correct way and therefore raised no errors when the program was run.

Testing and Evaluation

Overall, I would evaluate this project as a partial success. Most of the features have been implemented, the prototype is functional, and the user interface matches the original design to a reasonable degree of accuracy. Some deviance from the design was required in order to work around issues related to the technology used.

Testing

Test Details

If a feature does not have a detailed test, that is either because it is a graphical feature that can be proven by looking at the user interface or because the feature is missing from the prototype and so cannot be tested beyond the fact that its absent.

Test	Actions Performed	Expected	Result
Node Creation	Pressed “New Board” button to create new board named “123” Pressed “New Node” button.	Software makes a basic node with title “Add name”, content “Add text”, no picture and black borders	Software makes a basic node with title “Add name”, content “Add text”, no picture and black borders
Edit Node Title	Double clicked on “Add name”, typed “Alice” and pressed Enter	Software changes text	Software changes text
Edit Node Text	Double clicked on “Add text”, typed “Hi” and pressed Enter	Software changes text	Software changes text

Cancel Node Title Edit	Double clicked on “Add name” and pressed Esc	Software exits edit mode with no changes	Software exits edit mode with no changes
Cancel Node Text Edit	Double clicked on “Add text” and pressed Esc	Software exits edit mode with no changes	Software exits edit mode with no changes
Change One Node Style	Right clicked on the node Used dropdown menus to change the border to blue Pressed Save.	Software opens the node style screen Software changes only the border to blue	Software opens the node style screen Software changes only the border to blue
Change Multiple Styles	Right clicked on the node Used dropdown menus to change the name to red, the background to green and the border to yellow Pressed Save.	Software opens the node style screen Software changes the name to red, the background to green and the border to yellow. The main text remains black.	Software opens the node style screen Software changes the name to red, the background to green and the border to yellow. The main text remains black.

Connector Creation	(Assume there exists at least 2 nodes on the board already) Holds Ctrl and clicks on the two nodes in sequence	Software creates a line that touches the top left corner of each node	Software creates a line that touches the top left corner of each node
Edit Connector Text	Double clicked on connector label, typed “Bob” and pressed Enter	Software changes text in label to “Bob”	Software changes text in label to “Bob”
Cancel Connector Text Edit	Double clicked on connector label, typed “Bob” and pressed Esc	Software exits edit mode with no changes	Software exits edit mode with no changes
Change One Connector Style	Right clicked on the connector Used dropdown menus to change the label to yellow Pressed Save.	Software opens the node style screen Software changes only label to yellow	Software opens the node style screen Software changes only label to yellow
Change Multiple Styles	Right clicked on the connector	Software opens the node style screen	Software opens the node style screen

	Used dropdown menus to change the label to yellow and the line to blue Pressed Save.	Software changes the label to yellow and the line to blue.	Software changes the label to yellow and the line to blue.
Inserting an image into a node	(Assuming that the .jpg file is in the root of the program directory) Right clicked on the node Clicked on “Choose” button and selected a .jpg file	Software opens the node style screen Software adds appropriately scaled image to node	Software opens the node style screen Software adds image to node at full size
Changing the image on a node	(Assuming that the .jpg file is in the root of the program directory) Right click on a node that already has an inserted image Clicked on “Choose” button and selected a .jpg file	Software opens the node style screen Software changes image on the node	Software opens the node style screen Software changes image on the node (There is nothing wrong with the change, but the size of the image is still too large)

Remove an image	There is no way to directly remove images from nodes. The user must create a new node without the image and copy the text over then delete and recreate the connectors	N/A	N/A
Move a node	Left click on a node and drag the mouse to another location on screen	Node moves on screen	Node moves on screen
Save board to file	With a board already on screen, press the Save button	Software saves a file with file name "*board_name.ser*" in the program root directory (Warning: This makes no allowances for duplicate file names)	Software saves a file with file name "*board_name.ser*" in the program root directory
Load board from file	Press the Load button Select a *.ser* file using the popup window	Software loads the contents of the board in the file onto the screen (Warning: This will overwrite unsaved boards)	Software loads the contents of the board in the file onto the screen

Search in board	Type search term in search box Press Search button	Software searches board and returns results in search result sidebar	Software searches board and returns results in search result sidebar
Set one core image	Right clicked on the node Checked the “Center Node” checkbox and pressed the “Save” button. Saved and reloaded the board	Software opens the node style screen Software reloads the board and moves the scrollbars to the middle of the pane and moves the node that was changed to the middle of the pane	Software opens the node style screen Software reloads the board and moves the scrollbars to the middle of the pane, but the node does not move
Set no core image	Do anything but press the “Center Node” checkbox	No change	No change
Collapsible toolbar	Click on the little black arrow at the top left of the toolbar	Toolbar collapses	Toolbar collapses

Table 3. Details of tests

Evaluation

This table takes the list of requirements from the specification and checks the software against each requirement. It differs from the tests earlier in that these evaluative tests do not explain exactly how it was tested, only whether it passed, and cover less quantifiable requirements like the flexibility of the software.

Specified Task	Achieved	Notes
No unnecessary button presses on controls	Yes	All physical controls are bound to either a single or double left or right click with the press of either Ctrl or Shift. The alphanumerical keys are not bound to any controls
Create nodes	Yes	Nodes will always appear in the top left corner by default
Edit node text	Yes	Double click to enter mode Enter to confirm Esc to cancel
Change node colours	Yes	Requires explicit confirmation via Save button. Does not update automatically
Add image to nodes	Yes	Image must be in same folder as other software files
Change image in node	Yes	Image must be in same folder as other software files
Remove image from node	No	All nodes are currently generated with an image for demonstration purposes. Disabling that will generate an imageless node. Choosing to not add a image to said node is the workaround.

Hyperlink support	No	N/A
File link support	No	N/A
Draggable nodes	Yes	N/A
Save to external file	Yes	File created in same folder as other software files
Export to cross compatible file format	No	N/A
Keyword search	Yes	N/A
Flexible	Yes	No added features prevent actions and other features from working. If the user is willing to spend the time to create such a diagram, any structure of mind map can be made.
Supports a core image	Yes	N/A
Supports no core image	Yes	N/A
Supports multiple core images	No	N/A
Supports recursive nodes	No	N/A
UI design is similar to other mind map software	Yes	N/A
Node creation bound to keyboard/mouse shortcut and to UI button	Partial	Creation only bound to UI button due to lack of available mouse buttons. The mouse I use does contain extra buttons, but they are not standard, and I do not expect users to need fancy mice to use the software
Node styling appears in popup window	Yes	N/A

UI buttons for moving between recursive node levels	Yes	Buttons are present but non-functional due to no support for recursive nodes
Infinite plane for map drawing	No	Plane is 10 times the size of the screen and traversable with scrollbars. Plane size can be adjusted with code changes.
Brainstorming mode	No	N/A
Horizontal quick action toolbar	Yes	N/A
Collapsible toolbars	Yes	N/A

Table 4. Overview of which Design specification goals were achieved and the degree to which it was achieved if it was a partial achievement

Limitations and Incomplete Features

This section covers the limitations of existing features of the software. This refers explicitly to issues that I was aware existed in the software prior to testing.

The software does not allow connectors to move around the screen when nodes connected by them are dragged by the user. A workaround for this is to move the node to where you want it to be, then enter edit mode on each connector in turn, copy the label to your clipboard, then delete the connector and recreate it in the new location.

The user interface styling windows had to be coded as separate windows rather than in the style of hovering semi-transparent panels just to one side of an on-screen box over the main portion of the user interface for several reasons. One, if I set it up such that hovering over boxes summoned the panel, the user would never be able to use the panel as moving the mouse to the panel would move it off the box and therefore trigger the removal of the panel.

A similar issue occurred with the sliding search result window. While I managed to make the software search the created boards for the given phrase and populate the search panel, I could not also link up the animation for the sliding to the same button as the animation and searching were initially each defined in their own files and had to stay in their files for access to class scope variables.

For this reason, the end prototype has the animation code disabled. The initial reasoning behind the animation as I will probably also discuss in the Requirements Specification sections, was to ensure maximum space for map drawing in line with giving the user as much free rein as possible.

Unimplemented Features

This section covers the features that were entirely not implemented and the reasons why.

The initial design idea called for the nodes to support HTML style links (indicated with Markdown-style formatting) and file links so that the limitations of the software could be surpassed and extended by external web pages and local files. This was not implemented in the final prototype due to difficulties in making the text on rectangles support any content except plain text. This meant the HTML links would never be active as they would not be treated as HTML. File links support was dropped due to being perceived as harder than the HTML links as they would have to share the same space and support an arbitrary number of links.

A ‘Brainstorming mode’ was also planned which would have allowed automatic creation of boxes and lines on the map in the most likely next wanted location. This was renamed to Quick Add in the final prototype but never implemented. Since I was unable to make connectors move their ends in response to their respective nodes moving, I was sure that creating a feature focussed entirely on moving connectors and nodes tangentially related to other nodes and connectors based on predictive logic would end in failure

Allowing for the nodes to contain other nodes and connectors, support multiple images and for several nodes to be in the middle of the map was not added to the software due to the complexity of dynamically moving the images and nodes around on the nodes and boards they were created on whenever a new image or node is added to the node or board

I was unable to set up a software-agnostic export system to allow other mind map, diagram or drawing software to import and open files created by the software as the method used to save the structures to disk are not one of the standard compatible file formats like XML or Markdown.

The ability for nodes to have different shape border and for connectors to have optional dashed lines and/or arrows at one or more ends was not implemented due to challenges with the graphical representations remaining clear to view.

I planned to only have pictures on the toolbar, but this was never implemented.

Conclusion

During this project I set out to test my limits and challenge myself by creating a piece of software more advanced than any of my previous pieces of software. I had never made software where the user could edit the contents quite as much as they can in this project. I chose to make software for making mind maps for two main reasons. One, I struggled to use existing mind map software and thought it would be interesting to try and make one I would understand better. Two, I did not fully understand mind maps so I had a genuine reason to need to research about them for the literature review, rather than just researching for the sake of filling the literature review. I hoped to achieve a fully functional mind map creating software with lots of features for customisation for users. In the end, some of the advanced customisation features proved challenging to implement, but I managed to implement the basic customisation and map creation features so the software is functional.

In order to complete the project and create the software prototype, after submitting my proposal for the project, I first had to research about mind map software in journal articles, books and other related media. I also installed and explored a selection of freely available or free with optional subscription mind map software to analyse the existing options. I prioritised exploring software praised in literature or software containing features like the features I was hoping to implement in my project. I then designed the back-end data structures and front-end user interface for my software inspired by what I researched and explored. I used the design to help me to program the software prototype to match the design. Finally, I tested the software to analyse to what extent the software matched the initial specification and find out what needed to be discussed in this report as limitations or missing features.

I believe that the approach that I took to doing this project was effective as the bottom-up approach to coding the software meant that later features were easier to add in due to them building off earlier features and functions. It also meant there was less need to go back and rework earlier code. I still did go back and rework code often but for the most part features were added building off others. The software that I have created is a proof of concept or minimum viable product as the implementation of features core to its functionality was prioritised over the refining of existing features and the implementation of advanced features that only enhanced the software’s capabilities but did not prevent it from performing its purpose.

If I were to start the project again, I would make more effort to plan out the classes and attributes required for the project as the final code consists of almost twice as many Java classes as was planned. Hopefully, this would result in less need to refactor large chunks of code mid-project. I am not opposed to iterative development, but I feel if it is possible to avoid confusion it should be attempted.

I am wondering if Java might not have been the best language to use for the project from the point of view of a more advanced developer who knows more about which languages and technologies are best for different types of project. Java was sufficient for this project, but several commercial mind map ‘software’ are now websites, proving that it is also possible to make similar projects using JavaScript and other web development technologies. This means that depending on the built-in functions of web development frameworks and the abstractions used in JavaScript some of the features I could not implement in my prototype might have been easier in JavaScript or another language. It is possible that by that logic; other features may have been harder to implement, and Java was indeed the right choice for me. It is just hard to be certain as I am not experienced at the difference between languages at the implementation level.

Further improvements to the project would be twofold:

Firstly, adding in all the features that I wrote into the specification but did not implement such as the recursive nodes, quick add mode, customisable shapes, software agnostic export system, multiple centre nodes, multiple images on one node, link support and a pictural toolbar. To implement recursive nodes, I would need to write a function that when given a node containing other nodes and connectors, saved the current board to memory (or possibly to file like normal saving) then parsed the node instance’s content attribute that contains the other node and connector instances and renders them to the screen and a pair of functions to wipe the screen and load the previous or next layer of nodes from memory assuming the earlier function had been used to view them. I would also need some way to create the first node inside a node if none existed inside the current node. Maybe if no recursive nodes exist in a node, pressing the keyboard shortcut to view the recursive nodes first offers the option to create one first, then proceeds as above. There are already buttons on the UI for moving up and down the layers of recursive nodes, they would just need hooking up to the functions. I would also add some reminder, perhaps on the footer as to which layer the user was on to avoid confusion.

To implement the quick add mode I would have to add an additional branch of code to the creation of nodes that checked if a Boolean was true and if so, upon the creation of a node, forced the user to edit the title and then the body of the node and then after editing was exited, created a new connector pointing from that node out into open space and generated a new node at the other end and then looped until the user pressed a button to stop. Since connectors made by the user can only be made between 2 existing nodes, the software would have to check the locations of all other nearby nodes and connectors and use the constructors for ConnectorPanes and NodePanes to generate them out of order in a appropriate location so they do not overlap with other elements. Upon instantiation of the NodePane, it would be linked back to the connector from earlier. The Boolean that activates this would be toggled by the Quick Add Mode button on the toolbar.

To implement customisable shapes, I would change the Rect attribute on NodePanes to Shape then enable the ComboBox in the style window to delete the current Shape instance and create a new Shape Instance depending on what was selected. In the event Java does not have a prebuilt subclass of Shape that is needed, I think it could be replaced with a Group of Shape subclasses, possibly Lines. Then the code for positioning of images, text and connectors would have to be changed so it dynamically adjusts to the shape to avoid overlaps.

To implement the software-agnostic export system, I would need to find out exactly what formats multiple popular software accept as imports and create a new Button on the toolbar that opens a new window. This window would function like Save except rather than serializing the classes to file, it would read all the attributes in the Board’s content attribute and construct a new file from scratch in one of a few formats that I have proven in research that other software can take. They might take CSV, or Markdown, or PDF or something similar.

To implement the ability for boards to have multiple centre nodes, I would change the centre node code so that if there existed multiple nodes in the Board’s nodeContent attribute, all such nodes were moved to the middle in different ways depending on how many there were to move. Since this movement would keep them attached to the other nodes they are connected to and so on by proxy down the board, this is essentially a board-wide dynamic refactoring program which would probably be challenging to create.

To implement the ability for nodes to have multiple images, I would have to change the media attribute of nodes into an ArrayList<String> and refactor the code for rendering the one image into code for iterating over the list and rendering each image stored within onto the node, scaling them dynamically to fit (Or, I guess, enlarging the node to fit instead depending on which made sense in the wider context). Then I would need to change the UI for adding an image to allow multiple files to be selected. It might be useful if the ImageViews used could be dragged around like nodes, but I think that would require code to prevent them from exiting the node and 'best fit' code to allow the images to move around each other to prevent overlap and that is not something I am sure I could implement.

To implement HTML links, I would run the node text through my linkfinder function which converts links in Markdown into HTML link format then replace the Label on the NodePane with a WebView to render the HTML. Since untagged text in HTML renders like normal text there should be no formatting issues.

To implement file links, I have no idea where to begin implementing them. I know that I can create a file chooser and bind that to a button. I am not sure how to then bind the chosen file to another button and ensure when the button is pressed, the file opens in the default program chosen for that type of file on the user's computer.

To implement a pictural toolbar – a toolbar where all buttons have no text labels, just pictures – would simply involve setting the button label to an empty string and setting the graphic of each button to an appropriate image. I am uncertain as to what would be the best image to use for each button though. For example, the most recognised symbol for saving is a floppy disk, even though floppy disks haven't been in widespread use for about 20 years.

Secondly, I would refine the existing code to fix bugs like the image scaling issue and node centring problems. The functionality would also be increased by implementing warning and confirmation dialogues for loading and saving to prevent the overwriting of unsaved boards and failing to save boards when a save already exists with the same name. Also, search would need to be improved as the ability to jump to the location of the results on the board is not proven to work for all scenarios. The styling for nodes might also need some tweaking so text and images are always evenly spaced and do not overlap the border and each other. This would involve tweaking some of the hard coded variables in node creation.

Appendices

A - Project Supervisor Meeting Logs

Date: 2/10/2020	Time: 15:30-15:50
Discussed concerns and worries about the project and ideas for the project. Agreed to have concrete ideas by next week	
Date: 9/10/2020	Time: 15:30-15:50
Discussed 3 ideas of varying scale Selected the most developed idea to take forward Supervisor prompts me to start writing the proposal and finding the articles/documents related to the theme I ask questions about the proposal. Gantt charts, and the resources section and promise to have part of the proposal written by next week.	
Date: 16/10/2020	Time: 15:30-15:58
Asked about Gantt charts, appropriate reference articles and how to improve the proposal Plan to have proposal draft complete by next week	
Date: 23/10/2020	Time: 15:30 – 15:54
Asked about sourcing from specific websites, referencing and formatting of proposal Agreed to fix problems with proposal before deadline	
Date: 30/10/2020	Time: No time specified
Asked several questions as written below - Do I submit copies of the articles I am going to use? - How do I chop up the software bar? - How do I do/start the literature review? - Are we going to meet biweekly from now on?	

- Should I submit it today? Agreed to fix problems with proposal before deadline	
Date: 6/11/2020	Time: 15:30-15:44
Asked supervisor to clarify exactly what I had to submit and what was the timetable for the next steps of the project. Asked how I should start the next step of the project. Goal for next week was to edit the proposal for the last time and submit it and to make notes on the literature I had sourced.	
Date: 13/11/2020	Time: 15:30-15:45
Asked what to do if a source I want to use has a quote from another source that I am also referencing Goal for next week was to fix the Gantt chart and continue literature research with notes and research on other mind map software	
Date: 20/11/2020	Time: 15:30-15:44
Discussed literature review and tactics for reading sources and was told to start making literature review draft	
Date:27/11/2020	Time: 15:30 – 15:50
Discussed how to improve tone and quality of literature review and avoid overuse of direct quotes and prevent plagiarism. Goal for next week is to reference the homepages of the software being referenced and remove editing comments	
Date: 4/12/2020	Time: 15:30-16:03
Discussed literature review Starting to write up the requirements section	
Date: 11/12/2020	Time: 17:00-17:26
Discussed literature review and design/requirements	

Goal for next week was to start proofreading the literature review, add a proper reference list and improve the design ideas with more detail	
Date: 17/12/2020	Time: 15:00 – 15:19
Queried about word count, references, line spacing, and reference dates Also queried about software features	
Date:8/1/2021	Time: Not specified
Asked about introduction and conclusion for literature review and putting the Gantt chart in the literature review Needed to start implementing the software	
Date:15/1/2021	Time: 15:30 - 15:55
Asked about help with challenges in the code Planned to have the back-end code done by next week	
Date: 22/1/2021	Time: 15:30 – 15:49
Asked for help on link recognition, file support and search UI. Asked if I needed to start the report yet Planned to have started the UI code by next week. Told to leave report to later	
Date: 29/1/2021	Time: 15:30 – 15:56
Asked for help solving some bugs in the code Planned to work on taking the help and feedback into account and improving the UI code and allowing the user to create and edit nodes and boards	
Date: 5/2/2021	Time: 15:30 – 15:56
Had concerns about self-plagiarism when using sections of proposal and literature review in the final report. Asked about help regarding ironing out issues with node styles, borders, movement and onclick functions Goal for next week was to allow the creation of multiple nodes on one board	

Also told to start on the report on week 21 (5 weeks in the future)	
Date: 10/2/2021	Time: 18:00 – 18:46
Asked for help to solve some bugs that rendered the project unusable if not solved. I needed to find a way to link UI elements to back-end class instances so when the user clicked on a node, the program knew exactly which instance to edit Planned to implement the changes suggested to solve the problems.	
Date: 12/2/2021	Time: 15:30 – 16:00
Updated supervisor on progress Was told to let them know if the nodes were still glitching around when dragged after I implemented a change they suggested.	
Date: No date given	Time: No time given
Asked for help on how to have the location of nodes on screen persist through save and load and how to delete a node Notes do not mention what I agreed to do for next week but I assume I worked on the above problems.	
Date: 19/2/2021	Time: 15:30 – 16:07
Discussed code bugs and resolved to add new features No more detail was written in my notes	
Date: 26/2/2021	Time: 15:30-16:22
Needed help with permanently deleting nodes and synching changes to Pane instances through the separate style window Planned to solve by the following week • Node deletion • Connector movement • Styling connectors	

• Delete connectors • Search • Core image search	
Date: 5/3/2021	Time: 15:30 – 16:03
Discussed how to get ConnectorPanes to save their styles to the backend, how to attach two event handlers on one button and how to enable dynamic movement. Also asked how to import javafx.scene.web.WebView for the HTML links Planned to fix minor bugs and start writing the report by the following week	
Date: 12/3/2021	Time: 15:30- 15:48
Asked if references went in the report appendix and if I needed to reference in the report code that I found on the internet and used in the project. By the following week I planned to have inserted the literature review and design and specification from earlier into the report and edited/extended it as necessary.	
Date: 19/3/2021	Time: 15:30 – 15: 54
Discussed the report as a whole and reflected on feedback Planned to have started the implementation and testing section with proper methodical testing by the next meeting. Supervisor asked that I send them a mostly complete draft on Wednesday Week 24 (14/4/2021)	
Date: 26/3/2021	Time: 15:30 – 16:00
Received more feedback on the report Needed to: • Explain what the tables I used were for • Add details of tests • Talk more about how I made the features	

• Write the evaluation and conclusion	
Date: 16/4/2021	Time: 15:30 – 16:30
Discussed improvements to the report and more reasons and details to add. Resolved to add these improvements to the report. Discussed the date of my viva and who would be my second marker. Put the date on my calendar.	
Date: 27/4/2021	Time: 15:30 - unknown
Asked for clarification on the title page, references, conclusion, evaluation, introduction and abstract before submission on 30/4/2021	

B Running Guide

I wrote this using JDK 11.0.8 and JRE 1.8.0_281 and VS Code Insiders build 1.56. There are dependencies but they are specified in the pom.xml file and handled by Maven 3.6.3.

To run the prototype, navigate to the folder named my-app in the code and type:

```
mvn exec:java.
```

C Controls

Interface Button Controls

- ‘New Board’ button opens a popup window to allow the user to create and name a new Board. The name of the board will appear on the footer and is also the default name for the save file linked to that board. This is essentially analogous to a Word document in Microsoft Word. A board must be active (created) in order to create boxes and lines on the mind map
- ‘New Node’ button creates a new rectangular box on the top left corner of the screen
- ‘Save’ button saves a file with the same name as the active board to the directory that the program is located in. If one already exists, the file is overwritten. (Warning: The software does not include version control or other systems to avoid accidental overriding of files. I suspect if a file not made with this software already exists in the directory it would also be overwritten if the user tried to save a board with the same name)
- ‘Load’ button opens a traditional file selection window and prompts you to select a file to load. Please only select files made with this software. The contents of the file are then loaded to the screen.
- To search the mind map currently on the screen for a word or phrase, type the phrase into the search bar (blank box to the left of the button marked Search) and press the button marked Search. A list of boxes and lines that contain that word or phrase, sorted by type will appear under the search bar. You can click on the button to automatically move the scroll bars to that location on the map.
- Left click on dropdown boxes and check boxes on style screens like normal.

Mouse/Keyboard Controls

- Left click on the ‘blank’ parts of a box and drag to move it around the screen
- Right click on a box or line to bring up the style screen for it
- Hold Ctrl and left click on two boxes one after the other to create a line or Connector between them
- Hold Shift and left click a box or line to delete it
- Double (left) click on text to edit the text
- Enter to confirm text edits
- Esc to cancel text edits

References

Buzan, T. and Buzan, B. (2010) *The Mind Map Book: Unlock your creativity, boost your memory, change your life*. Harlow, Essex: BBC Active.

Elhoseiny, M. and Elgammal, A. (2012) ‘English2mindmap: An automated system for mindmap generation from english text’, *2012 IEEE International Symposium on Multimedia* pp. 326-331. doi:10.1109/ISM.2012.103. Available at: https://ieeexplore.ieee.org/abstract/document/6424680 (Accessed: 29 October 2020).

Hiranabe, K. (2009) ‘Agile Modeling with Mind Map and UML’. Available at: http://files.agilerepository.webnode.com/200000000-56b3657aac/Agile%20Modeling%20with%20Mind%20Map%20and%20UML.pdf. (Accessed: 29 October 2020).

Kudelić, R., Maleković, M. and Lovrenčić, A. (2012) ‘Mind map generator software’ *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)* 3 pp. 123-127. Available at: https://ieeexplore.ieee.org/abstract/document/6272922 (Accessed: 29 October 2020).

Tsinakos, A.A. and Balafoutis, T. (2009) ‘A comparative survey on mind mapping tools’. *Turkish Online Journal of Distance Education*, *10*(3), pp.55-67. Available at: https://dergipark.org.tr/en/pub/tojde/issue/16913/176433 (Accessed 26 October 2020).

Stack Overflow (2019) *How do I create an editable Label in javafx 2.2.* Available at: https://stackoverflow.com/questions/25572398/how-do-i-create-an-editable-label-in-javafx-2-2 (Accessed 2 February 2021).