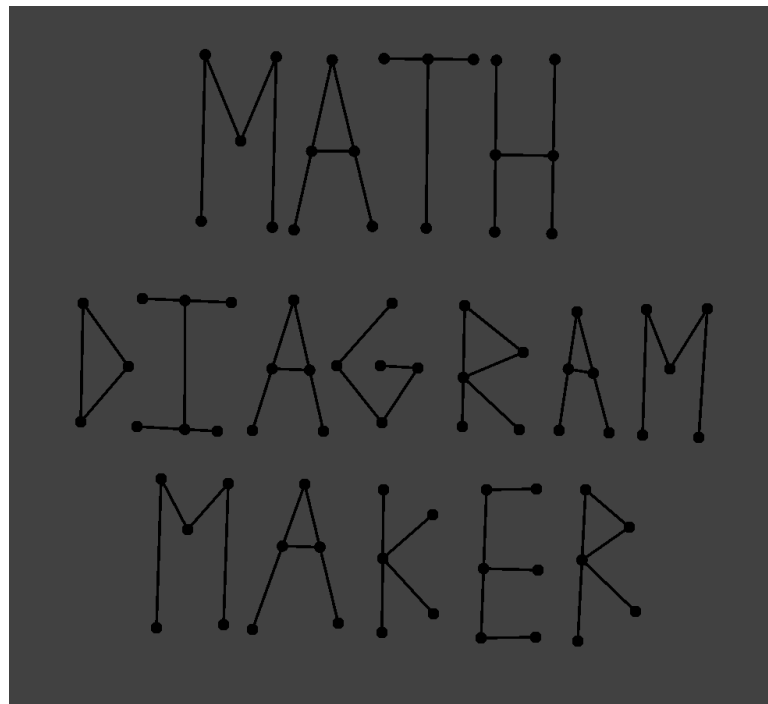




## Year 12 Software Engineering - Personal Project

### Math Diagram Maker



Student Name: Kobe Anderson

Date of Submission: 3/07/25

Github Link:

<https://github.com/RedWallaby/Math-Diagram-Maker>

<b>Math Diagram Maker</b>	<b>1</b>
1. Identifying & Defining	3
1.1 Problem Statement & Project Purpose	3
1.2 Project Purpose and Scope	3
1.3 Stakeholder Requirements	3
1.4 Functional Requirements	4
1.5 Non-Functional Requirements	4
1.6 System Requirements	4
1.7 Constraints	4
2. Research & Planning	5
2.1 Development Methodology	5
2.2 Tools and Technology	5
2.3 Gantt Chart / Timeline	5
2.4 Communication Plan	5
3. System Design	6
3.1 Structure Chart	6
3.2 Context Diagram	7
3.3 Level 1 Data Flow Diagram	7
3.4 Level 2 Data Flow Diagrams	8
3.5 UML Diagram - Diagram Features	9
3.6 Data Dictionary	10
4. Development & Implementation	11
4.1 Development Process	11
4.2 Complications and Bug Fixes	13
4.3 Interfaces	14
4.4 Version Control Summaries	17
5. Testing & Evaluation	17
5.1 Testing Methods Used	17
5.2 Test Cases and Results	18
5.3 Evaluation Against Requirements	20
5.4 Improvements and Future Work	20
6. Feedback & Reflection	20
6.1 Summary of Client Feedback	20
6.2 Personal Reflection	21
7. Appendices	21
7.1 Additional Tests	21

# 1. Identifying & Defining

## 1.1 Problem Statement & Project Purpose

A common issue with solving math problems is a lack of editable visual representation, as using pen and paper to draw diagrams limits the speed at which the problem can be solved especially if moving parts are involved or multiple visuals must be constructed. Drawing can also be inaccurate and cause visualisations to be faulty leading to potential mistakes in solving problems. Additionally if diagrams must be provided *for* a math problem such as in the creation of a geometry test, the visuals might be expected to be logical and to scale.

## 1.2 Project Purpose and Scope

This program aims to solve the issues defined in the problem statement by creating an intuitive and efficient diagram maker. This includes assorting points, lines, angles and circles in a freely customizable environment in order to produce high quality geometrical diagrams. These diagrams can be exported, saved and reopened to maximise efficiency and ease-of-use, allowing math problems to be visualised to aid in interpretation and provide visuals for test problems. Furthermore, the project will contain account and data management, which involves saving diagrams to certain users and creating accounts which are linked to a unique username and securely stored password. Users will be able to create accounts and enter the program through an authentication process that will return the users their saved diagrams for further editing.

## 1.3 Stakeholder Requirements

For this project, our stakeholders are math students and teachers who seek to use an efficient tool for diagram creation.

The student requirements are as follows:

- Program must be intuitive to use
- The program must be usable on low-end devices
- Diagram should allow components to be moved freely

The teacher requirements are as follows:

- Program needs account-based storage
- Program must allow downloading of diagrams
- Diagram placing should involve snapping to a grid to create aligned structures

Overlapping Requirements:

- Diagrams should incorporate core geometrical features including points, lines and angles
- Diagram must also include curved structures such as circles
- Diagrams should be savable and re-loadable
- Diagram components should be deletable

Conflicting Requirements:

- Students: Line length and angle size should be visible
- Teachers: Line length and angle size should be togglable and settable

### 1.4 Functional Requirements

- Menu Management:
  - Login and register users by storing secure account data
  - Implement main menu, diagram creator, save/load screen, login and register
- Diagram Creation
  - Add, Move and Remove diagram elements
  - Diagram elements include points, lines, angles and circles
  - Lines, angles and circles can be connected to points
  - Labels can be added and modified
- Saving and Opening
  - Diagrams are stored and linked to different users
  - Diagrams can be reopened and edited from load screen
  - Diagrams will be stored using JSON formatting and encrypted
- Exporting
  - While in the diagram creator, allow PNG exporting
  - PNG exports must border around the diagram elements

### 1.5 Non-Functional Requirements

- Performance (Given the system requirements)
  - Minimal lag should be present; free of lag spikes
  - Moving between menus should take less than 0.5 Seconds
- Reliability
  - Program should be error free
  - Edge cases should be managed gracefully
- Security
  - User passwords should be hashed and proper authentication should be implemented to prevent unwanted login
  - Input fields should be validated or sanitised
  - Saved diagrams should be stored under each user
- Interface
  - Menus must be intuitive
  - Should look acceptable and be free of generic 'unity appearance'

### 1.6 System Requirements

System Requirements:

- Minimum: 2GB RAM, 1GHz CPU, 1GB Storage
- Recommended: 4GB RAM, 1.5GHz CPU, 2GB Storage
- Windows operating system (Support for .exe files)

### 1.7 Constraints

With the development of the math diagram maker, certain constraints arise, particularly due to time and scope issues. This limits the implementation of features such as encryption. Encryption can only be implemented if the encryption keys can be stored on a server, however this project is entirely local and would require significant time to implement a server capable of handling and managing diagram keys. This would also go well beyond the scope of this project and in order to implement a persistent server, additional costs would be required to maintain its functionality.

## 2. Research & Planning

### 2.1 Development Methodology

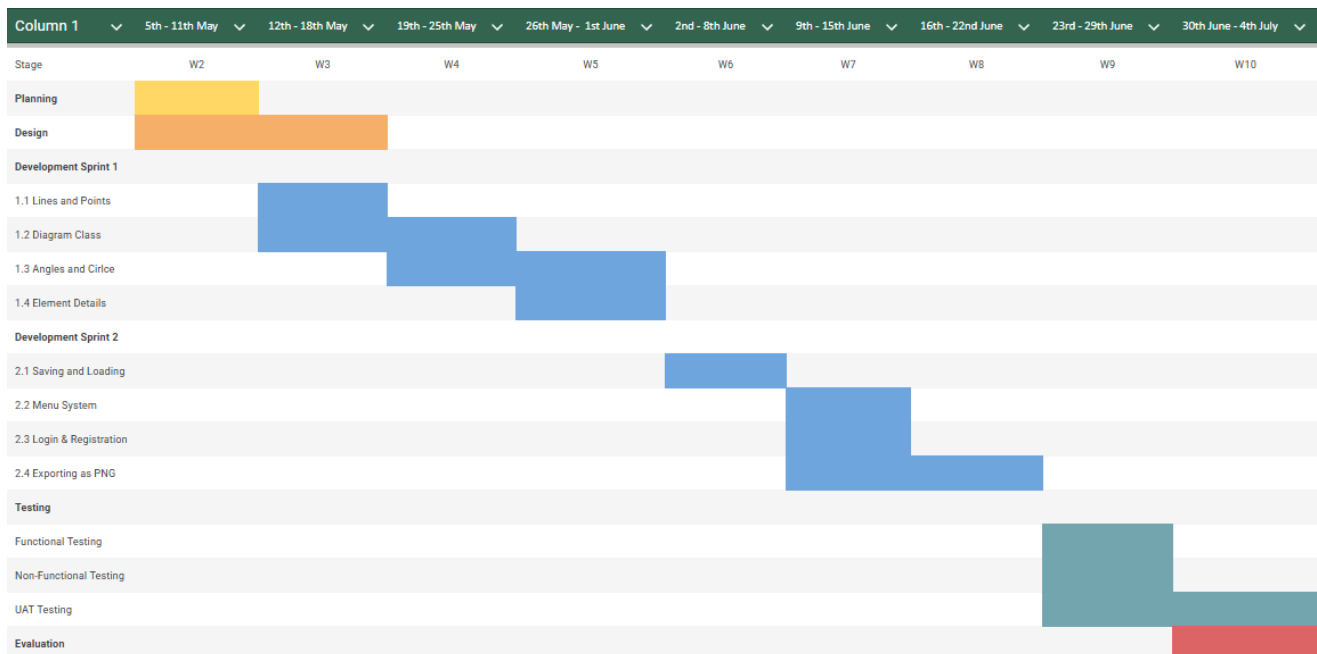
In order to develop this project, the AGILE methodology was used for its iterative and reflective design and implementation process. This allowed for the creation of two major 'sprints' or iterations in which a review of the project would be done to create feedback after the first sprint, allowing the project scope to be adjusted accordingly to match for unaccounted issues or integration complexities. The remainder of the project's creation similarly followed this methodology, including requirements gathering, planning and design and release.

### 2.2 Tools and Technology

This project was developed in the game engine or framework known as Unity. This engine is built using C++ but contains a C# scripting API for project development which was used to build this project. This framework is highly useful as it provides entire inbuilt systems for objects, hitboxes, coordinates and UI layouts which streamline the development process and remove the need to manually implement complex base systems. Furthermore, the IDE (Integrated Development Environment) known as Visual Studio was used along with additional tools such as GitHub Copilot to facilitate the development of this project's code. Additional packages were incorporated to simplify this process, including Json, for data management and storage, Linq for simplified looping in program code, and various Unity packages for integration with the frontend UI elements.

### 2.3 Gantt Chart / Timeline

A Gantt chart is a method used to plan the development of a software solution. This is used to visualise the timeline across which components of the project should be completed. For this project, the gantt chart shows the planning, design, development, testing and evaluation stages of which there are two different sprints, each with individual milestones. This is illustrated below.



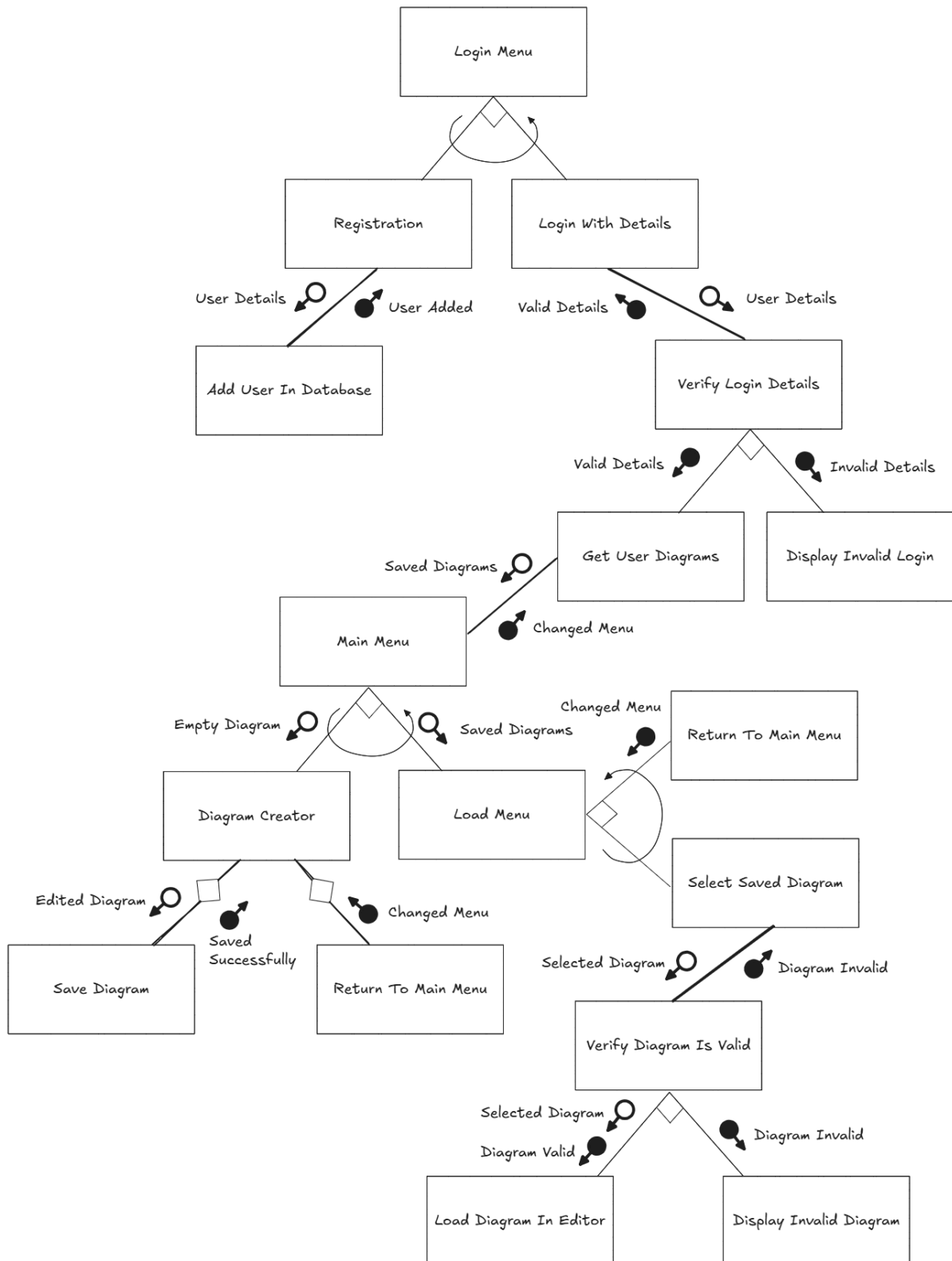
### 2.4 Communication Plan

Client engagement will be achieved through peer feedback and project reviews after each sprint in order to determine how well the project is progressing in relation to its stakeholder requirements. This information will be captured and used to improve the project by adapting the focus of the development process.

### 3. System Design

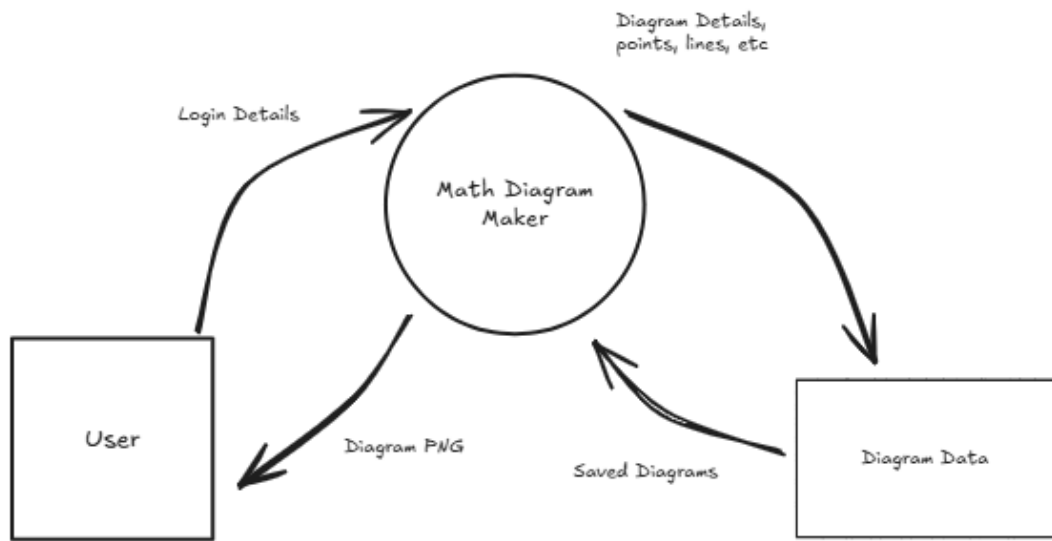
#### 3.1 Structure Chart

The following structure chart is used to illustrate the hierarchy of sub-modules used in the project's menus, showing the relationships and interactions between each one.



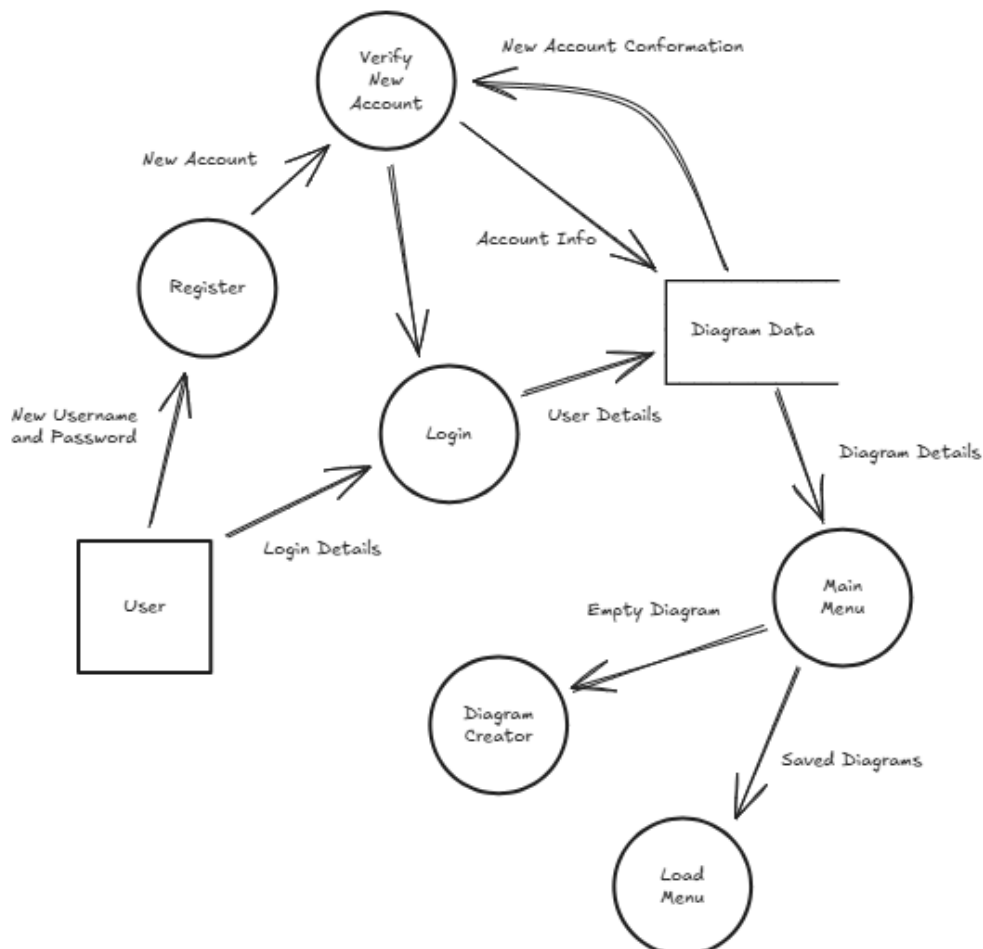
### 3.2 Context Diagram

The following context diagram visually represents the scope of the project, showing the overarching relationships between the user, program and data.



### 3.3 Level 1 Data Flow Diagram

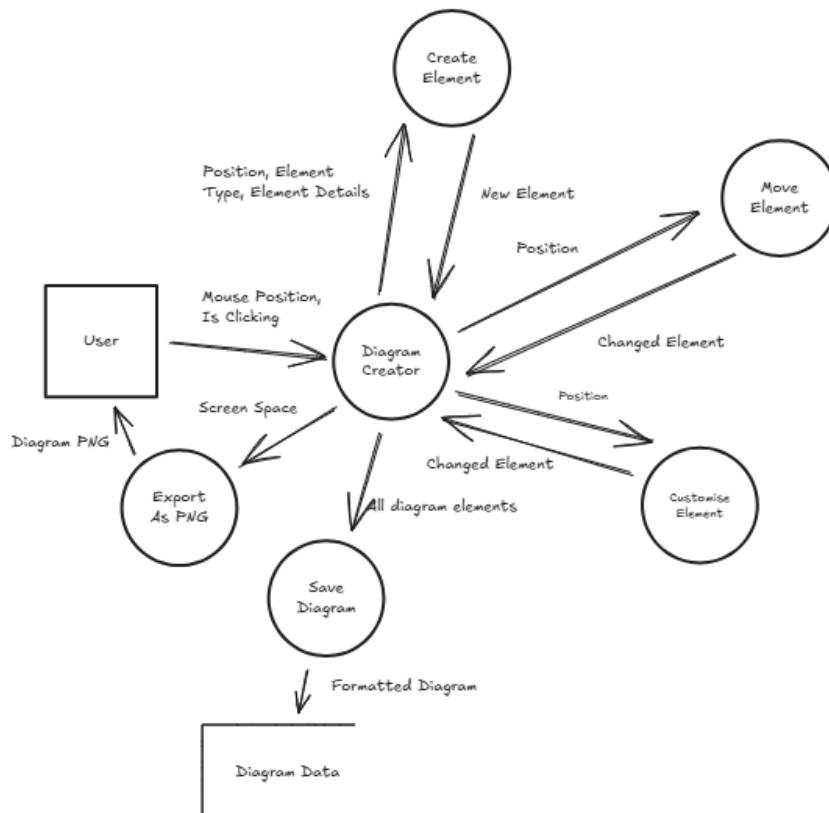
This level 1 DFD illustrates the flow of data within the system, including the login processes and main menu relationships.



### 3.4 Level 2 Data Flow Diagrams

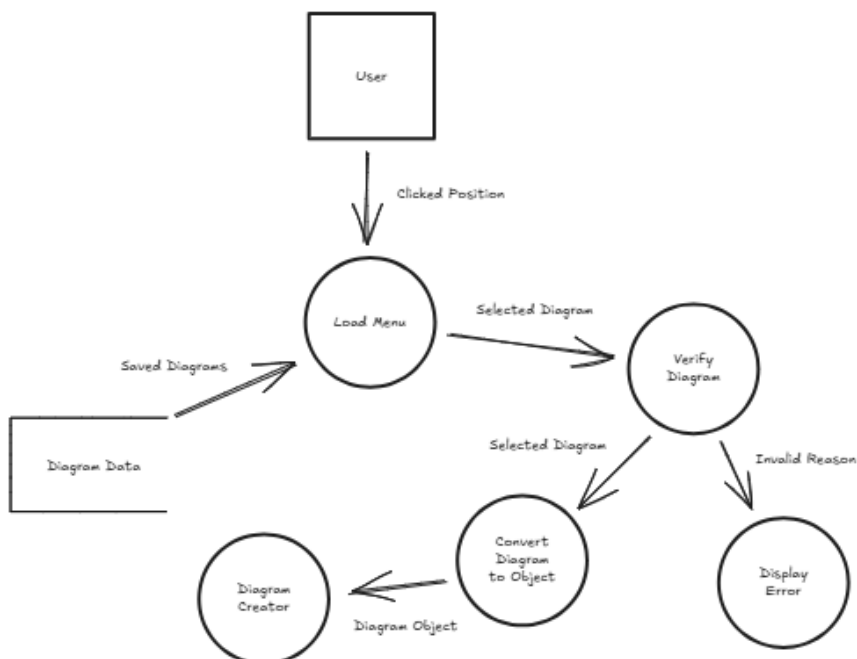
#### Diagram Creator

This DFD illustrates a specific process seen in the Level 1 DFD. It visually represents the flow of data within this process, showing how data moves within the diagram creator itself.



#### Load Menu

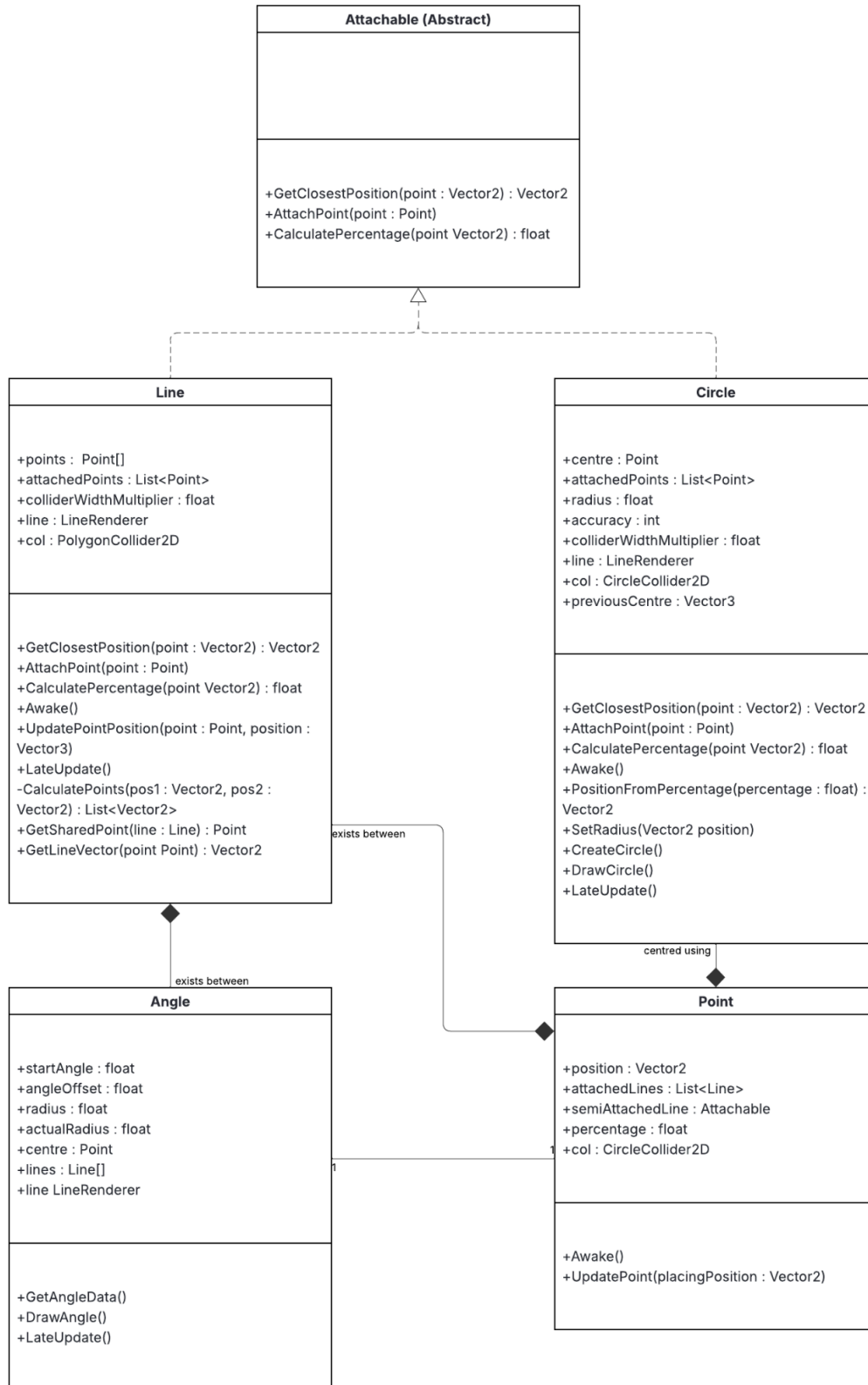
Similarly, this DFD represents the flow of data within the load menu.





### 3.5 UML Diagram - Diagram Features

A UML diagram is a crucial component of object oriented design, representing the structure of the system by displaying the attributes and operations of specific classes and highlighting their relationships. This diagram shows a small section of the system, describing the project's diagram element classes.



### 3.6 Data Dictionary

The following data dictionary contains a small section of the program's metadata, used to explain the properties and functions of certain variables within the project.

Field Name	Data Type	Size/Format	Description	Example Value	Constraints
radius	float	6-7 digits	Defined radius of a Circle object	3.5f	Must be >0
percentage	float	6-7 digits	Indicates a Points' percentage position between two 'ends' of a line or circle	0.77f	Must be between 0 and 1 (inclusive)
accuracy	integer	3 digits max	Determines the number of line segments used in a Circle or Angle object	360	Must be >0
LineCreator.placing	enumerable	1 digit	Determines the current placing stage (none, point or line) of the line creator	point (1)	Must be between 0 and 2 (none to line)
PointCreator.placing	boolean	True/False	Determines if the point creator should be placing	True	True or False
mousePosition	Vector3	3 floats (x,y,z)	Indicates current position of mouse in word space	(5, 5, 0)	All coordinates must be defined
lineWidth	float	6-7 digits	Length for any lines in the Circle or Line objects	2.0f	Must be >0
colliderWidthMultiplier	float	6-7 digits	Multiplies the hitbox size of the Line and Circle objects	3.0f	Must be >0
angleOffset	float	6-7 digits	Is defined by the shortest angle between an Angle objects' non-centre points	36.43f	Must be between 0 and 360 (inclusive)
moving	bool	True/False	Determines if the MoveSelector should be updating object positions	False	True or False

## 4. Development & Implementation

### 4.1 Development Process

The development of this project occurred over two main sprints and involved the creation of both the backend and frontend systems. The project's development began with the formation of the diagram creator and its sub-processes, followed by the creation of consecutive menu systems and the login/registration modules. A shortened and simplified summary of the entire implementation process (in order) can be seen below.

#### 4.1.1 Lines & Points

1. Point Implementation

The Point class and Prefab (front-end version of point) were added.

2. Line Implementation

Similarly the Line class and Prefab were added. The line was included to automatically update its hitbox.

3. Point Creator

Currently just a white box, the point creator could be clicked to instantiate new points. These points would snap to lines and would place on mouse click.

4. Line Creator

The line creator could be clicked to initiate line placement. This line would be placed between two selected points.

#### 4.1.2 Diagram & Diagram Editors

1. Main Diagram Class

The main diagram class was added and used to control the current diagram editor. It would later have much more functionality and include a majority of helper functions.

2. Diagram Editors

The *abstract* class DiagramEditor was added and would allow the point, line and future element creators to easily integrate with the diagram class through *inheriting virtual* and *abstract* methods. IPointerClickHandler (Unity Programming *Interface*) was used to detect mouse clicks with each editor.

3. Move Selector

The move selector was added as another DiagramEditor and would allow Points (and hence Lines) to be moved after placement. It would later gain the functionality to change the radius of circles.

#### 4.1.3 Angles, Circles & Attachables

1. Angle Implementation

Angle class and Prefab were added. Angle would be later changed to update its hitbox only after placement or one of its attached points moving, however at this state it updates constantly.

2. Circle Implementation

Circle class and Prefab were added. Similar to the angle, the circle updates constantly.

3. Angle Creator

The angle creator could be clicked to initiate angle placement. It would then place an angle between 3 selected points.

#### 4. Circle Creator

The circle creator would place a point (or select one) at a certain location then create a circle from that point to the user's cursor. Clicking again would lock the circle.

#### 5. Attachables

The *abstract* class Attachable was added and Lines and Circles would *inherit* from this class. Its purpose was to implement *polymorphism* by allowing Points to attach to other elements regardless of their type. It was used to simplify and streamline the code.

### 4.1.4 Elements & Labels

#### 1. Element Class

The *abstract* class Element was implemented and would allow Points, Angles and Attachables to *inherit* methods so that the Label system could use *polymorphism* to treat any diagram element as the same.

#### 2. Label System

The label system was added to toggle and override the labels of any diagram element. It also brought the functionality of deleting these elements.

This concluded the end of the first sprint, meaning that the project was reviewed and feedback was generated. This feedback revolved around reconfiguring the order of the second sprint, for example, the menu system was changed to be constructed after the saving/load system to improve the smoothness of integrating menus with the diagram framework and the scope was adjusted to remove encryption from the second sprint.

### 4.1.5 Saving & Loading

#### 1. Json Saving

Saving was largely implemented through Unity's inbuilt json serialisation. This meant new classes for the diagram (JsonDiagram) and each element (JsonElement, etc) needed to be implemented and would contain only the information that needed to be saved. Another class known as the SaveManager would then convert each actual object into these classes and serialise them into a single file.

#### 2. Diagram Loading

Similarly, the class LoadManager would deserialise these files back into actual diagram objects to be rendered in the diagram.

### 4.1.6 Menu System

#### 1. Menu Manager & Main Menu

The main menu was created through unity's scene editor and the MenuManager class was added to move between menus.

#### 2. Save Menu

A new front-end menu was developed which was linked to the SaveManager class through Unity's button components. These components use C# events to run all listening methods when the button is clicked.

#### 3. Load Menu

Another menu was implemented with support from Unity's auto-tiling components to automatically arrange multiple saved diagrams in a grid. This menu was similarly linked to the back-end using buttons that activate methods of the LoadManager class.

#### 4. Editor Panel

A proper front-end for the diagram creator was produced, including icons for each DiagramEditor and a button to access the save menu.

#### 4.1.7 Login & Registration

##### 1. Hashing

The C# Cryptography namespace was implemented to allow passwords to be hashed using the SHA256 hash function.

##### 2. Login Menu

The login menu front-end and back-end were implemented. The LoginManager implemented secure authentication by validating user input and comparing it against stored usernames and hashed passwords.

##### 3. Registration Menu

The registration menu front-end and back-end were implemented. The RegistrationManager would allow users to create accounts which store each user's username and hashed password. It similarly implements input validation and prevents duplicate usernames from being created.

#### 4.1.8 PNG Exports & Code Cleanup

##### 1. Exporting as PNG

A new button in the editor panel was implemented to convert the diagram into a png by reading from pixels within a determined area of the diagram.

##### 2. Notification System

A notification system was implemented to explain how to use each DiagramEditor and inform users of other significant information.

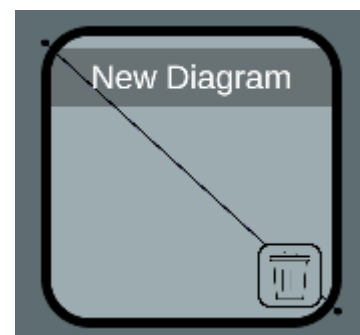
##### 3. Code Cleanup and Finishing Touches

Finally, to finish the implementation of the project, the code was cleaned up, inefficiencies were removed, docstrings were added and access modifiers were adjusted to ensure proper *encapsulation*.

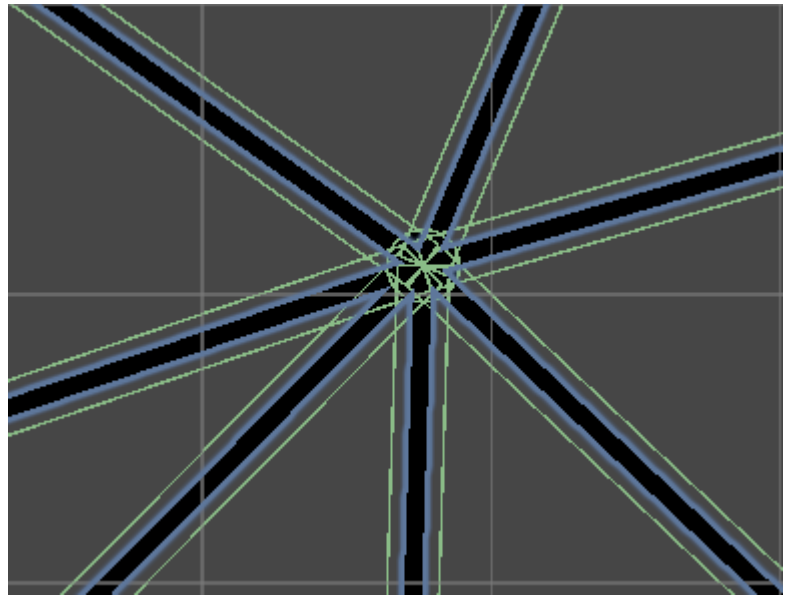
#### 4.2 Complications and Bug Fixes

Throughout the development of this project many complications and roadblocks arose and there were a multitude of edge cases that needed to be fixed. The following shows three of these.

- The Unity material being used to display the previews of diagrams in the load menu, did not have the correct shader in order for masking to be applied. This resulted in the preview clipping out of the masked zone as seen on the right. To fix this, the material shader had to be manually changed to "UI/Default".



- The Diagram.GetElement method did not prioritise certain elements over others so in the example shown on the right it would have been completely impossible to interact with the centre point directly.
- Entering a diagram name with over ~170 characters would result in a file path error (“Could not find part of the path”) and as such input validation had to be added to the save menu.



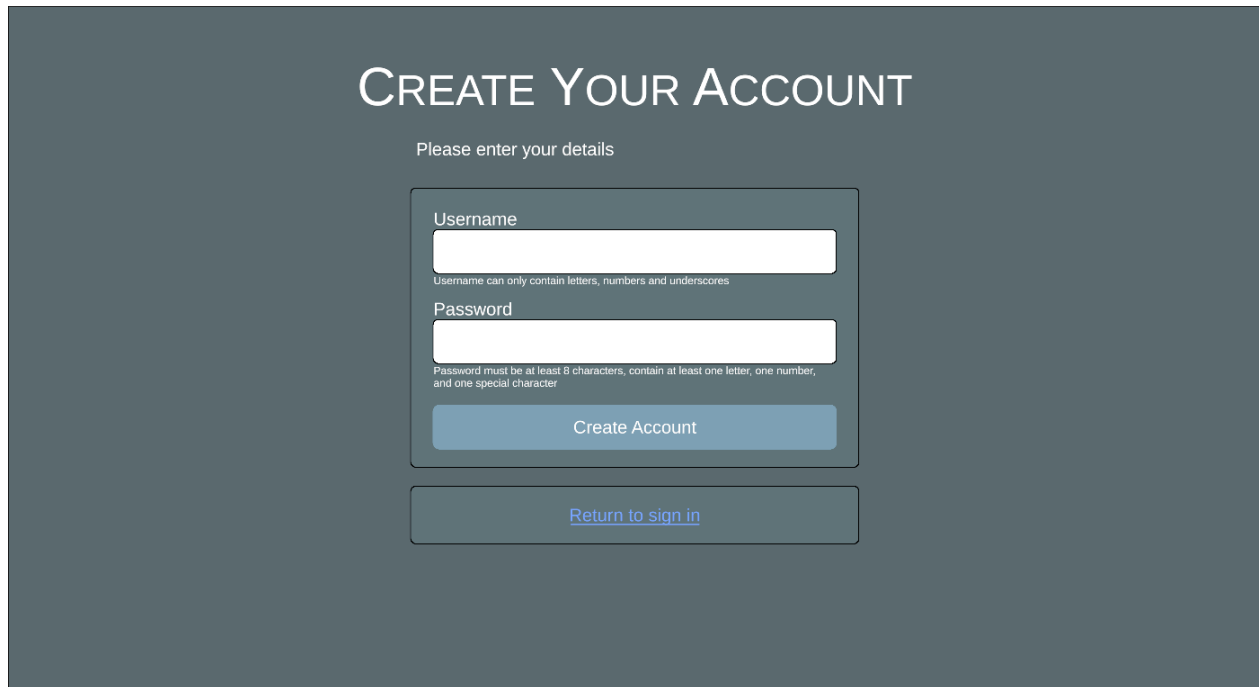
## 4.3 Interfaces

### 4.3.1 Login Menu

The image shown is the login menu which allows existing users to sign into their accounts to access the rest of the program and load their saved data. It contains two input fields for the user's username and password and two buttons, one for going to the sign up menu and the other for confirming logging in. Putting in an invalid username or password will cause an error message to appear.

#### 4.3.2 Registration Menu

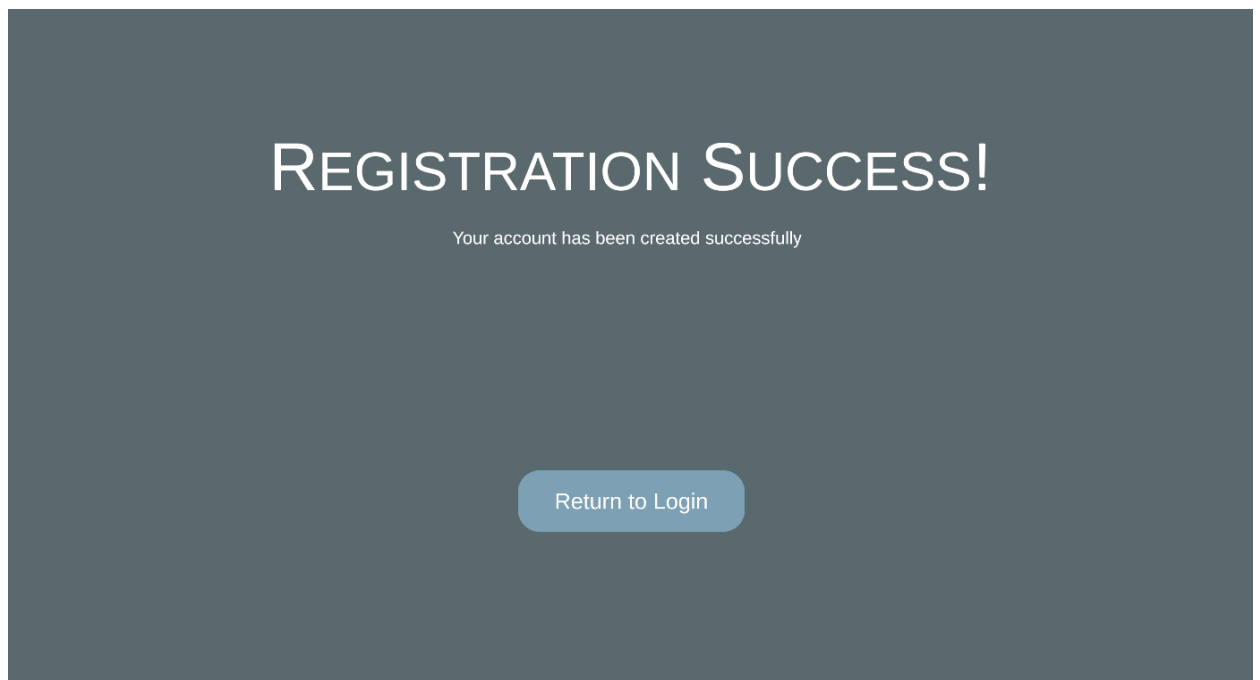
The following interface is the registration menu which allows users to create an account. It similarly contains two input fields for the username and password and two buttons, one for returning to the login screen and the other for confirming account creation. Accounts created here must adhere to the validation rules stated beneath each input field, otherwise an error message will be displayed.



The registration menu interface features a dark gray background. At the top, the text "CREATE YOUR ACCOUNT" is displayed in a large, white, sans-serif font. Below this, the instruction "Please enter your details" is shown in a smaller, white font. The main form is a light gray rounded rectangle containing two input fields. The first field is labeled "Username" and has a small text hint below it: "Username can only contain letters, numbers and underscores". The second field is labeled "Password" and has a hint: "Password must be at least 8 characters, contain at least one letter, one number, and one special character". Below the password field is a blue button with the text "Create Account". At the bottom of the form is a light gray button with the text "Return to sign in" in blue.

#### 4.3.3 Registration Confirmation Menu

The following menu is used for informing the user that they have successfully signed up. It contains one button allowing the user to return to the login page.



The registration confirmation menu interface has a dark gray background. It features the text "REGISTRATION SUCCESS!" in a large, white, sans-serif font. Below this, the message "Your account has been created successfully" is displayed in a smaller, white font. At the bottom center of the screen is a blue button with the text "Return to Login" in white.

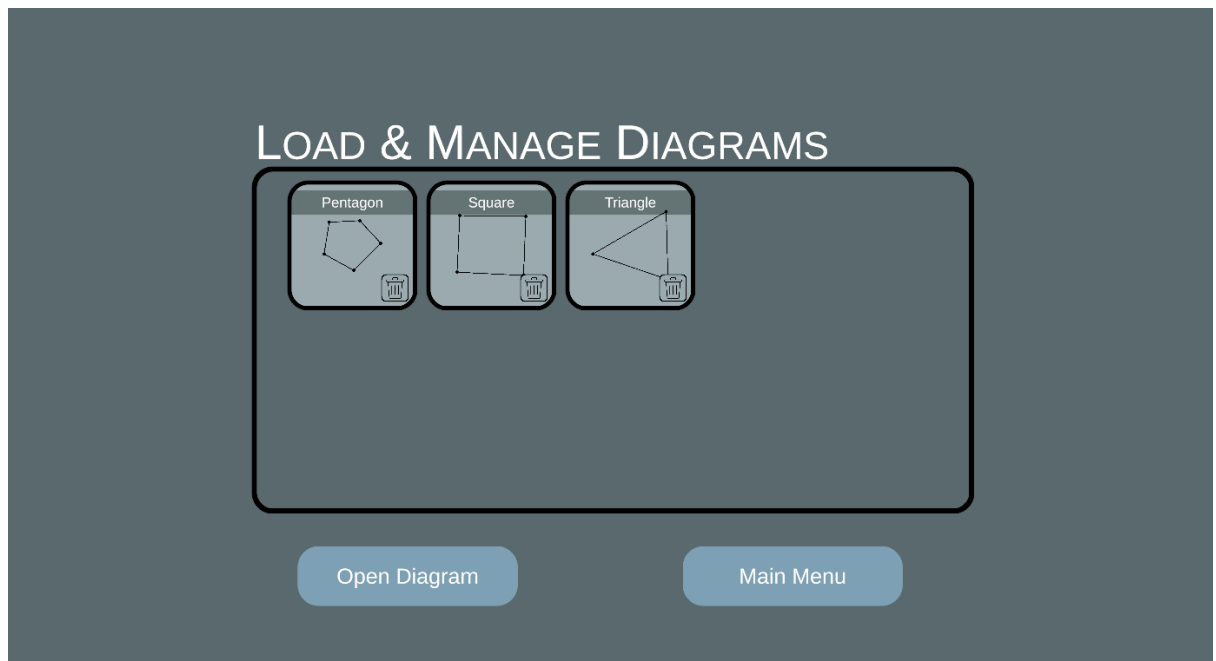
#### 4.3.4 Main Menu

This is the main menu which acts as a hub for users to access each part of the program. It contains 3 buttons, one for opening a new blank diagram, one for taking the user to the load menu and the other for quitting the program.



#### 4.3.5 Load Menu

The image shown below is the load menu used for loading the user's saved diagrams. It contains a variable number of selectable diagram previews, depending on the amount of saved diagrams the user's account has. The section in which these are kept is scrollable allowing for users to see all of their saved diagrams. It has two persistent buttons, one for opening the selected diagram and one for returning to the main menu.





#### 4.3.6 Editor Panel

The editor panel is shown on the right and is used to access the DiagramEditors, save the current diagram to a png or access the save menu. It is located on the left side of the screen when in the diagram editor.

#### 4.3.7 Save Menu

The menu shown below is the save menu and is responsible for saving the current diagram. It is only accessible when a diagram is open. The menu contains an input field which is used to customise the name of the diagram. It has three buttons, one for saving the diagram and setting its name, one for returning to the main menu and one for closing the menu which re-enables the editor.



#### 4.4 Version Control Summaries

The commits and their summaries can be viewed here:

<https://github.com/RedWallaby/Math-Diagram-Maker/commits/main/>

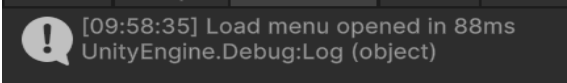


## **5. Testing & Evaluation**

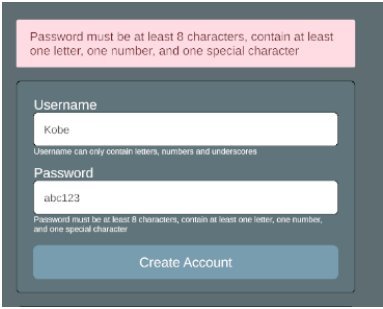
### 5.1 Testing Methods Used

In order to properly test the project and ensure edge cases are covered, a range of testing methods have been used. This includes the following:

- Unit Testing  
This involves testing individual components of the program in isolation in order to verify that each component behaves as expected.
- Black box testing  
Black box testing is a method that assesses the functionality or performance of the running program without acknowledging the internal structure of the system being tested.
- Integration testing  
This is a testing method that involves combining and testing multiple modules to assess their interactions and ensure they function correctly as one system.
- User Acceptance Testing (UAT)  
This is commonly regarded as the final stage of software testing as it checks to ensure the entire program meets its user requirements and is ready for release.

## 5.2 Test Cases and Results

Testing Type	Description	Expected Result	Actual Result	Pass/ Fail
Unit Testing	Minimising menu load times	<0.5s	Peak time of 0.088s across 15 tests 	Pass
	Clearing Diagram	Diagram.elements is cleared fully without any null elements, diagram should appear empty	Diagram length becomes 0, all elements in diagram are removed from scene and memory	Pass
	Centralising camera	Camera should be moved to average position of all elements	Camera x and y positions are set to average of all x and y positions of elements	Pass
Editor/Label System - Black Box Testing	Toggling Labels	Label element should hide/show on button press even if label is overridden	Text object successfully created on initial toggle, hidden and shown on consecutive button presses, overriding label does not change functionality	Pass
	Move elements with labels	Moving elements and attached points should cause element labels to change accordingly	- Moving point from (0,0) -> (1,0): label updated - Line of length 2 -> 3: label updated - Circle of radius 1 -> 2: label updated - Rotated point from 90 -> 180 degrees: label updated  	Pass
	Placing angles	Angle should be created between 3 selected points and start rendering after 2nd selected point	- <b>selecting same point twice causes angle to not render</b> - angle is created between initial two selected points and cursor position - Clicking locks angle to third point	Fail
	Moving line-attached point	Selecting and dragging a line attached point should snap the point to the nearest position on its line	- moves relative to cursor position along attached line - updates attached element positions when moving point	Pass
Registration/Login - Black Box Testing	Inputting password with length of 60	Registration form should display password is too long error	Error appears at top of screen displaying "Password is too long (maximum 50 characters)"	Pass
	Inputting password with	Registration form should display error informing	Correct Error appears at top of screen	Pass

	only alphanumeric characters	user that a special character is required		
	Inputting username with hyphens	Registration form should display error informing user that hyphens cannot be included	Error appears at top of screen displaying "Username can only contain letters, numbers, and underscores"	Pass
	Registered matching Username and Password are input into form	Input form should hide and show the main menu	User is successfully logged in using their username, account data is successfully loaded, new menu page is shown	Pass
Integration Testing	Correct communication between Load Manager and Diagram	The Load Manager should communicate with the Diagram in order to gather diagram textures and open the diagram with a user-specified choice	Integrating both modules together results in: <ul style="list-style-type: none"> <li>- Load manager renders multiple diagrams within one frame</li> <li>- Diagrams are distributed across generated load objects</li> <li>- Load objects can then be selected and opened into diagram</li> <li>- Load menu hides upon diagram open and updates load objects upon re-opening</li> </ul>	Pass
User Acceptance Testing	Menu Intuitiveness	All menu features should be intuitive to use, allowing users to see the program respond in a way they would expect	A large majority of the software solution is intuitive, however not <u>all</u> components are some include: <ul style="list-style-type: none"> <li>- No in-game tooltip for how to open the label menu</li> <li>- Selecting diagrams is not obvious on which diagram is currently selected</li> </ul>	Fail
	Diagram Creation	The diagram should meet all non-functional requirements and should be error free	The diagram successfully implements points, lines, circles and angles. Objects can be added, deleted or moved and can be connected to each other. Labels can be added and modified	Pass
	Performance	The program should be efficient even on low-end devices	The program has no lag spikes and the maximum duration for menu loading does not surpass 0.5 seconds.	Pass
	Security	The software should successfully implement proper authentication, validation and hashing.	Login, Registration and Save menus implement validation to prevent errors and secure user data. Proper authentication is implemented and includes hashing to further secure user accounts	Pass

### 5.3 Evaluation Against Requirements

The software system has met each stakeholder requirement and additionally completed most functional and non-functional requirements. However, certain specifications have not been met, including removing the 'unity-look' of the project as many of the menus utilise default unity sprites that have simply been adjusted or rounded. Additionally, encryption was not possible in the scope of this project as mentioned in *1.7 Constraints*, as this would require constructing a server and external storage system in order to prevent the encryption key from being accessed locally. Furthermore, the project has mostly stayed within the scope of the requirements, a few exceptions apply including, deleting diagrams and diagram controls (moving and zooming). Overall, the program has closely followed its requirements and was only particularly influenced by the scope of the project.

### 5.4 Improvements and Future Work

If the scope of the project was increased or another development sprint was added another range of features could be added and some current features could be upgraded. This includes the following.

- An animated background for the main menu and improved visuals overall, particularly in reworking current sprites.
- Colour changing for diagram elements which could be implemented by reworking the label system to add additional options.
- Shapes, in particular polygons, could be added as another diagram element and could be used to display the area (in units<sup>2</sup>) of an area enclosed by a set of lines.
- Elements could be added to be placeable or editable by customising their length or size from a user given value. This could be done through creating additional DiagramEditors and through upgrading the MoveSelector class into a complete EditingSelector.
- Another feature that would be highly useful would be the ability to log in as a guest. This would streamline the process for users who simply want to access the diagram maker for simple projects.
- The ability for the user to specify the download path for exported PNGs. Either through another menu with an input field or through bringing up file explorer.

## **6. Feedback & Reflection**

### 6.1 Summary of Client Feedback

Upon gathering feedback from one of the project's clients, the following recommendations were provided.

- Saving to PNG notification text should say "Diagram exported to png" not 'saved as'
- Tooltips should be present to inform users of certain 'hidden' functionalities such as right clicking objects to open the label menu, alt + click + dragging to move the camera, scrolling to zoom and holding shift to lock to the grid.
- A warning should be added to inform users if their diagram is not saved before returning to the main menu
- There should be a popup to confirm if a saved diagram should be deleted
- Saving diagrams should have a 'save as' feature to save a new copy of the diagram with a different name

## 6.2 Personal Reflection

Throughout the creation of this program I have been exposed to a variety of challenges causing me to find workarounds and create solutions involving new concepts that ultimately improve my understanding of programming languages and the Unity game engine. Some of my personal developments can be seen below.

- I Developed a stronger understanding of OOP principles through exploring new ways of implementing abstraction and virtualisation  
I Learned how to...
- Use json in unity and understand its severe limitations (without additional libraries)
- Use getters and setters, particularly over abstraction
- Use delegates and lambda functions
- Use Unity RenderTextures as well as convert those into PNGs
- Create a scroll-zoom effect that keeps the same point on the cursor
- Create custom hitboxes using Unity's polygon collider

## 7. Appendices

### 7.1 Additional Tests

Editor System - Black Box Testing	Confirm point placement	Point should follow cursor and snap to non-point elements	Cursor moved around diagram: - snapped to closest point on line or circle when touching hitbox - on click: point data is updated, attached element data is updated	Pass
	Confirm line placement	Line should snap to points, lines or circles on start and end of line placement	On first and second click: - Line snaps to all points, lines and circles if touching hitbox - New point is created if not clicked on existing point <b>- Selecting same point twice causes line to connect to a position without a point when moved</b>	Fail
	Confirm circle placement	Circle should snap to points, lines or circles on initial placement and change radius based on cursor position	On first click: - Circle snaps to points, lines or circles if touching hitbox - New point is created if not clicked on existing point After first click: - Radius adjusts to distance between clicked on/created point and cursor - Clicking locks radius	Pass
	Confirm non-line attached point movement	Selecting and dragging a non-line attached point should move to cursor and snap to non-line attached points, lines and circles	After selecting point: - moves to cursor position - snaps to non-line attached points, lines and circles if touching hitbox - recursively updates attached element positions when moving point	Pass
	Confirm line attached point movement	Selecting and dragging a line attached point should snap the point	After selecting point: - moves relative to cursor position along attached line	Pass

		to the nearest position on its line	- recursively updates attached element positions when moving point	
	Confirm circle movement	Selecting and dragging a circle should change its radius to the distance between its centre and the cursor	After selecting circle: - changes radius of circle to distance between its centre and cursor - recursively updates attached element positions when moving point	Pass