University of London

Bachelor of Science (Honours) in Computer Science (Machine Learning and Artificial Intelligence)

Individual Midterm Report for Week 12

<Nguyen Ngoc Quoc Cuong> <SIM ID: 10245826>

Academic Year 2023

Year 1 Semester 2

Task:

Please upload your code in text format (not screenshots) in a PDF file here. Please clearly label with start and end comments exactly which sections of code you personally wrote without assistance. 5% of the marks for this coursework are reserved for this part.

Assuming "without assistance" means sections that I come up with through research and personal planning, without anyone giving me a hint or me referring to a base template or someone giving me the solution.
Sections that I wrote without assistance are highlighted in green.

Table of content

index.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 →
    [start]
    <script src="lib/p5.min2.js"></script>
    [end]
    <script src="lib/p5.js"></script>

    <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>
     [start]
    <script src="stampTool.js"></script>
    <script src="editableShapeTool.js"></script>
    <script src="eraserTool.js"></script>
    <script src="grid.js"></script>
    <script src="addonHelpersStuff.js"></script>
     [end]
    <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>
           [start]
           <div id="dropbox">dropbox</div>
           <div id="slider">Thickness</div>
           <div id="radio">Layer:</div>
           [end]
  </div>
  </body>
</html>
```
——---

sketch.js

```
//global variables that will store the toolbox colour palette
//and the helper functions
var toolbox = null;
var colourP = null;
var helpers = null;
[start]
//Thickness helper function related variables
var slider;
var sW;

//Layers helper function related variables
var radio;
var topLayer;
var middleLayer;
var botLayer;
var backgroundIMG;

//p5.disableFriendlyErrors = true; // disables FES
var canvasW;
var canvasH;
[end]
function preload() {
        [start]
        backgroundIMG = loadImage('./assets/whiteBackground.png');

        //drop box to chose a grid
        GridPreload();

        //To create a slider to change thickness value
        ThicknessPreload();

        //To create a radio for the user to chose a drawing layer
        LayersPreload();

        //preloading all of the images related to the stamp tool
        StampToolSetUp();
        [end]
}

function setup() {

        //create a canvas to fill the content div from index.html
```

```
	canvasContainer = select('#content');
	[start]
	canvasW = canvasContainer.size().width;
	canvasH = canvasContainer.size().height;
	[end]
	var c = createCanvas(canvasW, canvasH);
	c.parent("content");

	[start]
	//my helpers
	LayersSetup();
	GridSetup();
	ESTSetup();
	[end]

	//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());

	[start]
	toolbox.addTool(new StampTool());
	toolbox.addTool(new EdittableST());
	toolbox.addTool(new EraserTool());
	[end]
}

[start]
//limit the times the layers will be rendered
var drawTimer = false;
var y = 0;
[end]

function draw() {
	[start]
	if (drawTimer){
		if(mouseIsPressed){
			y = 0;
		}
		//to generate the white background image
```

```
                image(backgroundIMG,0,0,canvasW, canvasH);
        [end]
                //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!");
                }

                [start]
                //my helpers
                ThicknessSliderOutput();
                gridSelection();
                layersDraw();

                //The canvas will stop rendering afte a while when the mouse is not clicked
                y++;
                print(y);
                if(y == 3){
                        drawTimer = false;
                }
                if (drawTimer == false){
                        noLoop();
                }
                print("draw running");
        }

        //frame rate
        let fps = frameRate();
        fill(255);
        stroke(0);
        text("FPS: " + fps.toFixed(2), 10, height - 10);
        [end]
}

[start]
//To change var drawTimer = true.
function mousePressed(){
        drawTimer = true;
        loop();
}
[end]
```
—---

addonHelpersStuff.js

```
[start]
//Layers
function LayersSetup(){
	//To create three blank transparent canvas that stack on top of one another.
	topLayer = createGraphics(canvasW, canvasH);
	middleLayer = createGraphics(canvasW, canvasH);
	botLayer = createGraphics(canvasW, canvasH);
	topLayer.background(0,0,0,0);
	middleLayer.background(0,0,0,0);
	botLayer.background(0,0,0,0);

	//To make make the rotation of the stamp easier less confusing.
	botLayer.angleMode(DEGREES);
	botLayer.imageMode(CENTER);
	middleLayer.angleMode(DEGREES);
	middleLayer.imageMode(CENTER);
	topLayer.angleMode(DEGREES);
	topLayer.imageMode(CENTER);
}

//To allow the user to select which layer to draw on.
function LayersPreload(){
	this.radio=createRadio();
	this.radio.option("BaseLayer");
	this.radio.option("2ndLayer");
	this.radio.option("3rdLayer");
	this.radio.parent("radio");
}
function radioEvent(){
	var value= radio.value();
	print(value);
	return value;
}
//To allow rendering of layers in sketch.js.
function layersDraw(){
	image(botLayer,0,0);
	image(middleLayer,0,0);
	image(topLayer,0,0);
	image(gridLayer,0,0);
	image(ESTlayer,0,0);
}
[end]

//To check whether the mouse is on the canvas.
function mouseOnCanvas(){
	if ((mouseX < canvasW && mouseX > 0)&&(mouseY < canvasH && mouseY > 0)){
				return true;
			}
			else{
				return false;
			}
}

[start]
//Grid
function GridSetup(){
				//To generate a layer for the grids.
				gridLayer =  createGraphics(canvasW, canvasH);
				gridLayer.background(0,0,0,0);
}
//To allow the user to pick which type of grid they wants.
function GridPreload(){
			this.dropbox = createSelect();
   this.dropbox.option("None");
   this.dropbox.option("Camera Grid");
   this.dropbox.option("Line Grid");
   this.dropbox.parent("dropbox");
}

//Thickness
function ThicknessPreload(){
			//To create a slider so the user can change their tools' thickness.
			this.slider = createSlider(0,50,0);
   this.slider.parent("slider");
}
function ThicknessSliderOutput(){
			var thickness = sliderSelected();
			if (thickness > 0){
				//For general tools.
				sW = map(thickness,0,50,2,60);

				//For spraycan.
				spread = map(thickness,0,50,0,60);
				points = map(thickness,0,50,13,240);
			}
			else if(thickness <= 0){
				//For general tools.
				sW = 2;

				//For spraycan.
				spread = 5;
				points = 8;
			}
}
```

```
function sliderSelected(){
        var item = slider.value();
        return item;
}
[end]
```
—---

colourPalette.js

```
//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);
                [start]
                topLayer.fill(c);
                middleLayer.fill(c);
                topLayer.stroke(c);
                middleLayer.stroke(c);
                botLayer.fill(c);
                botLayer.stroke(c);
                [end]
                //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();
}
```
—--

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: [start]45px;[end]
        max-width: [start]45px;[end]
        padding:[start]5px;[end]
        margin: 0 auto;
}

.sideBarItem img{
        max-height: 40px;
        max-width: 40px;
}

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

.options{
        grid-area: options;
        padding: 15px;
}
[start]
.dropbox{
        grid-area: dropbox;
        padding: 15px;

}

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

.radio{
        grid-area: radio;
        padding: 15px;
}
[end]
.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 header header header"
                        "sidebar content content content content content"
                        "colourP colourP options [start] radio slider dropbox [end]";
                        /* radio slider dropbox */
                background-color: #fff;
                color: #444;
        }
.box {
  background-color: #444;
  color: #fff;
  font-size: 150%;
}

.header {
  background-color: #999;
}
```
—--

editableShapeTool.js

```
[start]
var isFillBPressed = false;
let tempEST;

//To generate a layer for the red dots of the shape during the editting state.
function ESTSetup(){
        ESTlayer = createGraphics(canvasW, canvasH);
        ESTlayer.background(0,0,0,0);
}
[end]
function EdittableST(){
        //set an icon and a name for the object
        this.icon = "./assets/flower.jpg";
        this.name = "editableST";

        //buttons.
        var editB;
        var finishB;
```

```
	[start]
	var fillB;
	var editState = false;
	[end]
	var currentPoints = [];

	this.draw = function() {
		[start]
		ESTSetup();
		[end]
		//To prevent the shape from being drawn onto the canvas yet.
		botLayer.updatePixels();
		middleLayer.updatePixels();
		topLayer.updatePixels();

		[start]
		//To identify the user's chosen layer.
		var layer = radioEvent();
		[end]
		if (mouseIsPressed && !editState){
			//This 'if' statement is disabled if in editing state.
			//To check whether the cursor is on the canvas.
			if(!mouseOnCanvas()){
				return;
			}
			//To log in the position of the cursor when the mouse is pressed.
			currentPoints.push({x:mouseX,y:mouseY});
		}

		//After pressing the 'Edit' button. To allow the user to change the shape's
contour.
		if(editState){
			for(var i=0; i<currentPoints.length;i++){
				var editRange =
dist(currentPoints[i].x,currentPoints[i].y,mouseX,mouseY);
				if (editRange <= 15){
					if(mouseIsPressed == true){
						currentPoints[i].x = mouseX;
						currentPoints[i].y = mouseY;
					}
				}
			}
		}
		[start]
		//To Un-fill the shape
		if (!isFillBPressed){
			botLayer.noFill();
			middleLayer.noFill();
```

```
                    topLayer.noFill();
            }
            [end]
            //To draw the contour of the shape
            if (currentPoints.length > 0){
                    if (layer == "BaseLayer"){
                            [start]
                            if (isFillBPressed){
                                    botLayer.point(currentPoints[0].x,currentPoints[0].y);
                                    tempEST =
botLayer.get(currentPoints[0].x,currentPoints[0].y);
                                    botLayer.fill(tempEST);
                            }
                            [end]
                            botLayer.beginShape();
                                    for (var i=0;i<currentPoints.length;i++){
                                            botLayer.strokeWeight(sW);

botLayer.vertex(currentPoints[i].x,currentPoints[i].y);
                                    }
                            botLayer.endShape();
                    }
                    else if (layer == "2ndLayer"){
                            [start]
                            if (isFillBPressed){
                                    middleLayer.point(currentPoints[0].x,currentPoints[0].y);
                                    tempEST =
middleLayer.get(currentPoints[0].x,currentPoints[0].y);
                                    middleLayer.fill(tempEST);
                            }
                            [end]
                            middleLayer.beginShape();
                                    for (var i=0;i<currentPoints.length;i++){
                                            middleLayer.strokeWeight(sW);

middleLayer.vertex(currentPoints[i].x,currentPoints[i].y);
                                    }
                            middleLayer.endShape();
                    }
                    else if (layer == "3rdLayer"){
                            [start]
                            if (isFillBPressed){
                                    topLayer.point(currentPoints[0].x,currentPoints[0].y);
                                    tempEST =
topLayer.get(currentPoints[0].x,currentPoints[0].y);
                                    topLayer.fill(tempEST);
                            }
                            [end]
                    topLayer.beginShape();
                        for (var i=0;i<currentPoints.length;i++){
                            topLayer.strokeWeight(sW);

topLayer.vertex(currentPoints[i].x,currentPoints[i].y);
                        }
                    topLayer.endShape();
                }
            }

            //To draw the red dots for the editing state
            if(editState){
                for(var i = 0; i<currentPoints.length;i++){
                    drawDotsforEST(i);
                }
            }
        };

        this.unselectTool = function(){
            select(".options").html("");
            finishBpressed();
        };

        this.populateOptions = function(){
            console.log("edittableStool selected");
            //To set noFill() on by default.
            noFill();
            //To save the canvas when the tool first load.
            botLayer.loadPixels();
            middleLayer.loadPixels();
            topLayer.loadPixels();

            //To generate the buttons.
            select(".options").html("<div id='startEditing'></div>[start]&nbsp;<div
id='FillTS'></div>[end]<br><div id='stopNfinish'></div>");
            editB = createButton("Edit");
            finishB = createButton("Finish");
            fillB = createButton("Fill the shape")
            editB.parent("startEditing");
            finishB.parent("stopNfinish");
            fillB.parent("FillTS")
            editB.mousePressed(editBpressed);
            finishB.mousePressed(finishBpressed);
            fillB.mousePressed(fillBpressed);

            editB.style("display","block");
            finishB.style("display","block");
            fillB.style("display","block");
```

```
};

//Edit button
function editBpressed (){
        print("edit button pressed");
        if (editState){
                editState = false;
                editB.html("Edit");

        }
        else{
                editState = true;
                editB.html("Add Vertices");
        }
}

//Finish button
function finishBpressed(){
        //Called again to clear the canvas off red dots from the editing state.
        ESTSetup();
        editB.style("display","block");
        finishB.style("display","block");
        editState = false;

        //To save the canvas
        botLayer.loadPixels();
        middleLayer.loadPixels();
        topLayer.loadPixels();

        //To reset the array for the next shape.
        currentPoints = [];
        editB.html("Edit");
        print("finsihBpress is ran");
}
[start]
//Fill button
function fillBpressed(){
        if(isFillBPressed){
                isFillBPressed = false;
                fillB.html("Fill the shape");
        }
        else{
                isFillBPressed = true;
                fillB.html("Un-fill the shape");
        }
}
[end]
//To draw red dots
```

```
	function drawDotsforEST(i){
		ESTlayer.fill("red");
		ESTlayer.strokeWeight(1);
		ESTlayer.ellipse(currentPoints[i].x,currentPoints[i].y,10);
		ESTlayer.noFill();
		ESTlayer.strokeWeight(sW);
	}
}
```
—---

eraserTool.js

```
function EraserTool(){
	//set an icon and a name for the object
	this.icon = "assets/eraser.jpg";
	this.name = "eraser";

	//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.draw = function(){
		//if the mouse is pressed
		if(mouseIsPressed){
			if(!mouseOnCanvas()){
				return;
			}
			//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;
				botLayer.loadPixels();
				middleLayer.loadPixels();
				topLayer.loadPixels();
			}
			//if we already have values for previousX and Y we can draw a line
from
			//there to the current mouse location
			Else{
				[start]
				var layer = radioEvent();

				//To 'erase' drawings
				if (layer == "BaseLayer"){
```

```
                                        botLayer.strokeWeight(sW);
                                        botLayer.blendMode(REMOVE);
                                        botLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                                        previousMouseX = mouseX;
                                        previousMouseY = mouseY;
                                }
                                else if (layer == "2ndLayer"){
                                        middleLayer.strokeWeight(sW);
                                        middleLayer.blendMode(REMOVE);
                                        middleLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                                        previousMouseX = mouseX;
                                        previousMouseY = mouseY;
                                }
                                else if (layer == "3rdLayer"){
                                        topLayer.strokeWeight(sW);
                                        topLayer.blendMode(REMOVE);
                                        topLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                                        previousMouseX = mouseX;
                                        previousMouseY = mouseY;
                                }
                        }
                        [end]
                }
                //if the user has released the mouse we want to set the previousMouse values
                //back to -1.
                else{
                        previousMouseX = -1;
                        previousMouseY = -1;
                        botLayer.loadPixels();
                        middleLayer.loadPixels();
                        topLayer.loadPixels();
                        [start]
                        botLayer.blendMode(BLEND);
                        middleLayer.blendMode(BLEND);
                        topLayer.blendMode(BLEND);
                        [end]
                }

        };
}
```
—--

freehandTool.js

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

	//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.draw = function(){
		//if the mouse is pressed
		if(mouseIsPressed){
			if(!mouseOnCanvas()){
				return;
			}
			//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;

				//save the current pixel Array (i edit)
				botLayer.loadPixels();
				middleLayer.loadPixels();
				topLayer.loadPixels();
			}
			//if we already have values for previousX and Y we can draw a line
from
			//there to the current mouse location
			Else{
				[start]
				var layer = radioEvent();

				//drawing the lines for freehandTool
				if (layer == "BaseLayer"){
					botLayer.strokeWeight(sW);
					[end]
					botLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
					[start]
					previousMouseX = mouseX;
					previousMouseY = mouseY;
				}
```

```
                    else if (layer == "2ndLayer"){
                        middleLayer.strokeWeight(sW);
                        [end]
                        middleLayer.line(previousMouseX, previousMouseY, 
mouseX, mouseY);
                        [start]
                        previousMouseX = mouseX;
                        previousMouseY = mouseY;
                    }
                    else if (layer == "3rdLayer"){
                        topLayer.strokeWeight(sW);
                        [end]
                        topLayer.line(previousMouseX, previousMouseY, 
mouseX, mouseY);
                        [start]
                        previousMouseX = mouseX;
                        previousMouseY = mouseY;
                    }
                }
                [end]
            }
            //if the user has released the mouse we want to set the previousMouse values
            //back to -1.
            else{
                previousMouseX = -1;
                previousMouseY = -1;
                botLayer.loadPixels();
                middleLayer.loadPixels();
                topLayer.loadPixels();
            }
        };
}
```
—---

grid.js

```
[start]
//grid
var gridLayer;
var gridType;
var dropbox;

function dropboxSelected(){
    var item = dropbox.value();
    return item;
}

function gridSelection(){
    gridLayer.strokeWeight(2);
    gridLayer.stroke(0,0,0,10);

    gridType = dropboxSelected();

    if  (gridType == "None"){
        gridLayer = createGraphics(canvasContainer.size().width,
canvasContainer.size().height);
        gridLayer.background(0,0,0,0);
    }
    if  (gridType == "Camera Grid"){
        gridLayer = createGraphics(canvasContainer.size().width,
canvasContainer.size().height);
        gridLayer.background(0,0,0,0);
        //draw the line of symmetry
        for (var i = 1; i <= 2; i++ ){
            gridLayer.line((canvasContainer.size().width / 3 * i)-7, 0,
(canvasContainer.size().width / 3 * i)-7, canvasContainer.size().height);
        }
        for (var i = 1; i <= 2; i++ ){

gridLayer.line(0,canvasContainer.size().height/3*i,canvasContainer.size().width,canvasConta
iner.size().height/3*i);
        }
    }
    if  (gridType == "Line Grid"){
        var boxW = 30;
        var boxH  = 13;
        gridLayer = createGraphics(canvasContainer.size().width,
canvasContainer.size().height);
        gridLayer.background(0,0,0,0);
        for (var i = 1; i <= boxW; i++ ){
            gridLayer.line((canvasContainer.size().width / boxW * i)-7, 0,
(canvasContainer.size().width / boxW * i)-7, canvasContainer.size().height);
        }
        for (var i = 1; i <= boxH; i++ ){

gridLayer.line(0,canvasContainer.size().height/boxH*i,canvasContainer.size().width,canvasC
ontainer.size().height/boxH*i);
        }
    }

};
[end]
```
—--

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() {
		[start]
		LayersSetup();
		[end]
	});

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

}
```

—---

lineToTool.js

```
//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(){
	//set an icon and a name for the object
	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(!mouseOnCanvas()){
				return;
			}
			//if it's the start of drawing a new line
			if(startMouseX == -1){
				startMouseX = mouseX;
				startMouseY = mouseY;
```

```
                    drawing = true;
                    //save the current pixel Array
                    botLayer.loadPixels();
                    middleLayer.loadPixels();
                    topLayer.loadPixels();
                }

                else{
                    //update the screen with the saved pixels to hide any previous
                    //line between mouse pressed and released
                    botLayer.updatePixels();
                    middleLayer.updatePixels();
                    topLayer.updatePixels();

                    [start]var layer = radioEvent();[end]

                    //draw the line
                    [start]
                    if (layer == "BaseLayer"){
                        botLayer.strokeWeight(sW);[end]
                        botLayer.line(startMouseX, startMouseY, mouseX,
mouseY);
                    }
                    [start]
                    else if (layer == "2ndLayer"){
                        middleLayer.strokeWeight(sW);[end]
                        middleLayer.line(startMouseX, startMouseY, mouseX,
mouseY);
                    }
                    [start]
                    else if (layer == "3rdLayer"){
                        topLayer.strokeWeight(sW);[end]
                        topLayer.line(startMouseX, startMouseY, mouseX,
mouseY);
                    }
                }

            }

            else if(drawing){
                //save the pixels with the most recent line and reset the
                //drawing bool and start locations
                botLayer.loadPixels();
                middleLayer.loadPixels();
                topLayer.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
		botLayer.updatePixels();
		middleLayer.updatePixels();
		topLayer.updatePixels();

		//do the drawing if the mouse is pressed
		if (mouseIsPressed) {
			if(!mouseOnCanvas()){
				return;
			}
			//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 {
                    [start]
                    var layer = radioEvent();
                    if (layer == "BaseLayer"){
                        botLayer.blendMode(BLEND);
                        botLayer.strokeWeight(sW);
                        [end]
                        botLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                        previousMouseX = mouseX;
                        previousMouseY = mouseY;

                        var oX = this.calculateOpposite(mouseX, "x");
                        var oY = this.calculateOpposite(mouseY, "y");
                        botLayer.line(previousOppositeMouseX,
previousOppositeMouseY, oX, oY);
                        previousOppositeMouseX = oX;
                        previousOppositeMouseY = oY;
                    }
                    [start]
                    else if (layer == "2ndLayer"){
                        middleLayer.blendMode(BLEND);
                        middleLayer.strokeWeight(sW);
                        [end]
                        middleLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                        previousMouseX = mouseX;
                        previousMouseY = mouseY;

                        var oX = this.calculateOpposite(mouseX, "x");
                        var oY = this.calculateOpposite(mouseY, "y");
                        middleLayer.line(previousOppositeMouseX,
previousOppositeMouseY, oX, oY);
                        previousOppositeMouseX = oX;
                        previousOppositeMouseY = oY;
                    }
                    [start]
                    else if (layer == "3rdLayer"){
                        topLayer.blendMode(BLEND);
                        topLayer.strokeWeight(sW);
                        [end]
```

```
                                        topLayer.line(previousMouseX, previousMouseY,
mouseX, mouseY);
                                        previousMouseX = mouseX;
                                        previousMouseY = mouseY;

                                        var oX = this.calculateOpposite(mouseX, "x");
                                        var oY = this.calculateOpposite(mouseY, "y");
                                        topLayer.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

                botLayer.loadPixels();
                middleLayer.loadPixels();
                topLayer.loadPixels();

                //push the drawing state so that we can set the stroke weight and colour
                push();
                strokeWeight(4);
                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() {
	botLayer.updatePixels();
	middleLayer.updatePixels();
	topLayer.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');
                        }
                });
        };
}
```

—---

sprayCanTool.js

```
var points;
var spread;

function SprayCanTool(){

        this.name = "sprayCanTool";
        this.icon = "assets/sprayCan.jpg";

        this.draw = function(){
                var r = random(5,10);

                //save the current pixel Array (i edit)
                botLayer.loadPixels();
                middleLayer.loadPixels();
                topLayer.loadPixels();
                [start]
                //to revert the strokeWeight changes of other tools
                botLayer.strokeWeight(1);
                middleLayer.strokeWeight(1);
                topLayer.strokeWeight(1);
                [end]
                if(mouseIsPressed){
                        if(!mouseOnCanvas()){
                                return;
                        }
                        [start]
                        var layer = radioEvent();

                        if (layer == "BaseLayer"){[end]
                                for(var i = 0; i < points; i++){
                                        botLayer.point(random(mouseX-spread, mouseX +
spread), random(mouseY-spread, mouseY+spread));
                                }
                        }
                        [start]
                        else if (layer == "2ndLayer"){[end]
                                for(var i = 0; i < points; i++){
```

```
					middleLayer.point(random(mouseX-spread, mouseX +
spread), random(mouseY-spread, mouseY+spread));
				}
			}
			[start]
			else if (layer == "3rdLayer"){[end]
				for(var i = 0; i < points; i++){
					topLayer.point(random(mouseX-spread, mouseX +
spread), random(mouseY-spread, mouseY+spread));
				}
			}
		}
		else{
			previousMouseX = -1;
			previousMouseY = -1;
			botLayer.loadPixels();
			middleLayer.loadPixels();
			topLayer.loadPixels();
		}
	};
}
```
—---

stampTool.js

```
[start]
// var mouseIsReleasedForStamp = false;
// function mouseReleased() {
//	mouseIsReleasedForStamp = true;
// }

//Setting up the variables for the dropbox of stamp images.
let catloaf;
let catsuprised;
let dogcute;
let dogmeh;
let dogscared;
var stampDropbox;

//used in sketch.js, to preload the stamp tool's images.
function StampToolSetUp(){
	catloaf = loadImage('./assets/cat1.png');
	catsuprised = loadImage('./assets/cat2.png');
	dogcute = loadImage('./assets/dogcute.png');
	dogmeh = loadImage('./assets/dogmeh.png');
	dogscared = loadImage('./assets/dogscared.png');
}
[end]
```

```
function StampTool(){
        //set an icon and a name for the object.
        this.name = "StampTool";
        this.icon = "./assets/stamp.jpg";
        [start]
        //stamping related types of states.
        var stampingState = true;
        var rotatingState = false;
        var objTemp;
        var stampLoadPixel=true;
        [end]
        this.draw = function(){
                [start]
                //To set the size of images.
                var thickness = sliderSelected();
                if (thickness > 0){
                        var sW = map(thickness,0,50,10,400);
                }
                else if(thickness <= 0){
                        var sW = 1;
                }
                var stampSize = sW;
                [end]
                //To set the selected image to a variable.
                let tempImg;
                var pickedStamp = selectedStamp();
                if (pickedStamp == "CatLoaf"){
                        tempImg = catloaf;
                }
                else if (pickedStamp == "CatSurprised"){
                        tempImg = catsuprised;
                }
                else if (pickedStamp == "DogMaid"){
                        tempImg = dogcute;
                }
                else if (pickedStamp == "DogMeh"){
                        tempImg = dogmeh;
                }
                else if (pickedStamp == "DogScared"){
                        tempImg = dogscared;
                };
                [start]
                //To identify which layer the user chose to draw on.
                var layer = radioEvent();
                [end]
                if(mouseIsPressed){
                        //To check whether the mouse is on the canvas.
                        print("mouseIsPressed");
```

```
                    if(!mouseOnCanvas()){
                              return;
                    }

                    [start]
                    if (stampingState){
                              //To save the coordinate of where the mouse clicked.
                              objTemp ={x:mouseX, y:mouseY,rAngle:0};

                              //To save the current canvas.
                              botLayer.loadPixels();
                              middleLayer.loadPixels();
                              topLayer.loadPixels();

                              //To change state.
                              rotatingState = true;
                              stampingState = false;
                    }
                    [end]
          }

          [start]
          if(rotatingState){
                    //To continuously remove previous iteration of the stamps when mouse
is hold.
                    botLayer.updatePixels();
                    middleLayer.updatePixels();
                    topLayer.updatePixels();

                    //To decide which direction the stamp will rotate in.
                    var dx = (mouseX - objTemp.x)/2;
                    objTemp.rAngle = dx;

                    //To reset everything and draw the final iteration of the stamp
orientation on the canvas.
                    if(mouseIsPressed == false){
                              rotatingState = false;
                              stampingState = true;
                              stampLoadPixel=true;
                              botLayer.loadPixels();
                              middleLayer.loadPixels();
                              topLayer.loadPixels();
                              mouseIsReleasedForStamp = false;
                    }


                    //To draw the stamp.
                    if (layer == "BaseLayer"){
                              botLayer.push();
                              botLayer.translate(objTemp.x,objTemp.y);
                              botLayer.rotate(objTemp.rAngle);
                              botLayer.image(tempImg,0,0,stampSize,stampSize);
                              botLayer.pop();
                    }
                    else if (layer == "2ndLayer"){
                              middleLayer.push();
                              middleLayer.translate(objTemp.x,objTemp.y);
                              middleLayer.rotate(objTemp.rAngle);
                              middleLayer.image(tempImg,0,0,stampSize,stampSize);
                              middleLayer.pop();
                    }
                    else if (layer == "3rdLayer"){
                              topLayer.push();
                              topLayer.translate(objTemp.x,objTemp.y);
                              topLayer.rotate(objTemp.rAngle);
                              topLayer.image(tempImg,0,0,stampSize,stampSize);
                              topLayer.pop();
                    }

                }
                [end]

        };

        this.unselectTool = function(){
                select(".options").html("");
        };

        this.populateOptions = function(){
                console.log("stamp tool selected");
                //To create a stamp dropbox and an instruction about rotating the stamp.
                select(".options").html("<div id='stampSelection'>Stamp</div><br><div>Drag
your mouse left or right to rotate</div>");

                //To create a dropbox containing all stamps selection.
                stampDropbox = createSelect();
                stampDropbox.option("CatLoaf");
                stampDropbox.option("CatSurprised");
                stampDropbox.option("DogMaid");
                stampDropbox.option("DogMeh");
                stampDropbox.option("DogScared");
                stampDropbox.parent("stampSelection");
                stampDropbox.changed(selectedStamp);
        };

        //To identify which stamp the user chose.
```

```
	function selectedStamp(){
		var value= stampDropbox.value();
		return value;
	}

}
```
—---

toolbox.js (no change)

```
//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();
                }
            }
        }
    };


}
```
—---