<InfPALS/>

Big Project 2

MS Paint

Let us know you took part!

Even if you are not here on the day but use the materials, we collect this data to plan future projects.

Project Overview

- Browser based version of MS Paint
- We will be writing the core functionality, other tools and features are left as an open extension.

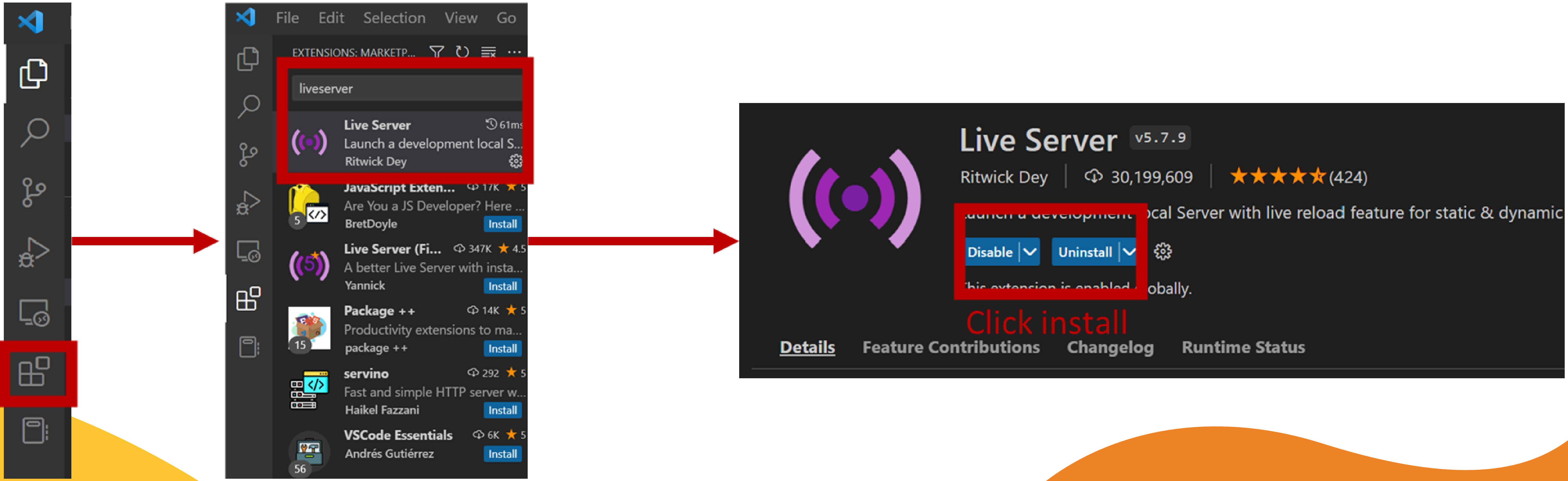
Draws a free-form line one pixel wide.

You will need

Code editor (e.g. VS Code)

- https://code.visualstudio.com/download

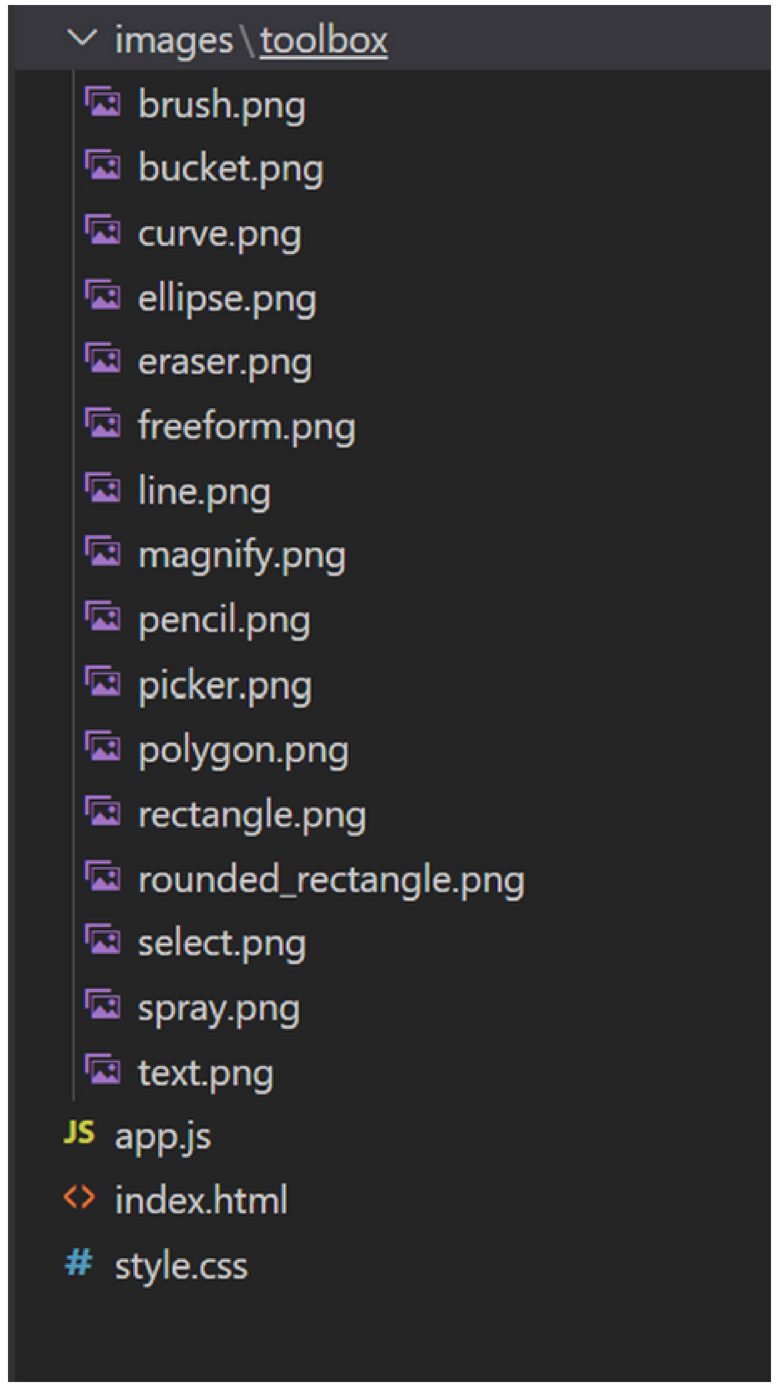
If using VS Code, we recommend the **Live Server** extension.



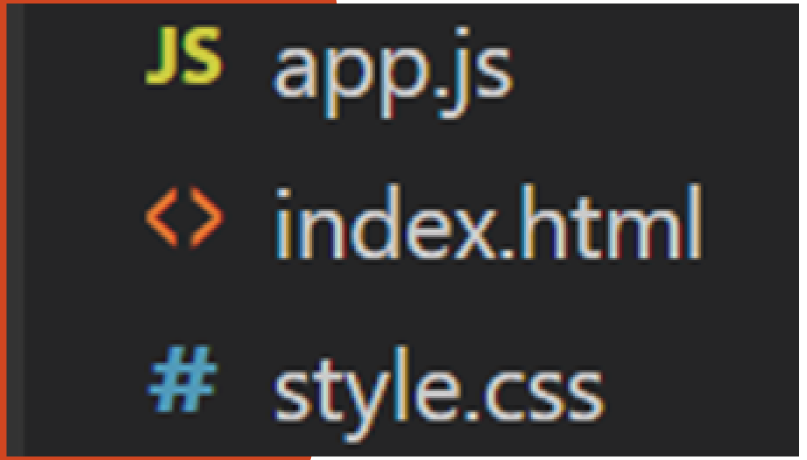
Starter files

We have provided a barebones structure and icons for the project, feel free to change these later to make this your own.

https://github.com/infpals/ip2023-big-project-2



```
images\toolbox
  brush.png
  bucket.png
  curve.png
  ellipse.png
  eraser.png
  freeform.png
  line.png
  magnify.png
  pencil.png
  picker.png
  polygon.png
  rectangle.png
  rounded_rectangle.png
  select.png
  spray.png
  text.png
app.js
index.html
style.css
```



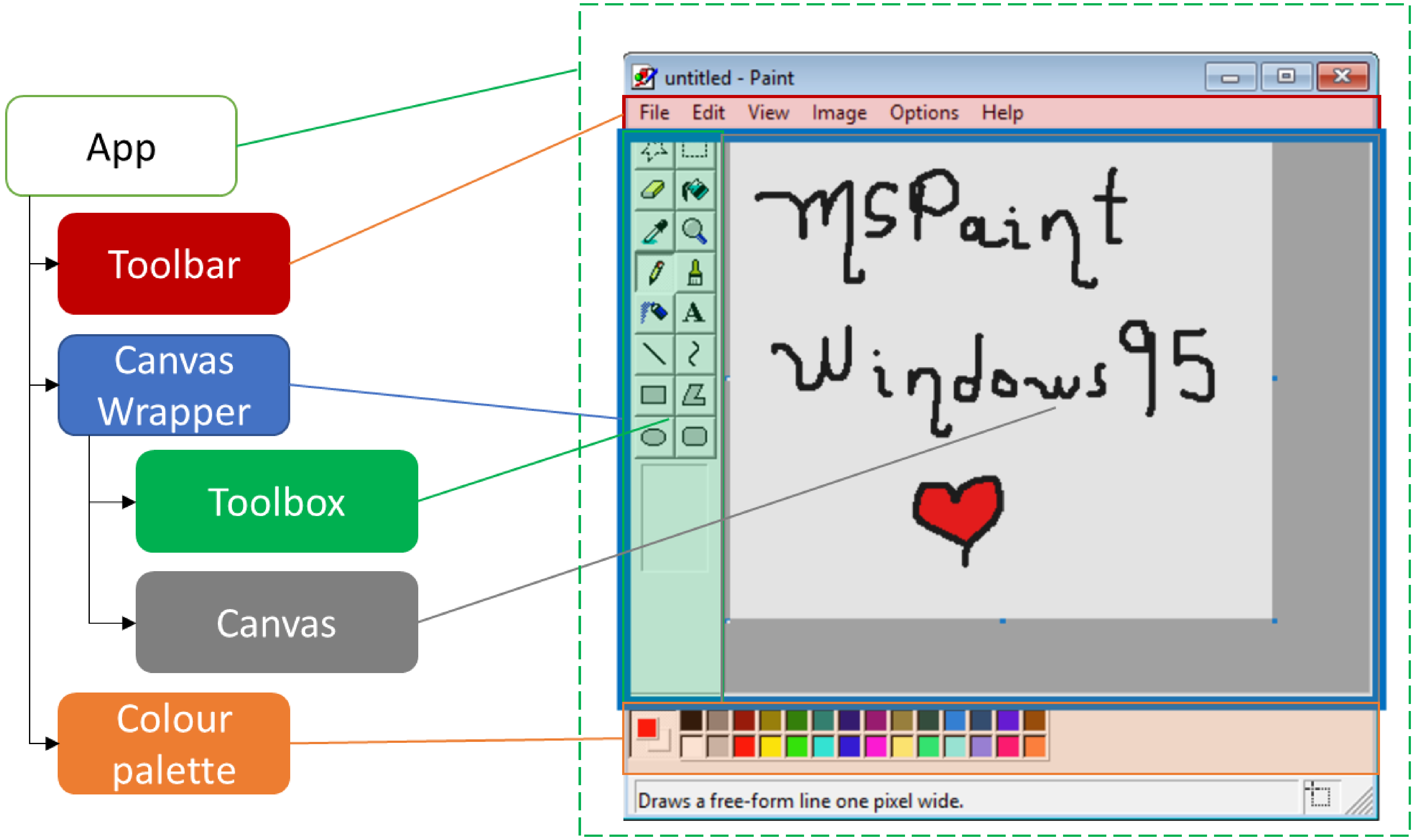
```
app.js
index.html
style.css
```

JavaScript, how the app behaves.

HTML, how the app is structured.

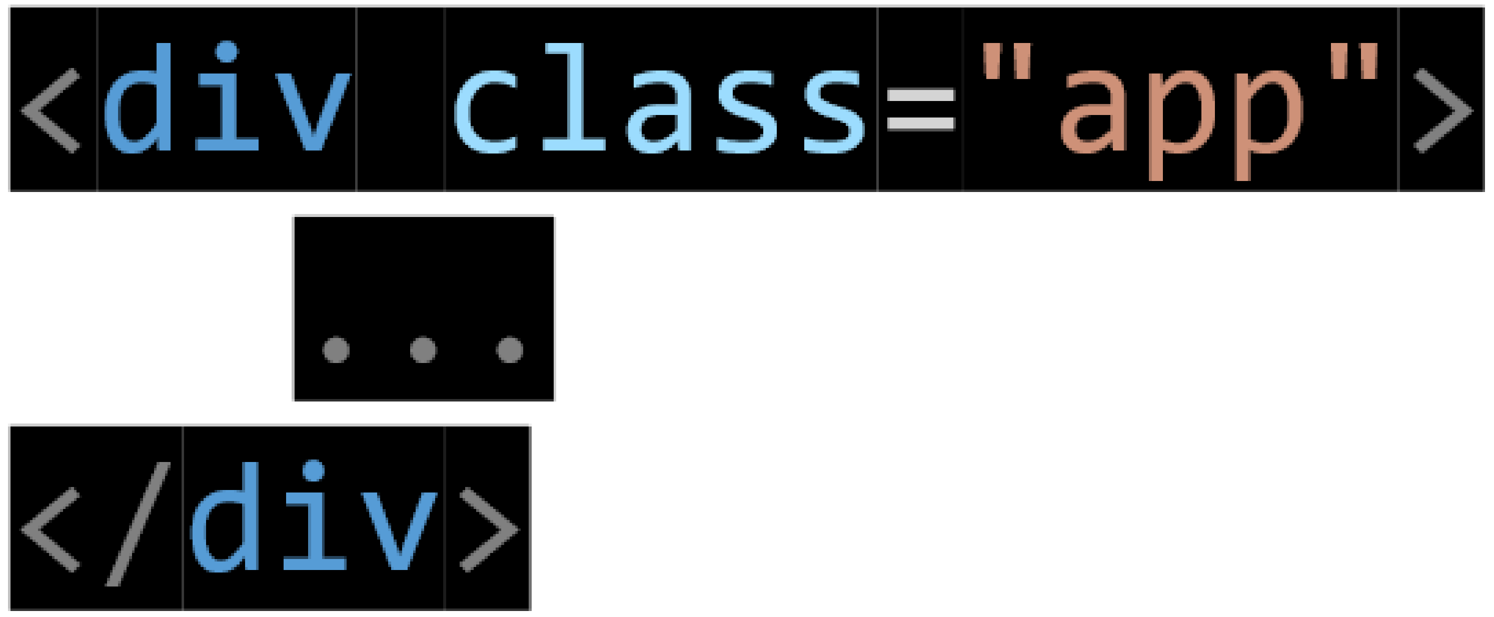
CSS, how the app looks (fonts, colours, margins, etc.).

User Interface

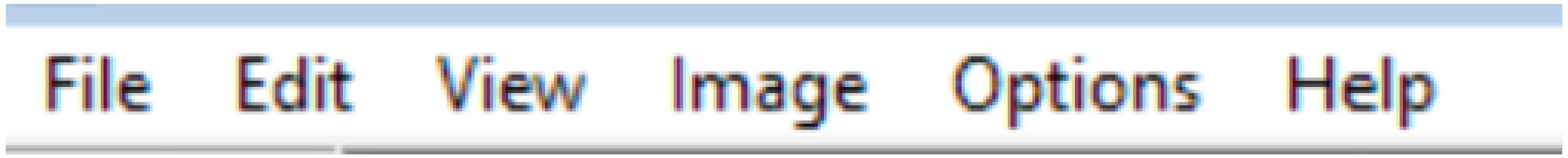


Create the layout

- Using the hierarchy on the previous slide, create <div> tags for each of the containers to form the structure of the app.
- Give each div tag a class of the relevant name, for example, the top-most tag would be:



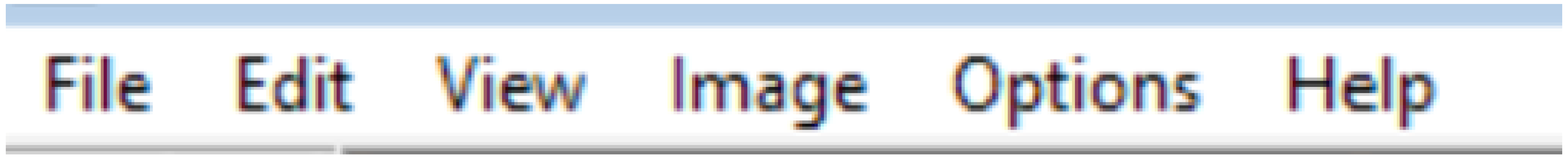
```
<div class="app">
    ...
</div>
```



Menu bar

- Inside of the toolbar, add ‘buttons’ for each of the toolbar dropdowns
  - Each button will be created with a link tag (<a>) inside the toolbar div
  - Use an ID for each button, this will be accessed later.
  - e.g. <a id="file">File</a>

Note that most of these buttons will be purely decorative for our project, but it is a good habit to make them functional if you wanted to extend the functionality.



Styling

- You should now have a menu bar, but it looks wrong.
- Open up style.css and add three headings:

```
.toolbar {
}
```

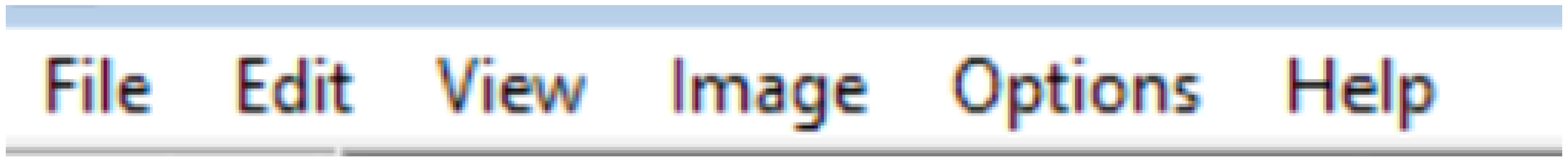
This changes the style of the whole toolbar

```
.toolbar a {
}
```

This changes the style of the links within the toolbar, i.e. the menu buttons.

```
.toolbar a:hover {
}
```

This changes the style of the menu buttons upon hovering.

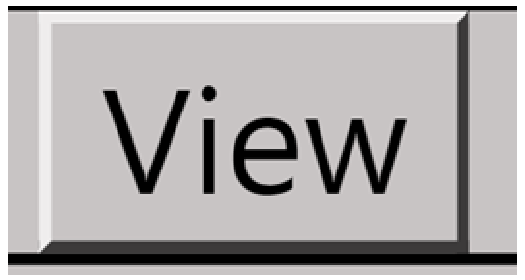


Styling

```
.toolbar {
}
.toolbar a {
}
.toolbar a:hover {
}
```

The toolbar requires a background colour (**#c8c4c4**) and a **black 1px border**.
You can also add 2px of top and bottom padding to fill out the toolbar.
The buttons require a transparent 1px border, 1px of padding on top and bottom, and 4px of left and right padding.

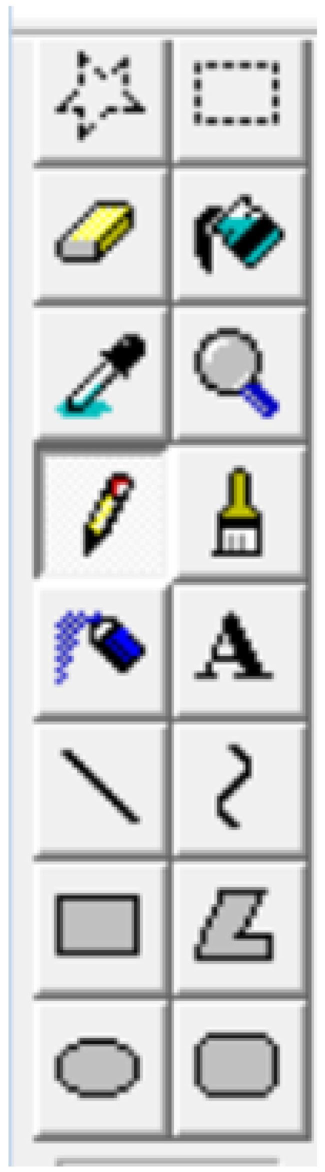
Upon hovering you want to create a 3D effect: light grey border on top and left and grey on bottom and right. As a hint, this is the bottom border:
border-bottom: 1px solid rgba(0, 0, 0, 0.7);



Toolbox

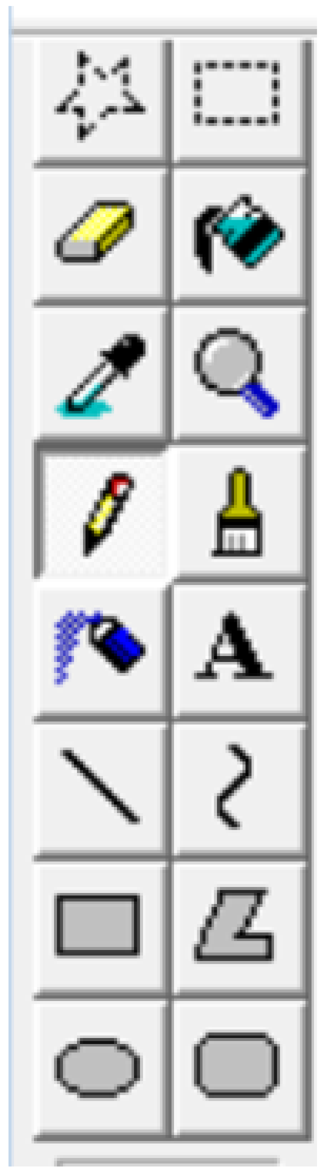
- To add tools to your toolbox, you can create more ‘button’ links with the <a> tag
  - Inside each link you can add an image <img> (instead of text)
  - Here is a example:
    - <a id="freeform"><img src="images/toolbox/freeform.png"></a>

Finish this for all the tools!



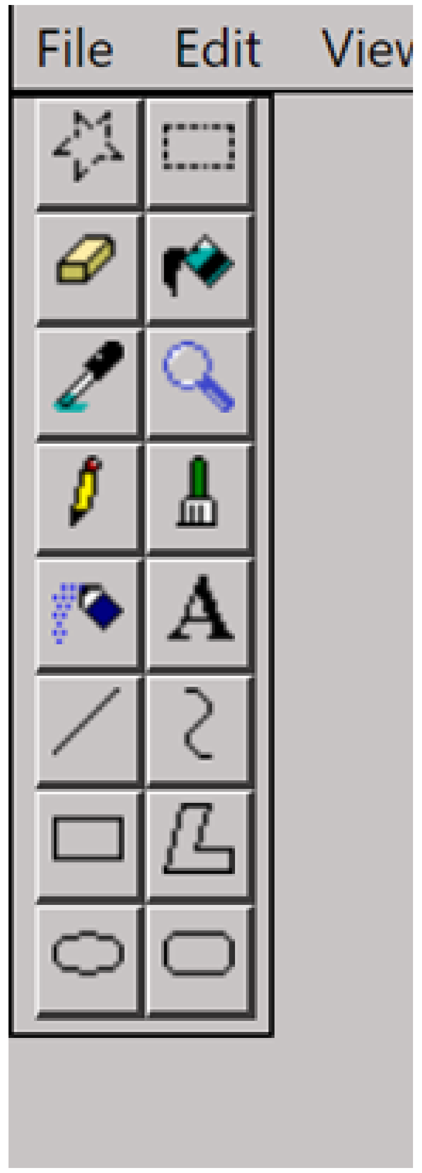
Styling

- As before we are going to style the outer frame and the buttons, this is done by adding **.toolbox** and **.toolbox a** to the stylesheet.
- The toolbox will require a **1px black border**, a **width of 50px**, a **height of 'max-content'**, a **padding of 5px on the left**, and **3px on the right and bottom**.
- In order to arrange the buttons you can use the '**display**' attribute and set it to **grid**, specifying the format. In this context, 1fr 1fr means two equally fractional columns.
    - **display: grid;**
    - **grid-template-columns: 1fr 1fr;**



Styling

- Now the buttons are arranged, they need to be styled.
- Each button has **2px padding**, text-align is set to **centre** (to centre the image icon), and requires the same 3D border from the toolbar.
- Once this is styled you should have a toolbox and toolbar.
    - *If you are stuck, ask for a leader to come help.*

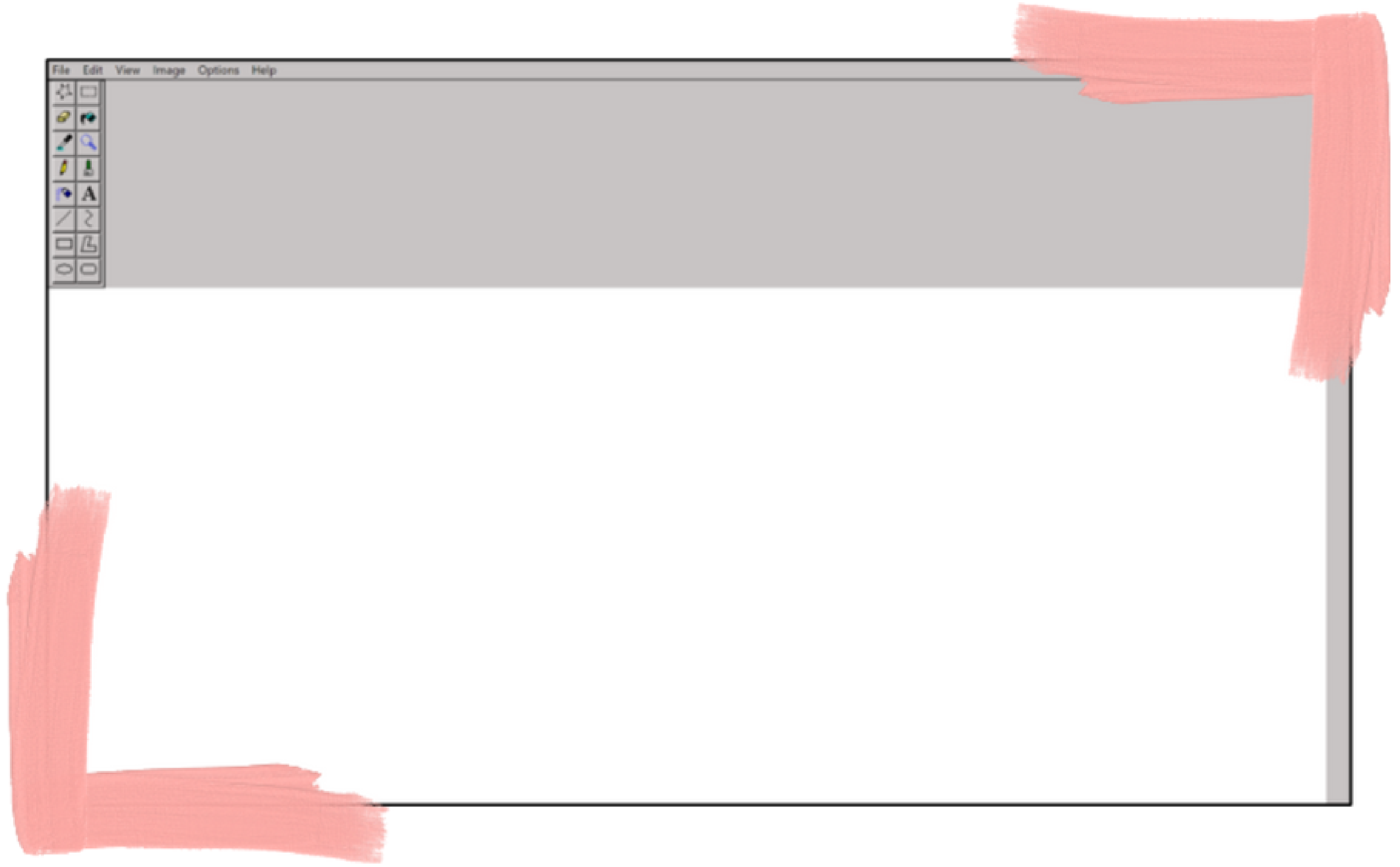


Canvas

- We need a canvas to draw onto, add this into the div using the <canvas> tag.
- In the stylesheet, give the canvas a width and a height, as well as a background colour.

Flex

- You may notice something looks wrong…
- By default divs spread the whole width as a 'block', we need to change this so they 'flex'.
- Create a heading in the stylesheet for the **canvas-wrapper** and set the **display** to '**flex**'.



You should now have a UI!

If you don't, ask for help

Select tool

- To add the ability to change tool, add an onclick event to the <a> buttons in the toolbox, for example:
    - <a id="eraser" onclick="selectTool('eraser')"><imgsrc="images/toolbox/eraser.png"></a>
- Now you need to create a function called selectTool in the app.js file
    - This can be done with: **function selectTool(toolName) {}**
- Within the function, you need to loop through all the tools in the toolbox and if the class name equals “selected”, set to blank (“”).
- Then, get the tool with id equal to the passed string (toolName), and set the class name to “selected”.
- The following functions might be useful:
    - https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName
    - https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById
    - https://developer.mozilla.org/en-US/docs/Web/API/Element/className

Styling

- Although the function now selects the tool, you need to add styling to reflect that.
- Create a heading of .toolbox .selected, it should have the following properties:
  - 2px dark top and left border
  - 1px light right and bottom border
  - background-image: radial-gradient(rgba(0, 0, 0, 0.7) 1px, transparent 0);
    - This creates a spotted dark background.
  - background-size: 3px 3px;
    - This changes the size of the spots.

Brush tool

- The brush tool is going to function as drawing lots of lines between the previous and current point when the mouse is down.
- First we need to declare variables, at the top of the file write:

```
let canvas;
let ctx;
let stroke = 15;
let currentTool;
let colour = 'black';
let isMouseDown = false;
let drawnPoints = [];
```

- Now we need a way of getting the current tool, create a function to do this.

Canvas Setup

- When the website loads, we want to trigger some functions and attach some event listeners, this is done within a function:
  - **window.onload = function () {}**
- Inside the function:
  - Update the canvas variable by getting the canvas by ID.
  - Update the context (ctx) variable by setting it to canvas.getContext(‘2d’)
  - This allows us to interact with the canvas in a 2D way.
  - Set the canvas width and height using the clientWidth/clientHeight properties of canvas.
    - This is so the canvas bounds are scaled.
  - Add three event listeners to the canvas, one for mousedown, movemove, and mouseup.
    - If unfamiliar with event listeners, see https://developer.mozilla.org/en-US/docs/Web/API/Element/mousedown_event

Mouse Position

- Before drawing the brush, we need to calculate where the mouse is relative to the canvas.
- Create a function **getMousePos(e)** that takes an event (such as a mouse click) and returns the position of the event within the canvas.
  - To get the x and y position of the event, you can use e.clientX/e.clientY
  - To get the canvas bounds, canvas.getBoundingClientRect(), from which you can then use bounds.top and bounds.left to get the relevant points.
  - Use these two sets of points to subtract the border from the x and y to return

```
return {
        x: // Your x calc here,
        y: // Your y calc here,
    };
```

Mousedown

- On mousedown the system needs to:
    - Get the current mouse position.
    - Set isMouseDown to true.
    - Get the current tool.
    - Then if the current tool is a brush, the line can be started by:

      ctx.moveTo(currentPosition.x, currentPosition.y);
      ctx.beginPath();
      ctx.lineWidth = stroke;
      ctx.lineCap = 'round';
      ctx.strokeStyle = colour;

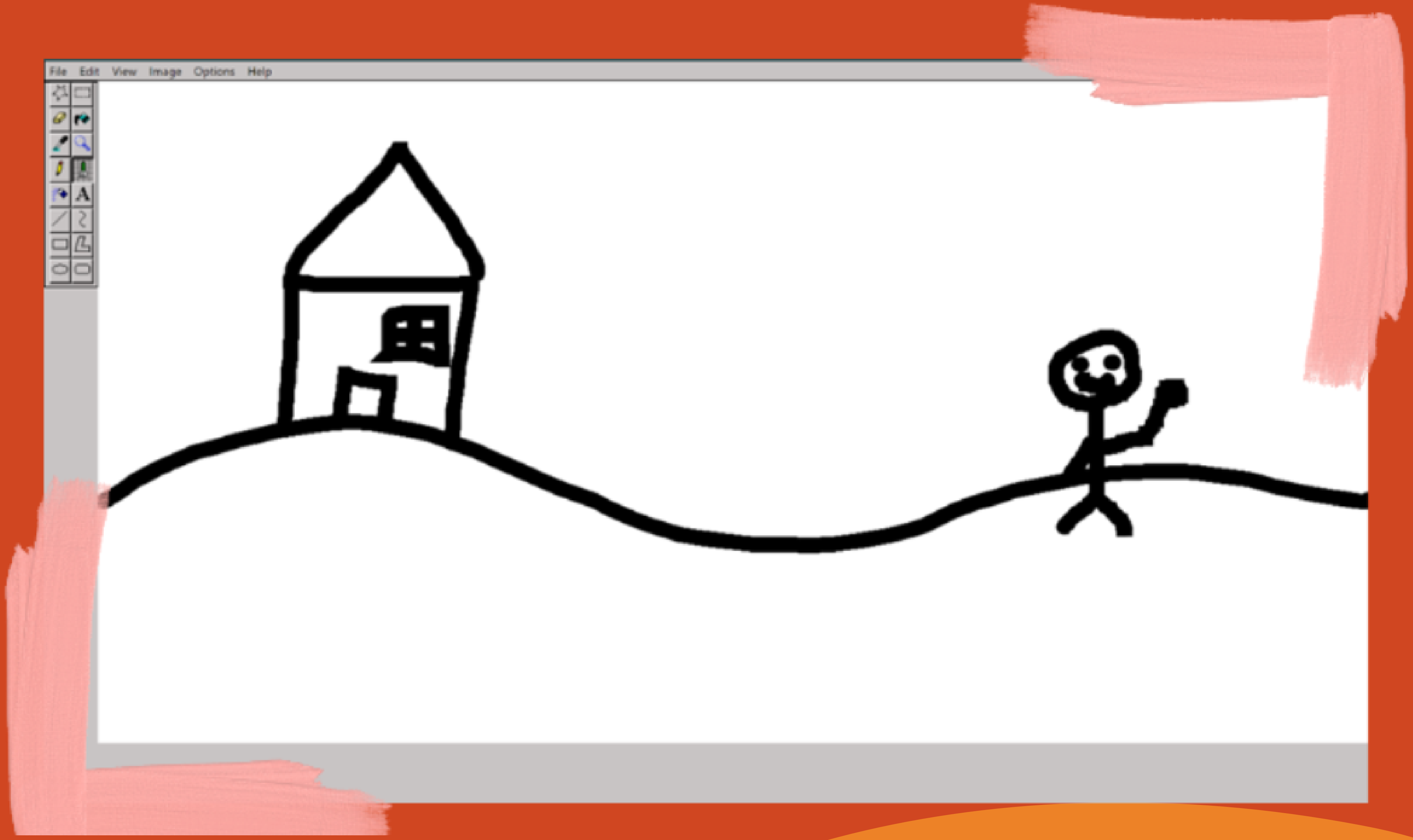
Mousemove & Mouseup

- In the mousemove function the system will need to:
  - Get the current position
  - If the mouse is down and the current tool is a brush, you will need to draw a line to the new current position:

    ctx.lineTo(currentPosition.x, currentPosition.y);
    ctx.stroke();
- On mouse up, the only thing the system needs to do is set the variable mousedown to false.

You should now have a draw tool!

If you don't, ask for help

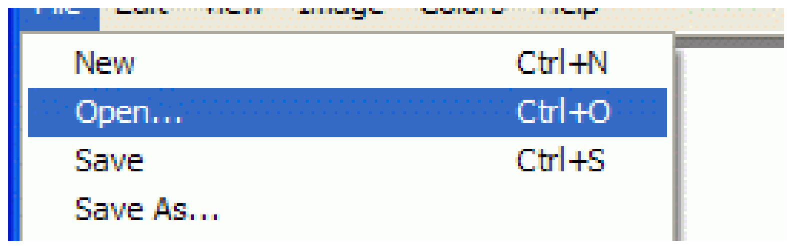


Your choice!

There are loads more tools you can build, so pick one you are interested in and try to create a solution. Here are some ideas to get you started:

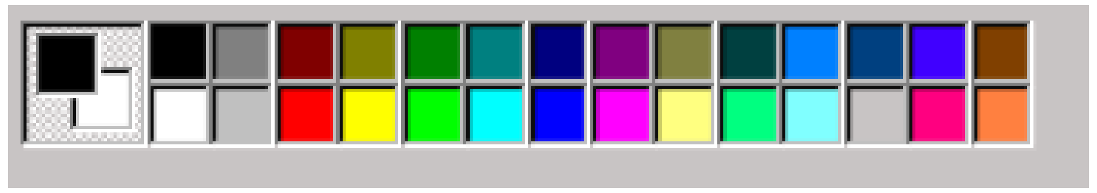
Eraser

MS Paint implements this as another brush that uses the background colour.



Save/Open

You can save and load images to the canvas by making a tool on the menu bar.



Colours & Size

Use your knowledge on how to build toolboxes to build a colour palette or size tool.

Thank you for coming!

We hope you enjoyed the workshop