Selected files

14 printable files

```
colourPalette.js
editshapeTool.js
eraserTool.js
freehandTool.js
helperFunctions.js
index.html
lineToTool.js
mirrorDrawTool.js
shapeTool.js
sketch.js
sprayCanTool.js
stampTool.js
style.css
toolbox.js
```

colourPalette.js

```javascript
//Displays and handles the colour palette.
function ColourPalette() {
    //a list of web colour strings
    this.colours = ["black", "silver", "gray", "white", "maroon", "red", "purple",
        "orange", "pink", "fuchsia", "green", "lime", "olive", "yellow", "navy",
        "blue", "teal", "aqua"
    ];
    //make the start colour be black
    this.selectedColour = "black";

    var self = this;

    var colourClick = function() {
        //remove the old border
        var current = select("#" + self.selectedColour + "Swatch");
        current.style("border", "0");

        //get the new colour from the id of the clicked element
        var c = this.id().split("Swatch")[0];

        //set the selected colour and fill and stroke
        self.selectedColour = c;
        fill(c);
        stroke(c);

        //add a new border to the selected colour
        this.style("border", "2px solid blue");
    }

    //load in the colours
    this.loadColours = function() {
        //set the fill and stroke properties to be black at the start of the programme
        //running
        fill(this.colours[0]);
        stroke(this.colours[0]);

        //for each colour create a new div in the html for the colourSwatches
        for (var i = 0; i < this.colours.length; i++) {
            var colourID = this.colours[i] + "Swatch";

            //using JQuery add the swatch to the palette and set its background colour
            //to be the colour value.
            var colourSwatch = createDiv()
            colourSwatch.class('colourSwatches');
            colourSwatch.id(colourID);

            select(".colourPalette").child(colourSwatch);
            select("#" + colourID).style("background-color", this.colours[i]);
            colourSwatch.mouseClicked(colourClick)
        }

        select(".colourSwatches").style("border", "2px solid blue");
    };
    //call the loadColours function now it is declared
    this.loadColours();
}
```

editshapeTool.js

```
// I took this feature from lectures, which I adapted to fit into drawing app. Work in progress
function editShapeTool(){
    var editButton;
    var finishButton;
    var editMode = false;
    var currentShape = [];
    this.icon =  "/assets/editshape.png";
    this.name = "editshape";
    this.populateOptions = function(){
       editButton = createButton("Edit Shape");
       finishButton = createButton("Finish Shape");
       console.log(c);
       select(".options").html("<div class='tool-info'>You can draw shapes and press finish button to allow initiate de edition mode</div><br>");
       select(".options").child(editButton);
       select(".options").child(finishButton);

    this.settingTool();
    this.mousePressOnCanvas = function(canvas) {
       if (mouseX > canvas.elt.offsetLeft &&
          mouseX < (canvas.elt.offsetLeft + canvas.width) &&
          mouseY > canvas.elt.offsetTop &&
          mouseY < (canvas.elt.offsetTop + canvas.height)
       ) {
          return true;
       }
       return false;
    }

    }
    this.draw = function(){
       if(mouseIsPressed){
          currentShape.push({
             x: mouseX,
             y: mouseY
          });
          beginShape();
          for (var i=0; i<currentShape.length; i++){
             vertex(currentShape[i].x, currentShape[i].y);
          }
          endShape();
       }

    }
    this.settingTool = function(){
       noFill();
       loadPixels();
       finishButton.mousePressed(function(){
          loadPixels();
          currentShape = []; // emptying the array
       })
    }
    this.unselectTool = function() {
         //clear options
         select(".options").html("");
     };

}
```

eraserTool.js

```
function eraserTool(){
    this.icon = "assets/eraser.jpg";
    this.name = "eraser";
    var eraserSizeSlider;
    // next function was entirely wrote by me
    this.populateOptions = function() {
        select(".options").html("<div class= 'description'>Eraser Tool, you can choose the size below:</div><br>");
        SliderValue = createDiv();
        select(".options").child(SliderValue);
        eraserSizeSlider = createSlider(10,50,10,10);
        select(".options").child(eraserSizeSlider);
        updateEraserSizeDisplay();
        eraserSizeSlider.input(updateEraserSizeDisplay);
    }
    // next function was entirely wrote by me
    this.draw = function (){

        if(mouseIsPressed)
        {
            eraserSize = eraserSizeSlider.value();
            stroke(255);
            fill(255);
            rect(mouseX,mouseY,eraserSize,eraserSize);
        }
    }

    this.unselectTool = function() {
        //clear options
        select(".options").html("");
    };
    // next function was entirely wrote by me
    function updateEraserSizeDisplay() {
        SliderValue.html("Eraser Size: " + eraserSizeSlider.value());
    }
}
```

freehandTool.js

```
function FreehandTool(){
    //set an icon and a name for the object
    this.icon = "assets/freehand.jpg";
    this.name = "freehand";
    var strokeline;
    var strokevalue;

    //to smoothly draw we'll draw a line from the previous mouse location
    //to the current mouse location. The following values store
    //the locations from the last frame. They are -1 to start with because
    //we haven't started drawing yet.
    var previousMouseX = -1;
    var previousMouseY = -1;

    this.populateOptions = function() {
        select(".options").html("<div class= 'description'>A freehand Tool. Choose the stroke below:</div><br>");
        strokevalue = createDiv();
        select(".options").child(strokevalue);
        strokeline = createSlider(5,30,5,5);
        select(".options").child(strokeline);
        updateStrokeSizeDisplay();
        strokeline.input(updateStrokeSizeDisplay);


    this.draw = function(){
        //if the mouse is pressed
        if(mouseIsPressed){
            //check if they previousX and Y are -1. set them to the current
            //mouse X and Y if they are.
            if (previousMouseX == -1){
                previousMouseX = mouseX;
                previousMouseY = mouseY;
            }
            //if we already have values for previousX and Y we can draw a line from
            //there to the current mouse location
            else{
                // strokeline is a feature added by me
                var s = strokeline.value()
                strokeWeight(s);
                line(previousMouseX, previousMouseY, mouseX, mouseY);
                previousMouseX = mouseX;
                previousMouseY = mouseY;
            }
        }
        //if the user has released the mouse we want to set the previousMouse values
        //back to -1.
        //try and comment out these lines and see what happens!
        else{
            previousMouseX = -1;
            previousMouseY = -1;
        }
    };
    this.unselectTool = function() {
        //clear options
        select(".options").html("");

    };
    // next function developed by me
    function updateStrokeSizeDisplay() {
        strokevalue.html("Stroke: " + strokeline.value());
}
}
}
```

helperFunctions.js

```
function HelperFunctions() {

    //Jquery click events. Notice that there is no this. at the
    //start we don't need to do that here because the event will
    //be added to the button and doesn't 'belong' to the object

    //event handler for the clear button event. Clears the screen
    select("#clearButton").mouseClicked(function() {
        background(255, 255, 255);
        //call loadPixels to update the drawing state
        //this is needed for the mirror tool
        loadPixels();
    });

    //event handler for the save image button. saves the canvas to the
    //local file system.
    select("#saveImageButton").mouseClicked(function() {
        saveCanvas("myPicture", "jpg");
    });
}
```

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <script src="lib/p5.min.js"></script>
    <script src="lib/p5.dom.js"></script>

    <script src="sketch.js"></script>

    <!-- add extra scripts below -->
    <script src="toolbox.js"></script>
    <script src="colourPalette.js"></script>
    <script src="helperFunctions.js"></script>
    <script src="freehandTool.js"></script>
    <script src="lineToTool.js"></script>
    <script src="sprayCanTool.js"></script>
    <script src="mirrorDrawTool.js"></script>
    <script src="eraserTool.js"></script>
    <script src="shapeTool.js"></script>
    <script src="stampTool.js"></script>
    <script src="editShapeTool.js"></script>

    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <div class="wrapper">
      <div class="box header">My Drawing App

        <button id="clearButton">Clear</button>
        <button id="saveImageButton">Save Image</button>
      </div>
      <div class="box" id="sidebar"></div>
      <div id="content"></div>
      <div class="box colourPalette"></div>
      <div class="box options"></div>
    </div>
  </body>
</html>

```

lineToTool.js

```javascript
//a tool for drawing straight lines to the screen. Allows the user to preview
//the a line to the current mouse position before drawing the line to the
//pixel array.
function LineToTool(){
    this.icon = "assets/lineTo.jpg";
    this.name = "LineTo";

    var startMouseX = -1;
    var startMouseY = -1;
    var drawing = false;

    //draws the line to the screen
    this.draw = function(){

        //only draw when mouse is clicked
        if(mouseIsPressed){
            //if it's the start of drawing a new line
            if(startMouseX == -1){
                startMouseX = mouseX;
                startMouseY = mouseY;
                drawing = true;
                //save the current pixel Array
                loadPixels();
            }

            else{
                //update the screen with the saved pixels to hide any previous
                //line between mouse pressed and released
                updatePixels();
                //draw the line
                line(startMouseX, startMouseY, mouseX, mouseY);
            }

        }

        else if(drawing){
            //save the pixels with the most recent line and reset the
            //drawing bool and start locations
            loadPixels();
            drawing = false;
            startMouseX = -1;
            startMouseY = -1;
        }
    };


}

```

mirrorDrawTool.js

```
function mirrorDrawTool() {
    this.name = "mirrorDraw";
    this.icon = "assets/mirrorDraw.jpg";

    //which axis is being mirrored (x or y) x is default
    this.axis = "x";
    //line of symmetry is halfway across the screen
    this.lineOfSymmetry = width / 2;

    //this changes in the jquery click handler. So storing it as
    //a variable self now means we can still access it in the handler
    var self = this;

    //where was the mouse on the last time draw was called.
    //set it to -1 to begin with
    var previousMouseX = -1;
    var previousMouseY = -1;

    //mouse coordinates for the other side of the Line of symmetry.
    var previousOppositeMouseX = -1;
    var previousOppositeMouseY = -1;

    this.draw = function() {
        //display the last save state of pixels
        updatePixels();

        //do the drawing if the mouse is pressed
        if (mouseIsPressed) {
            //if the previous values are -1 set them to the current mouse location
            //and mirrored positions
            if (previousMouseX == -1) {
                previousMouseX = mouseX;
                previousMouseY = mouseY;
                previousOppositeMouseX = this.calculateOpposite(mouseX, "x");
                previousOppositeMouseY = this.calculateOpposite(mouseY, "y");
            }

            //if there are values in the previous locations
            //draw a line between them and the current positions
            else {
                line(previousMouseX, previousMouseY, mouseX, mouseY);
                previousMouseX = mouseX;
                previousMouseY = mouseY;

                //these are for the mirrored drawing the other side of the
                //line of symmetry
                var oX = this.calculateOpposite(mouseX, "x");
                var oY = this.calculateOpposite(mouseY, "y");
                line(previousOppositeMouseX, previousOppositeMouseY, oX, oY);
                previousOppositeMouseX = oX;
                previousOppositeMouseY = oY;
            }
        }
        //if the mouse isn't pressed reset the previous values to -1
        else {
            previousMouseX = -1;
            previousMouseY = -1;

            previousOppositeMouseX = -1;
            previousOppositeMouseY = -1;
        }

        //after the drawing is done save the pixel state. We don't want the
        //line of symmetry to be part of our drawing

        loadPixels();

        //push the drawing state so that we can set the stroke weight and colour
        push();
        strokeWeight(3);
        stroke("red");
        //draw the line of symmetry
        if (this.axis == "x") {
            line(width / 2, 0, width / 2, height);
        } else {
            line(0, height / 2, width, height / 2);
        }
        //return to the original stroke
        pop();

    };

    /*calculate an opposite coordinate the other side of the
     *symmetry line.
     *@param n number: location for either x or y coordinate
     *@param a [x,y]: the axis of the coordinate (y or y)
     *@return number: the opposite coordinate
     */
    this.calculateOpposite = function(n, a) {
        //if the axis isn't the one being mirrored return the same
        //value
        if (a != this.axis) {
            return n;
        }

        //if n is less than the line of symmetry return a coorindate
        //that is far greater than the line of symmetry by the distance from
        //n to that line.
        if (n < this.lineOfSymmetry) {
            return this.lineOfSymmetry + (this.lineOfSymmetry - n);
        }

        //otherwise a coordinate that is smaller than the line of symmetry
        //by the distance between it and n.
        else {
            return this.lineOfSymmetry - (n - this.lineOfSymmetry);
        }
    };


    //when the tool is deselected update the pixels to just show the drawing and
    //hide the line of symmetry. Also clear options
    this.unselectTool = function() {
        updatePixels();
        //clear options
        select(".options").html("");
    };

    //adds a button and click handler to the options area. When clicked
    //toggle the line of symmetry between horizonatl to vertical
    this.populateOptions = function() {
        select(".options").html(
            "<button id='directionButton'>Make Horizontal</button>");
        //  //click handler
        select("#directionButton").mouseClicked(function() {
            var button = select("#" + this.elt.id);
            if (self.axis == "x") {
                self.axis = "y";
                self.lineOfSymmetry = height / 2;
                button.html('Make Vertical');
            } else {
                self.axis = "x";
                self.lineOfSymmetry = width / 2;
                button.html('Make Horizontal');
            }
        });
    };
}
```

shapeTool.js

```
function ShapeTool() {
    this.icon = "assets/shape.png";
    this.name = "shapeTool";
    var startMouseX = -1;
    var startMouseY = -1;
    var drawing = false;
    var selectedShape = null;
    var fillShape = true;

    // All inside populateOptions function was wrote by me
    this.populateOptions = function() {
        select(".options").html("<div class= 'description'>Select your favourite shape and push toggle button if you want fill or stroke shape </div><br>");
        // Creating DOM buttons
        var circleButton = createButton("");
        circleButton.id("button circle");
        let iconCircle = createImg('/assets/circle.png', 'Image Icon');
        iconCircle.size(45, 45); // Set the size of the image
        // Add the button to the DOM
        select(".options").child(circleButton);
        circleButton.child(iconCircle);

        var rectButton = createButton("");
        rectButton.id("button rectangle");

        let iconRectangle = createImg('/assets/rect.png', 'Image Icon');
        iconRectangle.size(45, 45); // Set the size of the image
        // Add the button to the DOM
        select(".options").child(rectButton);
        rectButton.child(iconRectangle);

        var triangleButton = createButton('');
        triangleButton.id("button triangle");
        let iconTriangle = createImg('/assets/triangle.png');
        iconTriangle.size(45,45);
        select(".options").child(triangleButton);
        triangleButton.child(iconTriangle);
        // Fill toggle

        var toggleButton = createButton('Fill/Stroke');
        select(".options").child(toggleButton);

        // Event handlers
        circleButton.mousePressed(function() { selectedShape = 'circle'; });
        rectButton.mousePressed(function() { selectedShape = 'rectangle'; });
        triangleButton.mousePressed(function() { selectedShape = 'triangle'; });
        toggleButton.mousePressed(function() {
            fillShape = !fillShape;
            if (fillShape) {
                fill(colourP.selectedColour);
            } else {
                noFill();
            }
            stroke(colourP.selectedColour);
        });
    };

    this.draw = function() {
        // If statements was an idea I took from freehandTool
        if (mouseIsPressed) {
            if (startMouseX == -1) {
                startMouseX = mouseX;
                startMouseY = mouseY;
                drawing = true;
                loadPixels();
            } else {
                updatePixels();
                // next switch was entirely wrote by me
                switch (selectedShape) {
                    case 'circle':
                        var radius = dist(startMouseX, startMouseY, mouseX, mouseY);
                        ellipse(startMouseX, startMouseY, radius * 2);
                        break;
                    case 'rectangle':
                        rect(startMouseX, startMouseY, mouseX - startMouseX, mouseY -
startMouseY);
                        break;
                    case 'triangle':
                        triangle(startMouseX, startMouseY, mouseX, mouseY, mouseX,
startMouseY);
                        break;
                }
            }
        } else if (drawing) {
            loadPixels();
            drawing = false;
            startMouseX = -1;
            startMouseY = -1;
        }
    };

    // I took this function from given tools
    this.unselectTool = function() {
        select(".options").html("");
    };
}
```

sketch.js

```
//global variables that will store the toolbox colour palette
//amnd the helper functions
var toolbox = null;
var colourP = null;
var helpers = null;


function setup() {

    //create a canvas to fill the content div from index.html
    canvasContainer = select('#content');
    var c = createCanvas(canvasContainer.size().width, canvasContainer.size().height);
    c.parent("content");

    //create helper functions and the colour palette
    helpers = new HelperFunctions();
    colourP = new ColourPalette();

    //create a toolbox for storing the tools
    toolbox = new Toolbox();

    //add the tools to the toolbox.
    toolbox.addTool(new FreehandTool());
    toolbox.addTool(new LineToTool());
    toolbox.addTool(new SprayCanTool());
    toolbox.addTool(new mirrorDrawTool());
    toolbox.addTool(new eraserTool()); //Tool added by me
    toolbox.addTool(new ShapeTool()); //Tool added by me
    toolbox.addTool(new stampTool()); //Tool added by me
    toolbox.addTool(new editShapeTool()); //Tool added by me
    background(255);

}

function draw() {
    //call the draw function from the selected tool.
    //hasOwnProperty is a javascript function that tests
    //if an object contains a particular method or property
    //if there isn't a draw method the app will alert the user
    if (toolbox.selectedTool.hasOwnProperty("draw")) {
        toolbox.selectedTool.draw();
    } else {
        alert("it doesn't look like your tool has a draw method!");
    }
}
```

sprayCanTool.js

```
function SprayCanTool(){

    this.name = "sprayCanTool";
    this.icon = "assets/sprayCan.jpg";
    var strokeline;
    var strokevalue;
    var points = 13;
    var spread = 10;

    this.draw = function(){
        var s = strokeline.value()
        if(mouseIsPressed){
            for(var i = 0; i < points; i++){
                point(random(mouseX-spread, mouseX + s), random(mouseY-spread, mouseY+s));
            }
        }
    };
    this.populateOptions = function() {
        select(".options").html("<div class= 'description'>A spray can Tool. Choose the stroke below:</div><br>");
        strokevalue = createDiv();
        select(".options").child(strokevalue);
        strokeline = createSlider(5,30,5,5);
        select(".options").child(strokeline);
        updateStrokeSizeDisplay();
        strokeline.input(updateStrokeSizeDisplay);
    }
        // next function developed by me
    function updateStrokeSizeDisplay() {
            strokevalue.html("Stroke: " + strokeline.value());
    }
        // I took this function from given tools
    this.unselectTool = function() {
            select(".options").html("");
    };
}

```

stampTool.js

```
function stampTool(){
    this.icon = "/assets/stamp.png";
    this.name = "stampTool";
    var selectedShape = null;
    var birdStamp = loadImage('/assets/bike.png');
    var starStamp = loadImage('/assets/star-stamp.png');
    var horseStamp = loadImage('/assets/horse.png');
    var stampSlider;
    var stampSize;
    // All inside populateOptions function was wrote by me
    this.populateOptions = function() {
        select(".options").html("<div class= 'description'>Select your favourite stamp and the size below:</div><br>");
        var starButton = createButton("");
        starButton.id("button star");
        let iconStar = createImg('/assets/star-stamp.png', 'Image Icon');
        iconStar.size(45, 45); // Set the size of the image
        // Add the button to the DOM
        select(".options").child(starButton);
        starButton.child(iconStar);
        var horseButton = createButton("");
        horseButton.id("button horse");
        let iconHorse = createImg('/assets/horse.png', 'Image Icon');
        iconHorse.size(45, 45); // Set the size of the image
        // Add the button to the DOM
        select(".options").child(horseButton);
        horseButton.child(iconHorse);
        var birdButton = createButton('');
        birdButton.id("button bird");
        let iconBird = createImg('/assets/bike.png', 'Image Icon');
        iconBird.size(45,45);
        select(".options").child(birdButton);
        birdButton.child(iconBird);
        stampSlider = createSlider(10,70,30);
        stampSlider.id("stampSize");
        select(".options").child(stampSlider);
        starButton.mousePressed(function() { selectedShape = 'star'; });
        horseButton.mousePressed(function() { selectedShape = 'horse'; });
        birdButton.mousePressed(function() { selectedShape = 'bird'; });

    };
    // next function was entirely wrote by me
    this.draw = function(){
        if(mouseIsPressed){
            stampSize = stampSlider.value();
            switch(selectedShape){
                case 'star':
                    console.log("HERE");
                    image(starStamp,mouseX,mouseY,stampSize,stampSize);
                    break;
                case 'horse':
                    image(horseStamp,mouseX,mouseY,stampSize,stampSize);
                    break;
                case 'bird':
                    image(birdStamp,mouseX,mouseY,stampSize,stampSize);
                    break;
            }
        }
    };
     // I took this function from given tools
    this.unselectTool = function() {
        select(".options").html("");
    };
}
```

style.css

```
html, body {
  margin: 0px;
  height: 100%;
}

#sidebar {
    grid-area: sidebar;
    overflow-y: scroll;
}

#content {
    grid-area: content;
}

.header {
    grid-area: header;
    font-family: Helvetica, sans-serif
}

.footer{
  grid-area: footer;
}

.sideBarItem{
    max-height: 50px;
    max-width: 50px;
    padding:5px;
}

.sideBarItem img{
    max-height: 50px;
    max-Width:50px;
}

.colourPalette{
    grid-area: colourP;
    display:flex;
    flex-direction:grid;
    flex-flow: wrap;
}

.options{
    grid-area: options;
    padding: 15px;
}

.colourSwatches{
    box-sizing: border-box;
    width: 40px;
    height: 40px;
    max-height: 40px;
    max-width: 40px;
    margin: 5px;
}


    .wrapper {
        display: grid;
        height: 100%;
        grid-template-columns: 70px 230px  minmax(500px, 1fr);
        grid-template-rows: 35px minmax(500px, 1fr) 160px;
        grid-template-areas:
               "header header header"
                "sidebar content content"
                "colourP colourP options";
        background-color: #fff;
        color: #444;
    }
.box {
  background-color: #444;
  color: #fff;
  font-size: 150%;
}

.header {
  background-color: #999;
}

```

toolbox.js

```
//container object for storing the tools. Functions to add new tools and select a tool
function Toolbox() {

    var self = this;

    this.tools = [];
    this.selectedTool = null;

    var toolbarItemClick = function() {
        //remove any existing borders
        var items = selectAll(".sideBarItem");
        for (var i = 0; i < items.length; i++) {
            items[i].style('border', '0')
        }

        var toolName = this.id().split("sideBarItem")[0];
        self.selectTool(toolName);

        //call loadPixels to make sure most recent changes are saved to pixel array
        loadPixels();

    }

    //add a new tool icon to the html page
    var addToolIcon = function(icon, name) {
        var sideBarItem = createDiv("<img src='" + icon + "'></div>");
        sideBarItem.class('sideBarItem')
        sideBarItem.id(name + "sideBarItem")
        sideBarItem.parent('sidebar');
        sideBarItem.mouseClicked(toolbarItemClick);


    };

    //add a tool to the tools array
    this.addTool = function(tool) {
        //check that the object tool has an icon and a name
        if (!tool.hasOwnProperty("icon") || !tool.hasOwnProperty("name")) {
            alert("make sure your tool has both a name and an icon");
        }
        this.tools.push(tool);
        addToolIcon(tool.icon, tool.name);
        //if no tool is selected (ie. none have been added so far)
        //make this tool the selected one.
        if (this.selectedTool == null) {
            this.selectTool(tool.name);
        }
    };

    this.selectTool = function(toolName) {
        //search through the tools for one that's name matches
        //toolName
        for (var i = 0; i < this.tools.length; i++) {
            if (this.tools[i].name == toolName) {
                //if the tool has an unselectTool method run it.
                if (this.selectedTool != null && this.selectedTool.hasOwnProperty(
                        "unselectTool")) {
                    this.selectedTool.unselectTool();
                }
                //select the tool and highlight it on the toolbar
                this.selectedTool = this.tools[i];
                select("#" + toolName + "sideBarItem").style("border", "2px solid blue");

                //if the tool has an options area. Populate it now.
                if (this.selectedTool.hasOwnProperty("populateOptions")) {
                    this.selectedTool.populateOptions();
                }
            }
        }
    };


}
```