Documented Design

The project has required combining skills and techniques which have required deeper thought than simple line by line programming.

Since I have used HTML Canvas for this project. HTML Canvas works similar to a real canvas, anything I paint onto the canvas, will stay until I decide I want to start again by painting over what I have already done. This leads to a system which is very forgetful, since after the canvas has been cleared, it doesn’t really remember what happened before it.

In programming terms, this means that I have to use variables which stay the same even after one frame has passed. This means that global variables, while frowned upon for their heavy memory usage and poor programming practice, are very useful to me, as they won’t reset after a function finishes.

Furthermore, events which occur on multiple frames, for instance the dragging of an object. Will require me to check for the right conditions on each frame (i.e. mouse down on 3 frames). Since I can’t just wait for 3 frames to pass and wait on that line of code, I have to let the rest of the processing for that frame complete and check again next frame if the conditions are met (Fig 1). This requires more thought than just using an event based pre-built GUI in another language (Fig 2).

On event Object.Drag

Start Drag

Figure 2 Pseudocode of a drag function which could be programmed with a GUI framework like VCL Forms.

Global drag counter

If mouse is down

Increment drag counter by 1

If drag counter is 3

Start drag

Figure 1 Pseudocode of a drag function in canvas, showcasing how the solution must work on multiple frames

Mapping out features on a flow chart gives me a good idea of how to implement them, for instance, dragging, I have very specific requirements for this feature which has lots of different states the user can be in, dragging an object, not dragging an object but selected, just released an object from a drag etc. Visualising problems like this gives me a better idea of the steps and variables needed before I waste time and computing power on a poor solution.

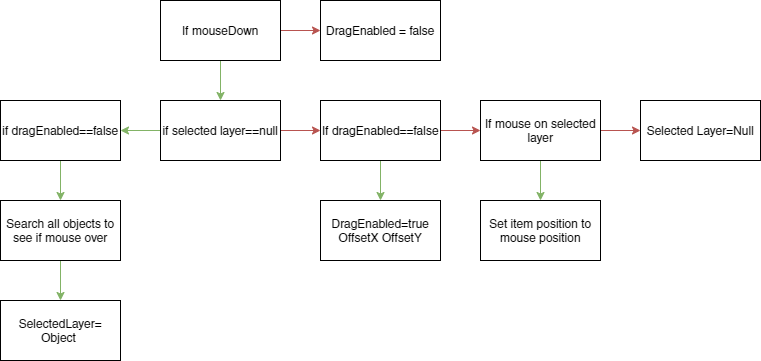


Figure 3 Full flow chart of the dragging function. Green arrows signify true and red arrows signify false.

I iterated over this flowchart multiple times to avoid unnecessary steps since some parts were repeated, furthermore, I decided not to use the “official” shapes of a flowchart as it would save me time to just use squares with coloured arrows.

Since most of my project relied on the manipulation of shapes I decided to figure out if there were any similarities to rectangles which I would support and Images which I would support. From this I discovered that they were indeed similar, in fact drawing an image was almost the same process since it had a location, height and width, the only difference was that one contained an image object.

From this I discovered that using classes to display rectangles and images would be beneficial as it would provide structure and allow me to organise my code into structured areas where classes could inherit from a main layer class, since all layers have some base properties and I decided to name an instance of this class a Layer, fitting with similar systems like Adobe Photoshop, which uses a similar system to keep track of its parts (Figure 4).

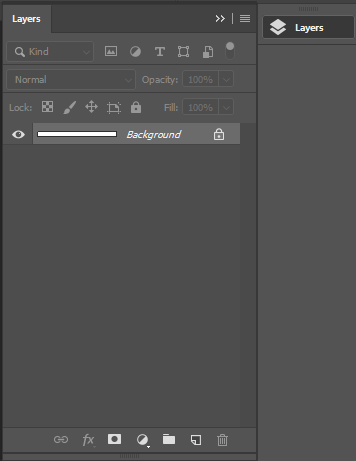


Figure 4 Screenshot from Adobe Photoshop, showcasing the layers system

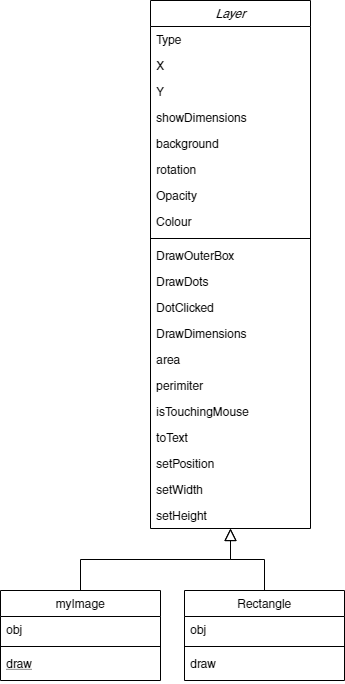


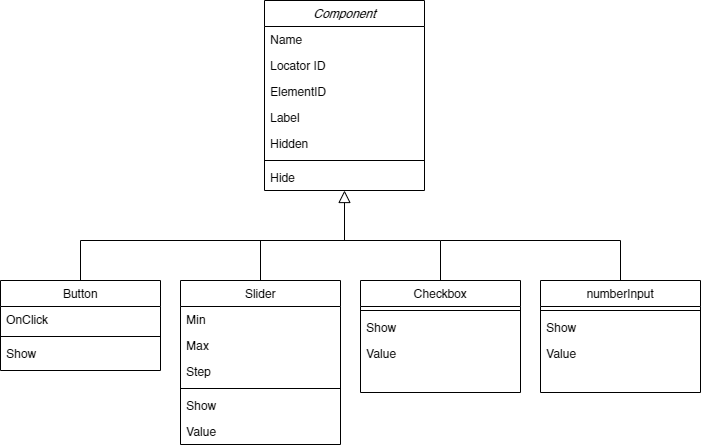
Figure 5 Class diagram for my Layer class

Using a class diagram has helped me to plan for what things I need in my classes and what is useful to inherit. I have one parent class which holds all of the base information (i.e. location) that a layer should have and allow for child classes to take these and have their own draw classes. In theory this approach should allow for other developers to easily add their own type of layer in the future and utilise the methods I have created.

Furthermore, making this a class allows me to split the code up into manageable packages which can be tested individually to check for bugs and errors, this should help me speed up development time significantly as I wont have to sift through code to find errors, this also decreases the amount of lines coded, as reusing classes stops me from retyping code, this is important, especially in JavaScript as the less the browser has to process the higher fps it can run at, improving the user experience and progressing to my goal of usability.

My project also uses classes to show the buttons on the side, this allowed me to have more control on my buttons rather than just interacting directly with the DOM.

Figure 6 User interface Class diagram



**Algorithms**

Since I am using the canvas, I have to operate over multiple frames. To detect if a rectangle is being dragged, (Which is an essential to the success of my system), I can store how many consecutive frames the mouse has been over the object and the mouse has been down. If this number is 2 then I know that the user intends to drag the object and can initiate a drag by setting the appropriate variable to save this state. As the mouse is being held down rather than just a click. However, this technique requires the use of a global variable since I must store the number of frames outside the detection function itself which wipes its variables after it returns or finishes (fig 7).

Global frameQuantity

Function Update

If mouse is down and over rectangle

frameQuantity = frameQuantity + 1

Else

frameQuantity = 0

if frameQuantity >= 2

rectangle Location = mouse Location

Figure 7: simplified pseudocode of the drag detection algorithm

However the story only gets more complicated from here, since I want to be able to just click the rectangle to access the size editing system or edit properties of the rectangle, I have to account for this and stop detecting clicks when editing size, and show the appropriate sizing features when the rectangle is clicked.

Furthermore, I cannot use a system as literal as the one in Figure 9, since I have lots of times when I would trigger the system by accident. If I were already dragging another rectangle I do not want to pick up this rectangle and start dragging too if I mouse over it as this would look strange and mean that you would have to drag rectangles carefully around other rectangles to avoid picking them up. From this I can learn that I will need lots of different global variables to describe the state which the interface is in, it should know when an object is being dragged and not try pick up others, which I will do by storing if an object is being dragged in a variable, since you can only drag the layer which is selected. I should also know which layer is selected, since I will want to show resizing features only for this layer and not for other layers which aren’t selected. Another consideration is if I want the rectangle to snap to the location of the mouse. Currently this system would do that as I haven’t allowed for the offset of a mouse being in the middle of a rectangle (Figure 8).

This can be solved with another variable which stores the offset of the mouse from the rectangle location and reapplies it when I am dragging the rectangle, which maintains the position of the mouse with the rectangle. However, this is another variable which will have to store data past one frame, meaning I have to use a global variable.

As you can see, a feature which seems so easy, branches out into many problems which must be solved to make the system function properly.

Figure 8: Visualisation of rectangle snapping to mouse

Frame 1, before mouse goes down

Frame 2, as mouse goes down, selected layer set

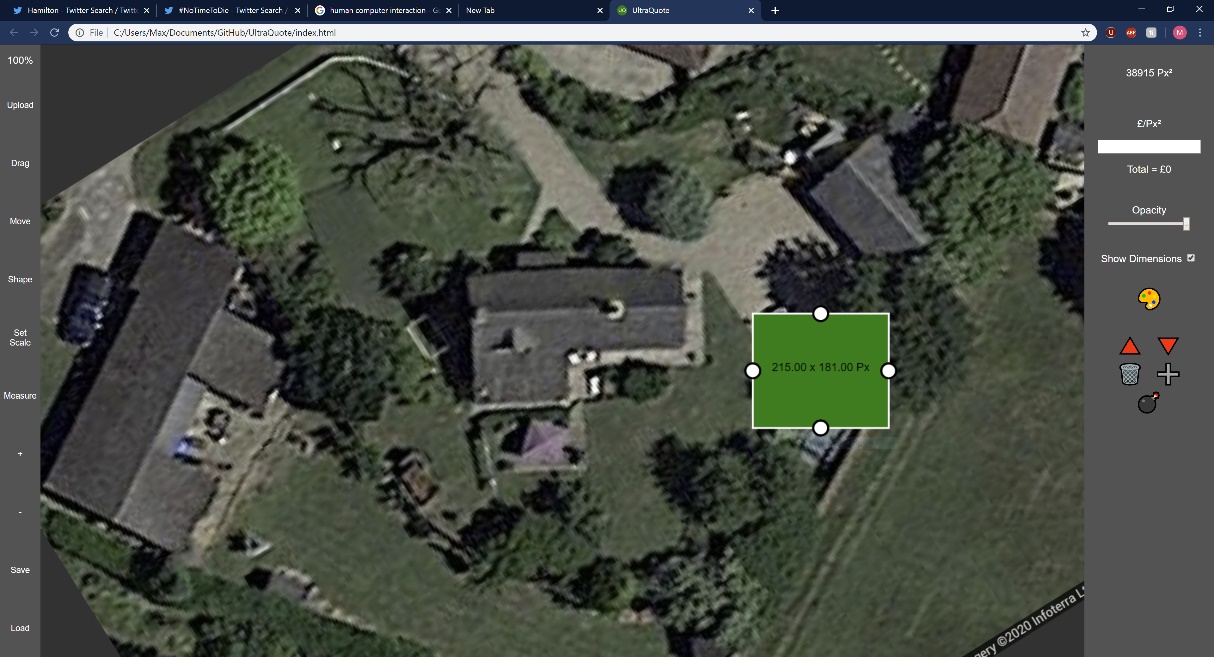
Frame 3, drag detected, rectangle snaps to location of mouse

This problem is properly visualised in Figure 3, which shows what has to be checked each frame to discover what to do on the screen.

**Data Structures**

Array of type Layer – used to store layers, this datatype is useful as it allows me to access layers and keeps the order of them. Since I need to be able to delete any layer, possibly in the middle or the end, a queue or a stack is inapplicable.

**Human Computer Interaction**



1

4

3

2

6

7

5

The idea for my Human Computer Interaction was to have as much accessible from on the screen at once, while only showing the user the things that are necessary to avoid overcrowding the screen space, for this reason I decided to have a section on the side of the screen which only popped up when a layer was clicked and disappeared when nothing was selected, this also goes towards meeting my 9th objective of a “usable” program as a feature like this directly supports usability by guiding them to the correct options.

1. Rectangle - allows the user to drag and drop it around the screen, this means they do not have to use arrow keys or type in a location manually and can instead use a mouse which is intuitive and fits closely to the previous solution allowing for a smooth transition from the previous solution to my new solution. Plus, a user can choose to just select the layer by clicking which brings up the layer specific buttons (3).
2. Price input box – Takes the value, which is input and reactively adjusts the price above, very simple solution, instead of having an on-screen prompt/alert which would have taken time to load and possibly confused the user. This solution gives the user room for error too since the box only allows numbers to be input.
3. Layer specific buttons – Buttons which only show when they are applicable, rectangles show colour, up layer, down layer, delete and duplicate, these buttons disappear when the layer is unselected.
4. Resize dots – allow a user to visually drag these dots with the mouse to resize a selected layer, only show when a layer is selected to avoid screen clutter, only show for the selected layer to avoid user confusion.
5. Background layer – Image which doesn’t allow dragging and does allow rotation, this prevents users from accidentally clicking the background and misaligning their whole project, increases usability.
6. Always shown buttons – Buttons which must always be shown to the user, for instance the importing of an image, saving or loading their work, having these buttons always show gives the user a stable area which doesn’t change, gives user a sense of direction in the UI.
7. Zoom Indicator – Automatically adapts to show the current zoom, this gives the user a figure to stop them from zooming in/out too far to find their work in the canvas.