SCC 210 Design Report - Group 26

Dexter Latcham 38380447
Alden Ong 38828774
Hoang Ky Trinh 38461226
Jake Edwardson 38392224
Adil Wazeer 38535122
George Sanderson 38436876

18th November 2022

Contents

1 Motivation

Our decision to produce a raster art editor was based on several group decisions. Primarily, we chose this project over a vector editor as we believe this will present us with some interesting challenges to solve. Features we intend to implement such as the line or shape tool rely on the rasterisation of a vector shape onto a pixel grid. Additionally, we feel that if this project is successful, a lightweight and OS-agnostic editor would become genuinely useful to us. Lastly, creating the editor ourselves allows us to tune different components to our liking such as save frequency, design tools and UI layout.

2 Desired Features

Our goal is to create a lightweight pixel editor that can be used to create simple sketches. The application should also implement our favourite editor tools from other applications. At the same time we want our editor to be easily extensible for us to add any tools we may want later. Usability is also a core requirement for us as the application needs to both suit our needs and be preferable for general use over existing editors.

3 Software Requirements

Functional

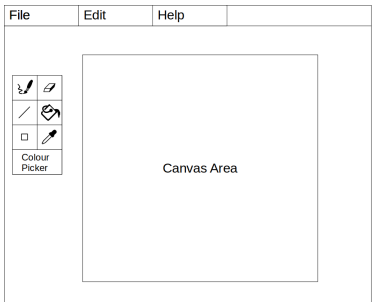
1. The program should launch and feature a blank canvas of size 500x500
2. Upon clicking the File tab, a drop-down menu of options should appear
3. Clicking new project should prompt the user for confirmation then wipe the canvas
4. Clicking save should prompt the user to enter a filename and location to store the canvas
5. Clicking undo should revert the canvas to the previous edit
6. If no changes have been made since using the undo function, this should reapply the last operation.
7. Upon clicking the edit tab, a drop-down menu of options should appear
8. Clicking canvas size should allow the user to change the dimensions of the canvas
9. The canvas size should never exceed 1000x1000
10. Clicking flip(vertical) should allow the user to flip the canvas
11. Clicking flip(horizontal) should allow the user to flip the canvas

12. Clicking rotate(clockwise) should rotate the canvas 90 degrees right
13. Clicking rotate(counter-clockwise) should rotate the canvas 90 degrees left
14. Clicking the help tab should display a dialogue box with information on the editor
15. The pen tool should allow the user to draw a think line below the cursor
16. The brush tool should apply the chosen colour across a circle of pixels beneath the cursor whilst the left click is held
17. The line tool should allow a user to click to select a starting point, then drag the cursor to the endpoint of the line and release it to draw it on the screen
18. The eyedropper tool should allow the user to click on a pixel of the canvas, select its colour for later use
19. The Eraser tool should reset pixels to their default colour as the user drags the cursor over them
20. The bucket fill tool should replace a contiguous area of like pixels with the selected colour
21. The shape tool should allow the user to draw from a set of basic shapes (square, circle etc)

Non Functional

1. The application should feel responsive, and operations should be applied in <300ms
2. Working with the application should feel intuitive, how user options are laid out should feel familiar to users coming from other editors.
3. The applications memory footprint should be kept to a minimum. Unnecessary resource allocation and references to unused objects should be disposed of when finished with.
4. Multi-threading should be used where possible to ensure fluidity and efficient resource use. Background processes like saving changes while editing should not interrupt the user as they interact with the editor.
5. Internally, the editor should be easily extensible with explicit structure and requirements as to how new tools and features should be implemented. Future contributors should also be able to readily call upon existing methods where needed to avoid code duplication.
6. The application should feel responsive, changes to the canvas should happen almost instantly after a user action.

4 UI-Mock Up



We iterated through several designs when planning our user interface. Initial designs featured a 2 column-wide block of tools on the left of the application (pictured right). This arrangement, inspired by early versions of ms paint, would allow for more tools to be presented to the user in a compact menu. This UI element was later redesigned to instead display the tools in a single column. This makes the application more usable as the user doesn't need to worry about overshooting the tool they are attempting to select. To facilitate this change, we migrated the colour functionality to a menu on the right of the display.

At this point we grouped other functionality into their respective navigation menu. Figure 5 and 6 show these menus with their contents as well as the confirmation box displayed before a new document is opened.

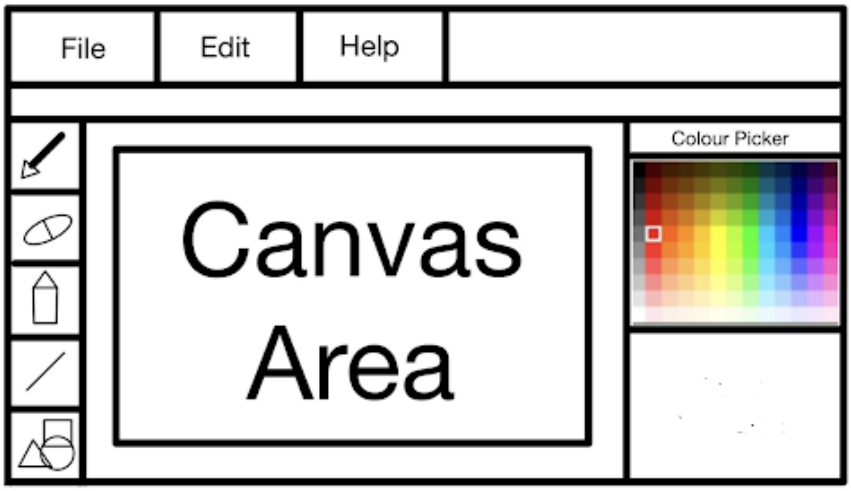
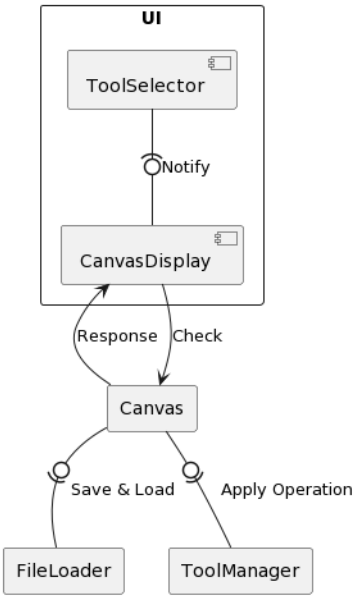


Figure 1: UI mock up

5 Code Design

Our canvas will be stored as an object with a 2x2 array of "Pixel" objects. Each pixel object will contain colour information as well as a method to change these values. This architecture will give us the flexibility to split up large tasks across multiple threads, each with a designated section of pixels to modify.



A toolbar object will be used to track which operation is currently selected and handle key presses accordingly. Recent changes will be saved on a stack, with the board state appended after a certain amount of time editing/number of changes. The undo feature will be implemented by popping elements off the stack and adding it to a second stack that stores recent changes. This will allow the user to return the canvas to a more recent state if need be.

We plan to complete our project with as few libraries as possible. Whilst we could utilize pre-written methods from a library such as Abstract Window Toolkit's Graphix, we feel that by implementing these features ourselves, our abilities will be better demonstrated. There are still library dependencies we must rely on of course. To display our user interface, we will be using Swing to comply with the project specification and we may end up relying on an image processing library if we meet our reach goal of exporting a PNG.

6 Design Principles

With the application offering multiple editing tools, dependency inversion will be in allowing easy extensibility. Namely, common behaviour between editing tools needs to be concrete to ensure that tools won't require mass changes for modifications to be made. At the same time higher level behaviours such as editing need to be easily extensible while also setting a standard for how new components should be made. Inheritance offers one way of doing this, but interfaces will also be required to define additional groups of specialised behaviour. Due to Java's single inheritance nature, inheritance will need to be reserved for sharing common attributes and behaviours.

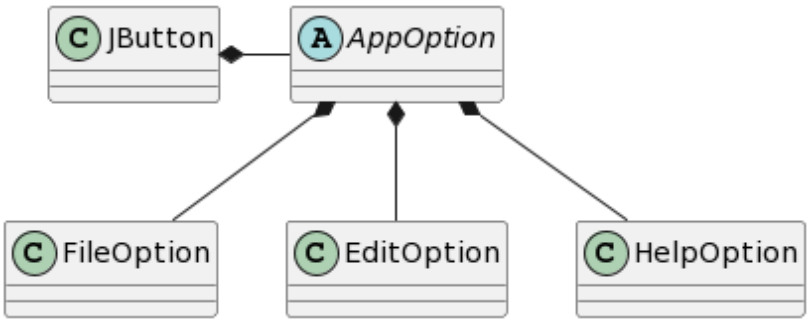


Figure 2: Button class structure, the AppOption class will implement the generic behaviour for interacting with the application and will in turn be inherited by more specific child options.

Interface segregation is also essential. The application will offer several possible ways for the user to interact with it. As such, it is important that distinct components do not depend on unnecessary methods and data. Ideally, a change to one component should force as little recompilation as possible. Categories such as UI, user options and editing tools should be segregated, with additional lower levels of abstraction for distinct component groups in each category.

Overall, code re-usability, extensibility and following DRY principles are essential to in ensuring the application is easy to maintain as it grows. This will be further emphasized by ensuring loose coupling between components. Abstract classes and interfaces should be used where appropriate to define generic behaviours that can further be extended upon through lower level components.

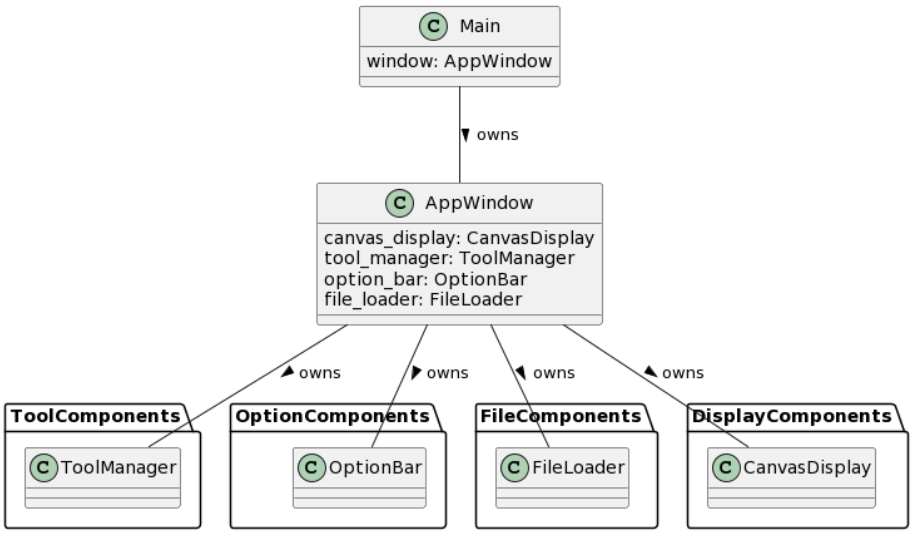


Figure 3: Component hierarchy, each distinct component will have it's own central class that'll be responsible for coordinating and encapsulating processes and data specific to it's component group. Data should not be shared between components unless necessary. Components will only communicate with each other through passing data to the central AppWindow.

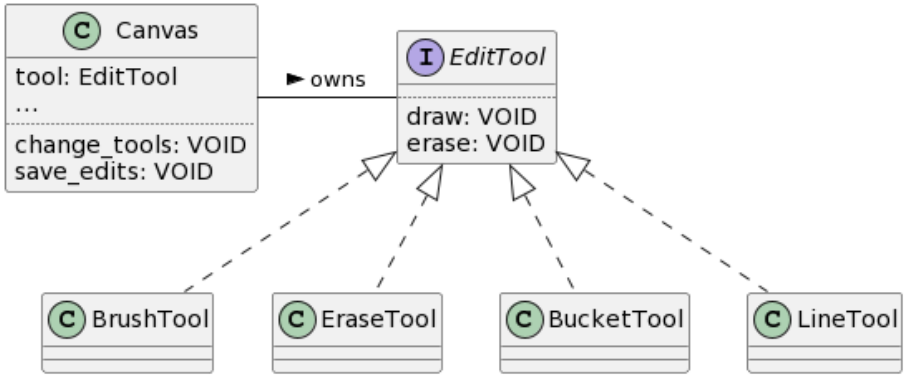


Figure 4: Tool diagram, the canvas will have a generic EditTool that specifies required behaviour to interact with the canvas. Class Polymorphism allows the interface to be substituted out by any child class that implements the required methods.

7 Project Plan

Since beginning this project, we have been in regular contact using discord. To begin, we drafted this document collaboratively using google docs. This allowed us to make concurrent changes to the document as well as communicating our thoughts on sections with comments. Once we had solidified our plan, we migrated to overleaf in order to typeset this report with LaTeX. To produce this application, we will loosely follow a scrum methodology. Each week will begin with a group call to discus the current state of the project and the tasks that need completing for that sprint. Each developer can choose one of these tasks and create their own branch. If anyone encounters difficulty, they can send a message to the team and anyone available can offer support to keep the project on track.

Date	**Milestone**
week 7-9	The application launches and is able to display a grid of pixels. A majority of UI elements display on the screen. The pen tool is functional.
week 10-12	The rest of the toolset is complete including brush, line, bucket fill, shape tools and eyedropper
week 13-15	Canvas operations have been built including change size, rotate, flip, undo and redo
week 16-17	Possible refactoring of code and general optimizations.

8 Risk Analysis

Feature	**Analysis**	**Risk**
1 Canvas	Drawing the canvas is an integral part of the program. This should be reasonably simple to implement successfully with basic elements of Java Swing.	low
2 File Menu	This is another key part of the application that the user requires to interact with other features of the program. This is where the user would be opening a new or existing file from.	low
3 New File	This is a simple enough feature that can be implemented by calling a function on the canvas	low

4 Save File	This requirement intentionally does not specify an output format. Implementing this using widely known formats such as PNG or JPG is a high risk goal as our group has little experience with image manipulation in java. If, after some research, we decide this is too optimistic, we can adopt another approach. Dumping the contents of the canvas as pixel colour values by virtue of data scraping would allow a user to save and load their work in the application.	high
7 Edit Menu	A menu-type graphical control element that handles the file. Options to undo, redo, copy and paste would be listed here in a dark font if the action is allowed and a lighter, grey font if the action is unavailable.	medium
8 Canvas Size	This feature depends on the quality of previous implementations of the navigation bar and the canvas feature. It will utilise this code and modify the original size of the array of pixels to display an altered size of the canvas.	high
9 Canvas Limit	This can be solved with a simple check in the resizing function.	low
10 Flip Canvas	This is a medium-risk feature as it will require efficient restructuring of the pixels on the canvas. The success of this feature is entirely dependent on the extensible of the basic paint program.	medium
12 Rotate Canvas	This is also medium-risk as it follows the same basic logic as the flip canvas feature, with some modifications.	medium
14 Help Menu	This feature will only depend on the navigation menu, and will be low-risk as it will be a very simplistic set of text and/or image prompts to guide the user through the application.	low
15 Pen Tool	This is core functionality of the program. as only one pixel is painted below the cursor at a time, this should be relatively trivial to implement.	low

16 Brush Tool	This is slightly more complex than the pen tool. A radius of pixels are painted beneath the cursor meaning a mechanism to find a circle of pixels must be implemented. Additionally, to provide a paintbrush effect, some randomness must be added around the edge of the stroke	medium
17 Line Tool	This feature requires some processing to calculate which pixels to colour between the start and end point. This shouldn't be too tricky to implement, however it must be done well as it may be accessed by the shape tools later on.	medium
18 Eyedropper Tool	This feature will allow for the possibility of greater user customisation and will interact with all drawing-related features in the program. It can be implemented simply with hex codes determined by the user, or it can be extended to a more user-friendly and interactive colour picker. This gives a level of flexibility to the risk level of this feature.	high
19 Eraser Tool	This feature entirely depends on the pen tool. In relying on its methods, this can be implemented cleanly, however this requires that the pen tool is built on schedule.	medium
20 Bucket Tool	This features may require complex computation to find the bounds to an area of pixels with the same colour.	medium
21 Shape Tool	This will allow the user to automatically place shapes(circles, squares etc) on the canvas without the need of the pen tool. Shapes with straight edges will be easier to implement than one with rounded edges.	medium

9 Acceptance Tests

Requirement	**Acceptance Test**	**Expected Result**
1 Displaying the canvas	Opening a new document	The application launches and displays a blank canvas
2 File menu	Clicking on the file option	A drop-down menu appears featuring several options
3 New file	Clicking the new file option	A box appears asking to confirm this choice, if the user does then the canvas wipes
4 Save and load	Clicking on the save option and entering a name and location to save	Opening this saved file yields the same image that was saved
5 Undo feature	Checking canvas state after several **undo** operations	A recent canvas state replaces the current canvas and the latest canvas state is saved on the redo stack
6 Redo feature	Checking canvas state after several **redo** operations	A recent canvas state replace the current canvas and the latest canvas state is saved state is on the redo stack
7 Edit menu	Clicking the edit icon in the navigation bar	A drop-down menu appears featuring options to edit the canvas
8 Resizing the canvas	Using the resize option to extend or shrink the canvas	The displayed canvas changes to the new size
9 Canvas size limit	Attempting to resize to larger than 1000x1000	An error box appears warning the user that this canvas is too large
10 Flipping the canvas	Clicking the flip option in the edit menu	The canvas flips across the desired access
12 Rotating the canvas	Clicking the rotate option in the edit menu	The canvas rotates in the desired direction and dimensions change accordingly
14 Help menu	Clicking on the help menu tab	A drop-down menu appears with several help options
15 Pen Tool	Clicking on the pen button and start drawing on the canvas	An accurate drawing without fuzzy edge should be made in the canvas from the user's input
16 Brush Tool	Clicking on the brush button and drawing on the canvas	An accurate drawing with fuzzy edge should be made on the canvas from the user's input
17 Line Tool	Clicking the line tool and drawing on the canvas	An accurate line should be drawn from the start point to the end point

18 Eyedropper	Clicking on the eyedropper tool and pressing a colour on the canvas	The correct colour used on the canvas should be selected for the user
19 Eraser Tool	Clicking on the eraser tool and going over a drawn line	The sections pressed with the eraser tool should be erased
20 Fill Tool	Clicking on the bucket tool and selecting a colour from the colour wheel, then clicking on the canvas.	The area enclosed by non-empty pixels on the canvas should change to the same colour as the colour selected using the bucket tool.
21 Shape Tool	Clicking on the shape tool and drawing on the canvas	The correct shape should be drawn to the correct size as chosen by the user

10 Costing

Task List	**Cost per hour**	**Estimated Hours**
Planning and Design phase (w4-6)	**£2179.00**	**72**
Team on-boarding	£30.00	10
Initial topic selection	£30.00	2
Feature selection and principles	£45.00	10
Requirements analysis	£20.00	5
UI design	£20.55	15
Architectural design	£20.55	15
Risk analysis	£15.50	5
Acceptance tests	£40.00	8
Schedule planning (tasks, activity gantt)	£50.00	2
Design report writing and editing	£15.50	10
Implementation phase (w7-15)	**£14,466.60**	**352**
Milestone 1 development (w7-9)	**£4,240.20**	**113**
Design (eg, class design)	£20.55	14
Development (coding)	£50.00	60
Documentation	£15.50	15
Testing	£50.00	12
Coordination	£10.00	12
Milestone 2 development (w9-12)	**£4,533.25**	**116**
Design	£20.55	15
Development	£50.00	65
Documentation	£15.50	10
Testing	£50.00	14
Coordination	£10.00	12

Milestone 3 development (w12-15)	**£4,745.25**	**123**
Design	£20.55	15
Development	£50.00	63
Documentation	£15.50	14
Testing	£50.00	19
Coordination	£10.00	12
User Evaluation phase (w16-17)	**£947.90**	**36**
Evaluation design	£20.55	8
Ethics approval process	£40.00	3
Participant recruitment	£15.50	10
Running evaluation	£50.00	8
Analysis	£15.50	7
Resulting changes	**£5,175.25**	**139**
Design	£20.55	15
Development	£50.00	70
Documentation	£15.50	14
Testing	£50.00	19
Coordination	£10.00	20

Non-staff costings	**P/P phase**	**M1 phase**	**M2 phase**	**M3 phase**	**User Eval phase**
Developer equipment	£7,000.00	£0.00	£0.00	£0.00	£8,000.00
Hosting costs	£0.00	£0.00	£0.00	£0.00	£0.00
Software subscriptions	£0.00	£0.00	£0.00	£0.00	£0.00
User eval participant costs	£0.00	£0.00	£0.00	£0.00	£2,000.00

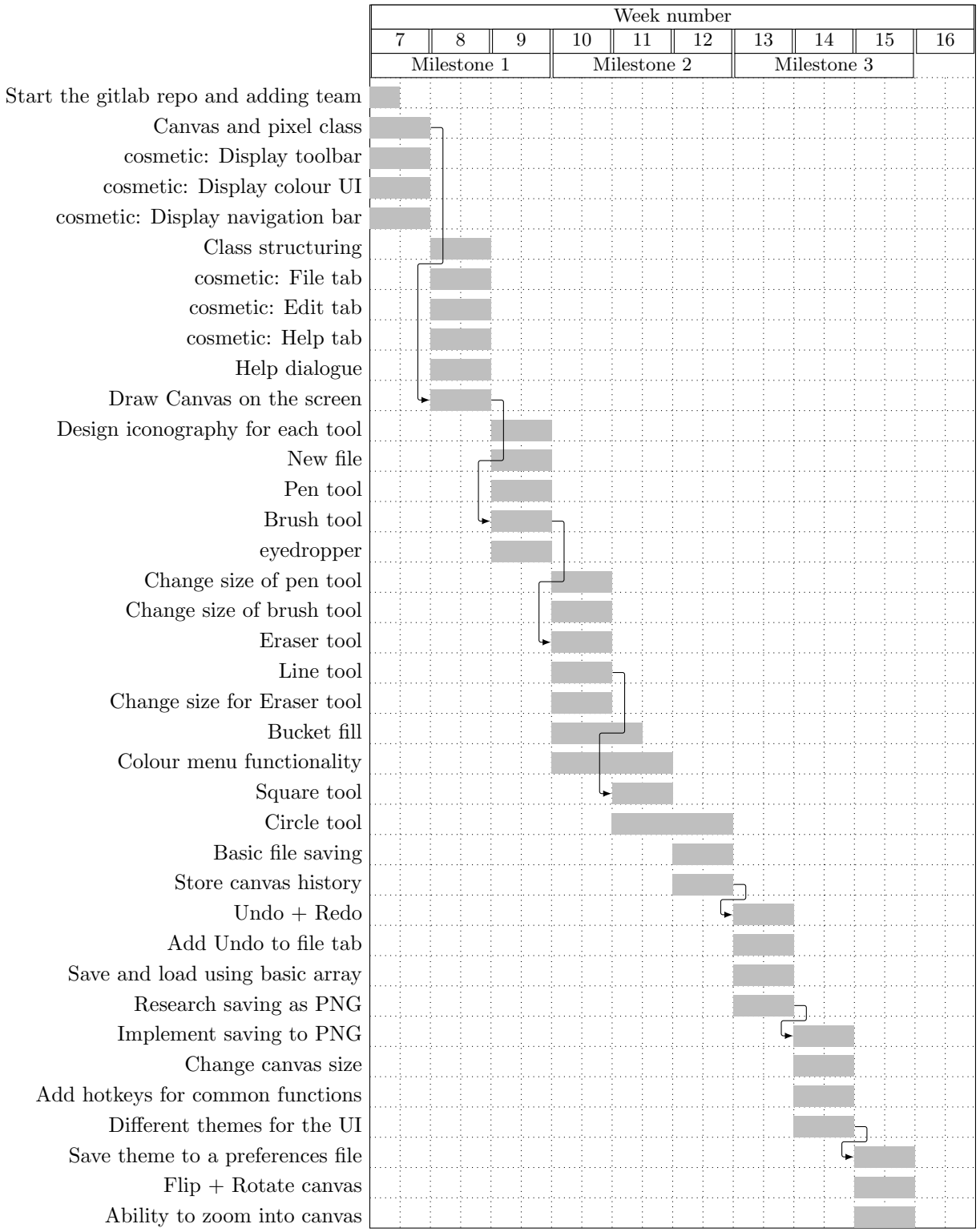
Total person-hours effort:	599 hours
Total staff costs:	£21,820.85
Total non-staff costs:	£17,000.00
Total project cost estimate:	**£38,820.85**

11 Task list

TASK	INFORMATION
Week 7	
Create gilab repo	Start the gitlab repo and adding team to project
Canvas and pixel class	UI of Canvas and Pixel
cosmetic: Display toolbar	UI of Toolbar
cosmetic: Display colour UI	UI of colour box
cosmetic: Display navigation bar	UI of navigation bar
Week 8	
Class structuring	UI skeleton created
cosmetic: File tab	Adding "File..." to navigation bar
cosmetic: Edit tab	Adding "Edit..." to navigation bar
cosmetic: Help tab	Adding "Help..." to navigation bar
Help dialogue	Listing information of creators and support number
Draw Canvas on the screen	UI display the canvas - Blank page
Week 9	
Design iconography for each tool	UI add image.png of each tool
New file	Allow user to create new canvas
Pen tool	Allow user click to the pen tool and draw on canvas
Brush tool	Allow user click to the brush tool and draw on canvas
eyedropper	Allow user pick a color
Week 10	
Change size of pen tool	Allow user change size of pen tool
Change size of brush tool	Allow user change size of brush tool
Eraser tool	Allow user delete which pen tool and brush tool made
Colour menu functionality	Make the colour board working
Bucket fill	Allow user fill the colors which zoned by pen tool
Line tool	Allow user create draw straight lines
Week 11	
Change size for Eraser tool	Allow user change size of Eraser tool
Square tool	Allow user quick draw a Square
Circle tool	Allow user quick draw a Circle
Week 12	
Basic file saving functionality	Allow user save and load the canvas withing the application
Store canvas history	Allow user see what has been change

Week 13	
Undo + Redo	Allow user undo and redo which saved in history
add undo to file tab	Allow user undo with the file tab
Save and load using basic array	Allow user save and load canvas
research saving as PNG	Start research to save as PNG
Week 14	
implement saving to PNG	Allow user save canvas as PNG
Change canvas size	Allow user change the canvas size
add hotkeys for common functions	Allow user get access to the tool and function from keyboard
different themes for the UI	Adding more themes to the UI
Week 15	
flip + rotate canvas	Allow user flip or rotate canvas
ability to zoom into canvas	Allow user zoom in and out
save theme to a preferences file	Allow user change the theme and never have to change again

12 Activity Network


Week number
7
8
9
10
11
12
13
14
15
16
Milestone 1
Milestone 2
Milestone 3
Start the gitlab repo and adding team
Canvas and pixel class
cosmetic: Display toolbar
cosmetic: Display colour UI
cosmetic: Display navigation bar
Class structuring
cosmetic: File tab
cosmetic: Edit tab
cosmetic: Help tab
Help dialogue
Draw Canvas on the screen
Design iconography for each tool
New file
Pen tool
Brush tool
eyedropper
Change size of pen tool
Change size of brush tool
Eraser tool
Line tool
Change size for Eraser tool
Bucket fill
Colour menu functionality
Square tool
Circle tool
Basic file saving
Store canvas history
Undo + Redo
Add Undo to file tab
Save and load using basic array
Research saving as PNG
Implement saving to PNG
Change canvas size
Add hotkeys for common functions
Different themes for the UI
Save theme to a preferences file
Flip + Rotate canvas
Ability to zoom into canvas

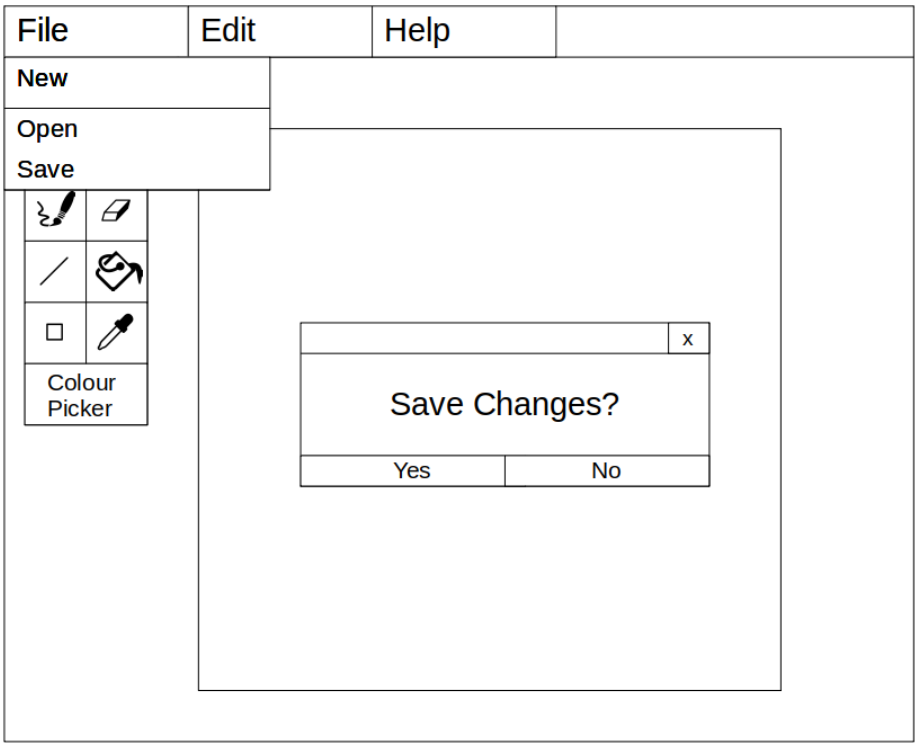


Figure 5: Save menu

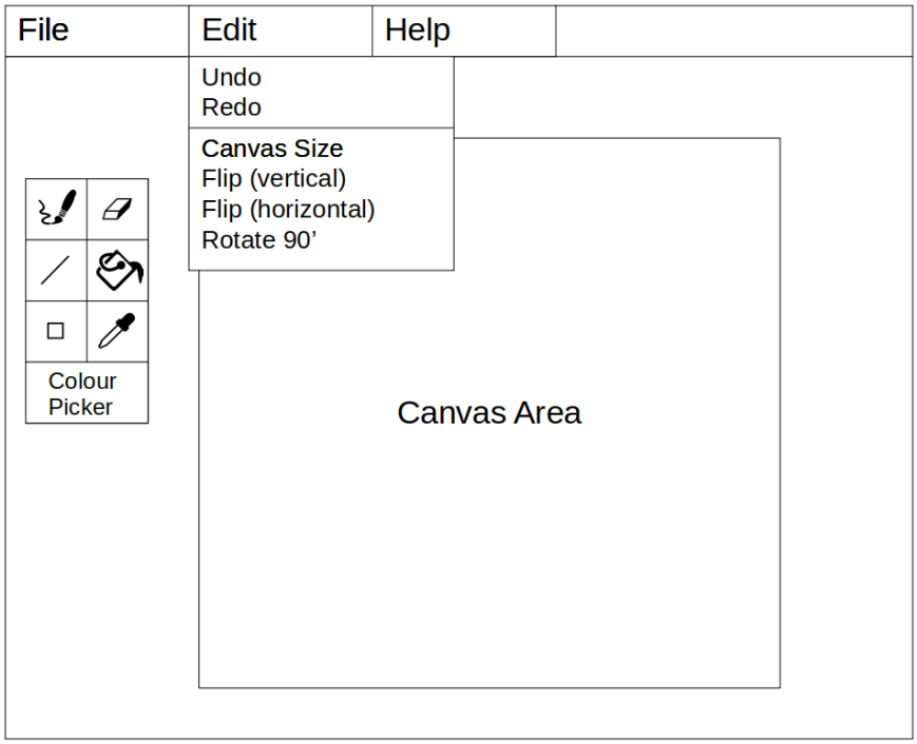


Figure 6: Edit menu