Liam Gonzales 3181360

Final Assignment Document

Rock Paper Scissors

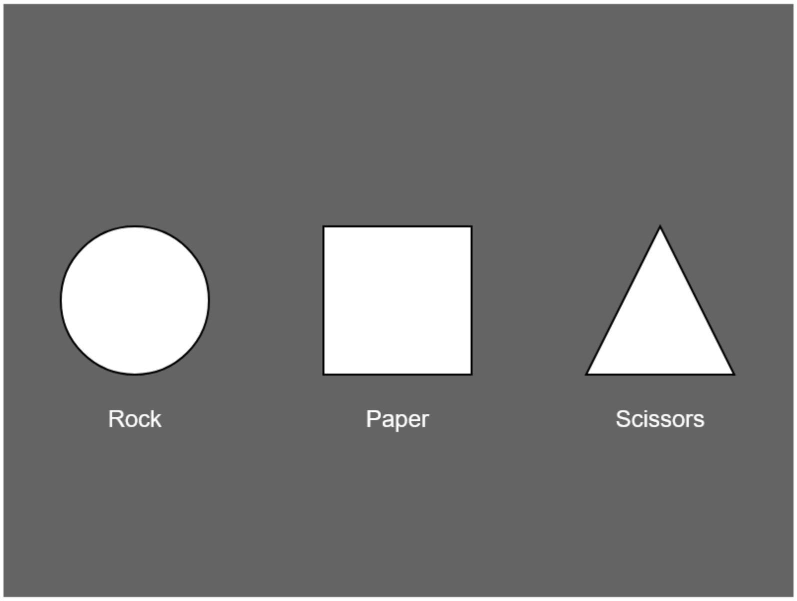
Code Key:

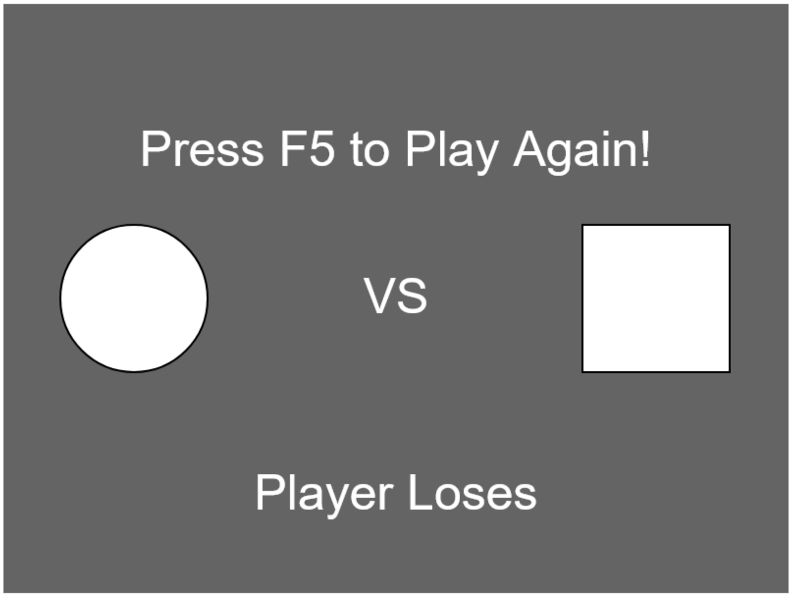
Variables – arr_arrayname for arrays, no naming convention otherwise

Classes – cl_classname

Class Methods – mt_methodname

Functions – fn_functionname



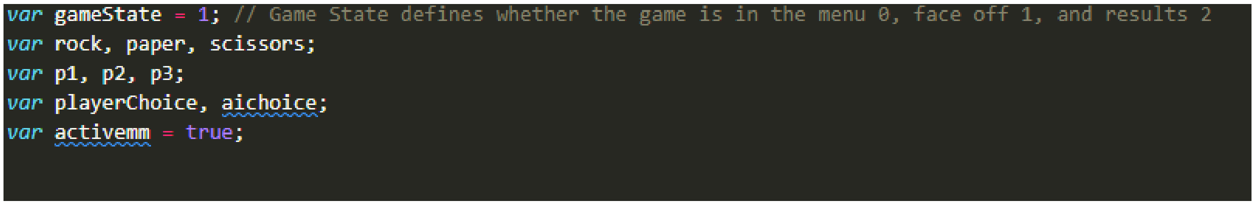

Press F5 to Play Again!

VS

Player Loses

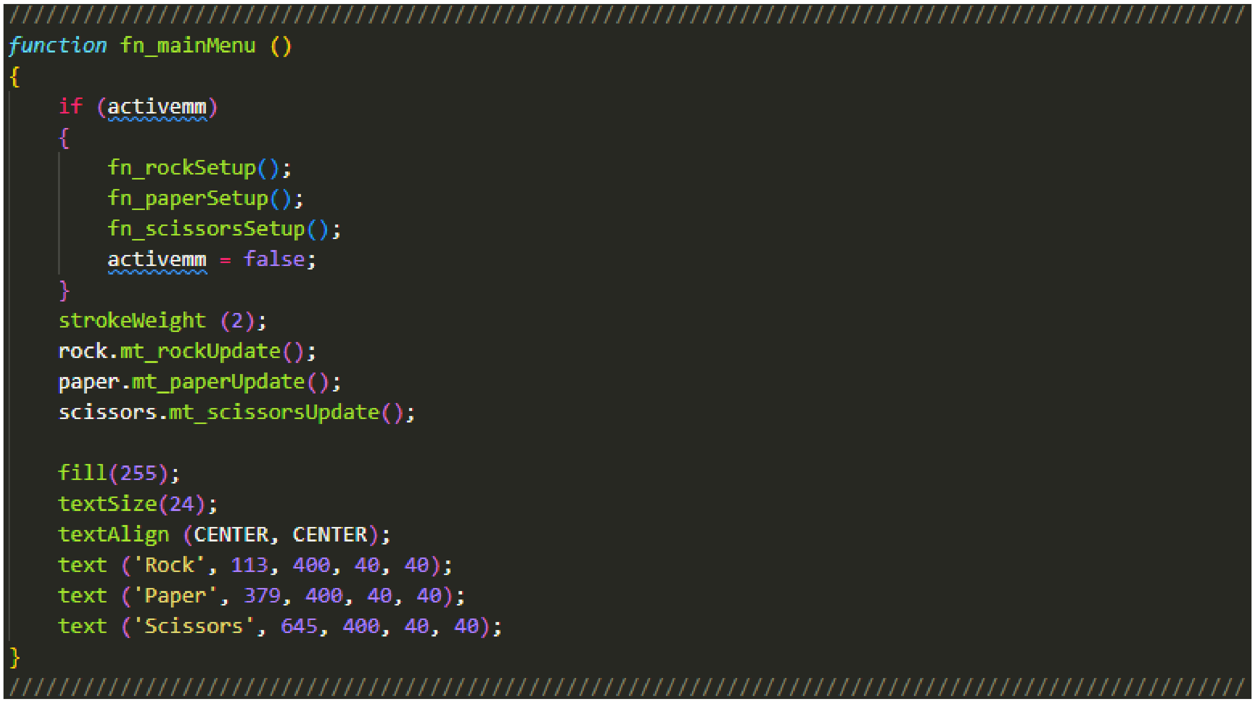
Game States

I decided to separate the functions of the game into two states, switching between them based on the input.



```
var gameState = 1; // Game State defines whether the game is in the menu 0, face off 1, and results 2
var rock, paper, scissors;
var p1, p2, p3;
var playerChoice, aichoice;
var activemm = true;
```

The very first line of code declared a variable gameState and assigned it a value of 1 which would be associated with the main menu.

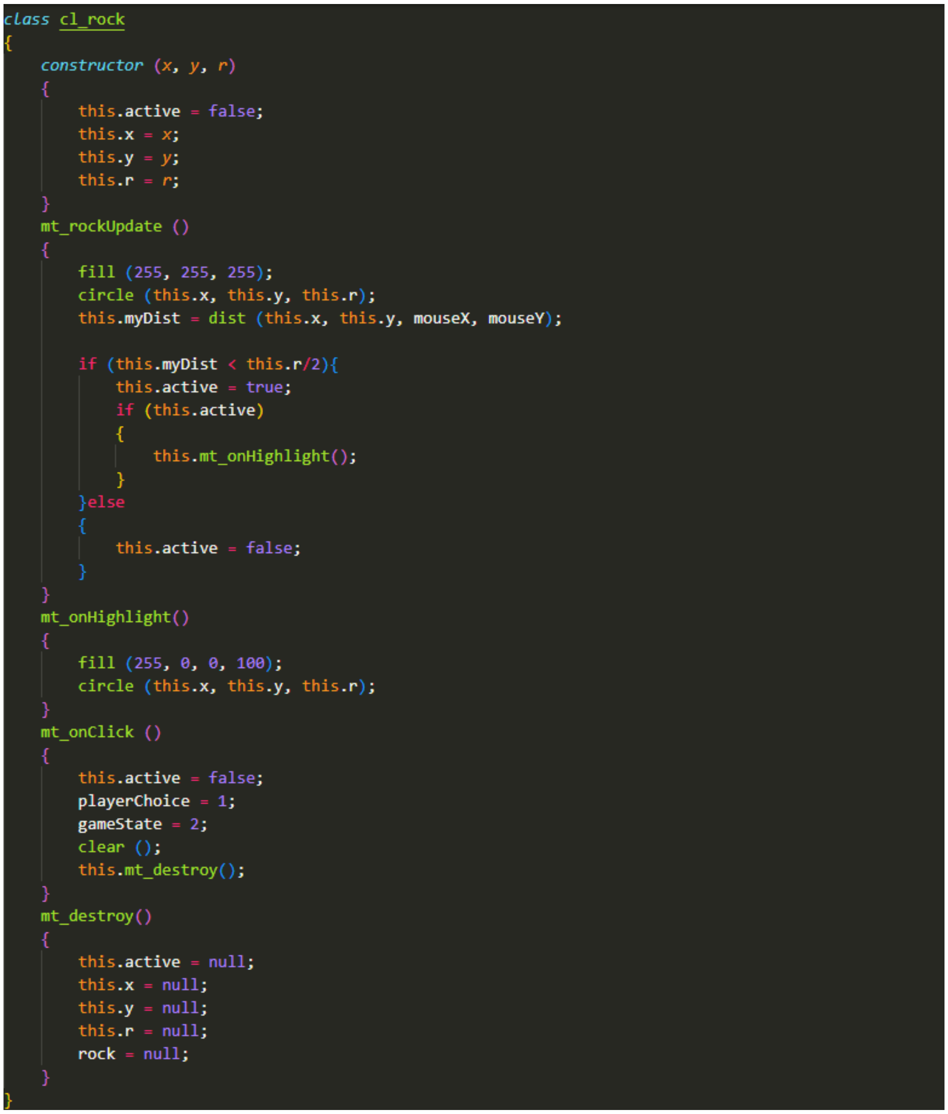


```
////////////////////////////////////////////////////////////////////////////////////////////////////
function fn_mainMenu ()
{
    if (activemm)
    {
        fn_rockSetup();
        fn_paperSetup();
        fn_scissorsSetup();
        activemm = false;
    }
    strokeWeight (2);
    rock.mt_rockUpdate();
    paper.mt_paperUpdate();
    scissors.mt_scissorsUpdate();

    fill(255);
    textSize(24);
    textAlign (CENTER, CENTER);
    text ('Rock', 113, 400, 40, 40);
    text ('Paper', 379, 400, 40, 40);
    text ('Scissors', 645, 400, 40, 40);
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

fn_mainMenu is called when gameState is 1 it runs an if statement that, if activemm is true calls the setup functions fn_rockSetup, fn_paperSetup, and fn_scisssorsSetup, and then reassigns activemm to false.

The stroke weight is set to 2 and the methods mt_rockUpdate, mt_paperUpdate, mt_scissorsUpdate are called for their respective objects. The fill is set to a value of 255, or white, the text size is set to 24 and the text alignment is set to center. Three p5.js text functions are called and the words 'Rock', 'Paper' and 'Scissors' are drawn to the screen at x and y positions hard coded to be below the shapes representing Rock Paper and Scissors. Their text boxes are set to 40 width and height.

The classes followed the same formula, constructors which received the x, y and size values, they also had an active parameter which was set to false on construction. They then had between 3 and 5 methods, an update method, an onHighlight method an onClick method and a destroy method. All of these acted in a very similar fashion see below for the rock class and its methods.

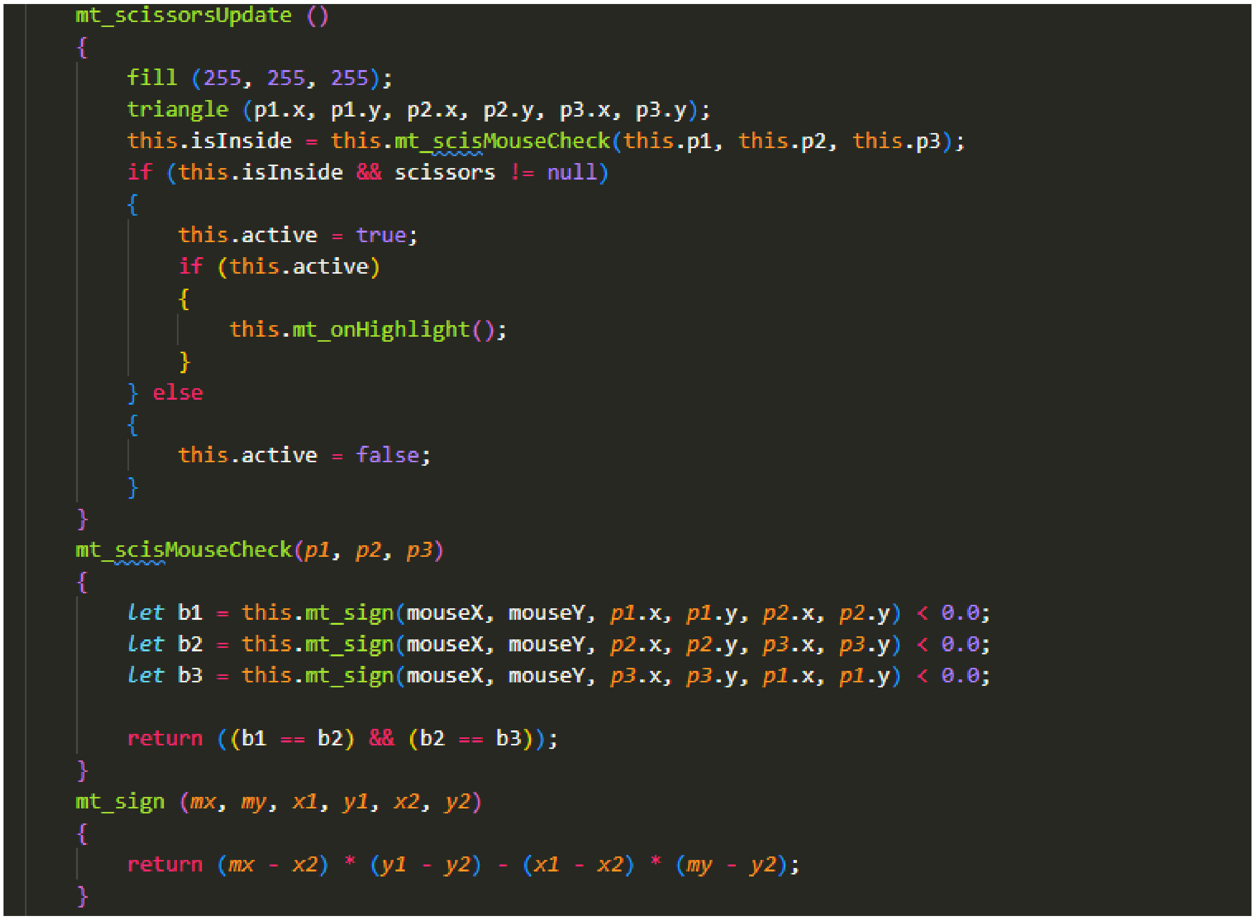


```
class cl_rock
{
    constructor (x, y, r)
    {
        this.active = false;
        this.x = x;
        this.y = y;
        this.r = r;
    }
    mt_rockUpdate ()
    {
        fill (255, 255, 255);
        circle (this.x, this.y, this.r);
        this.myDist = dist (this.x, this.y, mouseX, mouseY);

        if (this.myDist < this.r/2){
            this.active = true;
            if (this.active)
            {
                this.mt_onHighlight();
            }
        }else
        {
            this.active = false;
        }
    }
    mt_onHighlight()
    {
        fill (255, 0, 0, 100);
        circle (this.x, this.y, this.r);
    }
    mt_onClick ()
    {
        this.active = false;
        playerChoice = 1;
        gameState = 2;
        clear ();
        this.mt_destroy();
    }
    mt_destroy()
    {
        this.active = null;
        this.x = null;
        this.y = null;
        this.r = null;
        rock = null;
    }
}
```

The Update method was called every frame and redrew the object to the screen with its default colour value. I used a nested if statement to call the method mt_onHighlight, first checking if the mouses distance to the origin of the object is within its size, setting the active parameter to true and then checking if the active parameter is true then calling the method. Failing these checks would set active to false.

The method mt_onHighlight redrew the object with a translucent red colour. The mt_onClick method was called when the p5.js function mouseClicked was called, it set active to false, adjusted the playerchoice variable, changed the gameState, cleared the canvas and called the destroy method. The mt_destroy method changed all possible parameters of the object to null to free space. This was my first attempt at using a destructor of sorts so it wasn't as elegant as it could have been.

The scissors class had a few changes as it had to take extra steps in its hit tracking, see below.



```
mt_scissorsUpdate ()
{
    fill (255, 255, 255);
    triangle (p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);
    this.isInside = this.mt_scisMouseCheck(this.p1, this.p2, this.p3);
    if (this.isInside && scissors != null)
    {
        this.active = true;
        if (this.active)
        {
            this.mt_onHighlight();
        }
    } else
    {
        this.active = false;
    }
}
mt_scisMouseCheck(p1, p2, p3)
{
    let b1 = this.mt_sign(mouseX, mouseY, p1.x, p1.y, p2.x, p2.y) < 0.0;
    let b2 = this.mt_sign(mouseX, mouseY, p2.x, p2.y, p3.x, p3.y) < 0.0;
    let b3 = this.mt_sign(mouseX, mouseY, p3.x, p3.y, p1.x, p1.y) < 0.0;

    return ((b1 == b2) && (b2 == b3));
}
mt_sign (mx, my, x1, y1, x2, y2)
{
    return (mx - x2) * (y1 - y2) - (x1 - x2) * (my - y2);
}
```

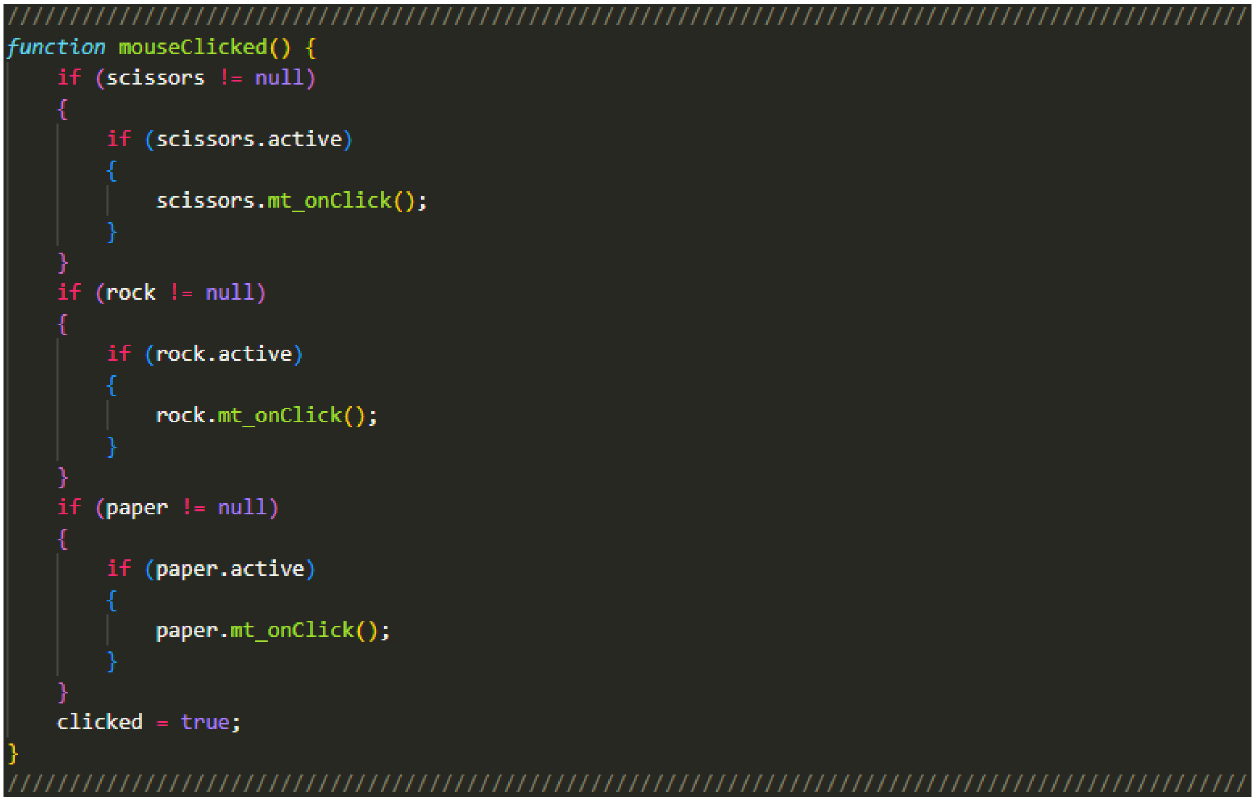
I had never worked with this complex level of hit tracking so I used online resources like stack overflow for suggestions and found this implementation. First thing to note is that p1, p2, and p3 the position parameters for the triangle are p5.js vectors which I used as it allowed me to easily store two values in one variable.

The value of a method mt_scisMouseCheck which used the p1, p2, and p3 parameters was assigned to the parameter isInside. mt_scisMouseCheck declared variables b1, b2 and b3, to the value of another method mt_sign which took the mouses x and y position, and the x and y positions of p1, p2 and p3. The method returned a Boolean if b1 is equal to b2 AND b2 is equal to b3.

The method mt_sign returned the value of this equation:

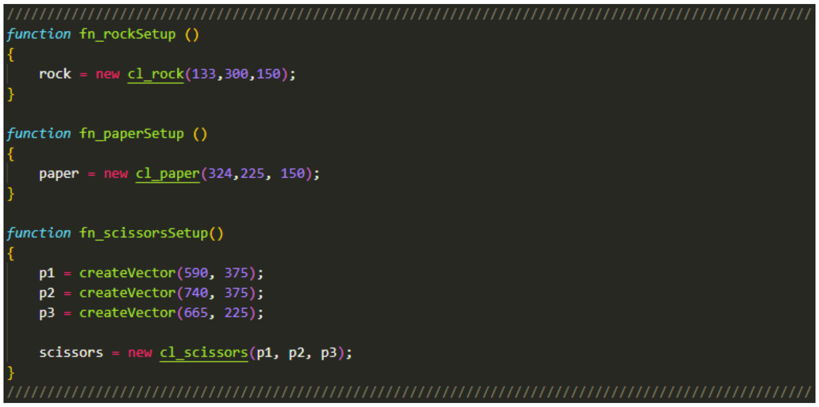
= (mouseX – x2) * (y1 – y2) – (x1 – x2) * (mouseY – y2)

These methods would work together to deduce whether or not the mouses current position within the canvas lay within a triangular space denoted by the x and y values of the three vectors used to draw the triangle.



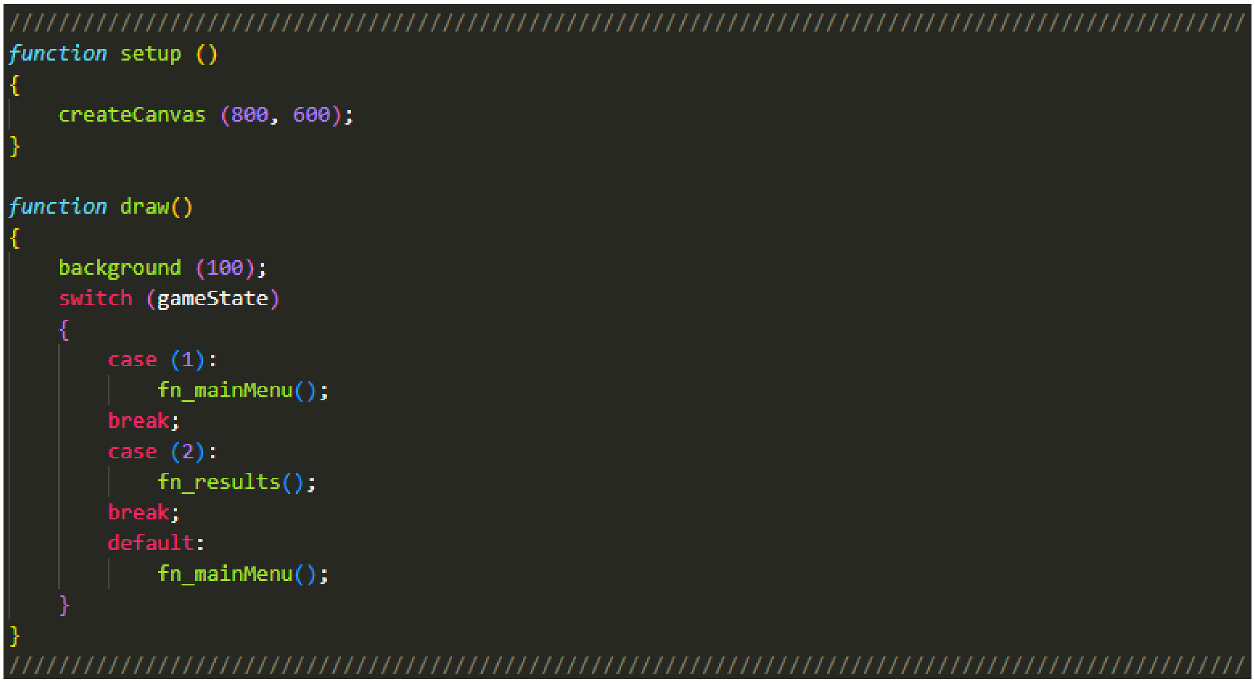
```
////////////////////////////////////////////////////////////////////////////////////////////////////
function mouseClicked() {
    if (scissors != null)
    {
        if (scissors.active)
        {
            scissors.mt_onClick();
        }
    }
    if (rock != null)
    {
        if (rock.active)
        {
            rock.mt_onClick();
        }
    }
    if (paper != null)
    {
        if (paper.active)
        {
            paper.mt_onClick();
        }
    }
    clicked = true;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

The mouseClicked function had to be rewritten for my purposes and used nested if statements to detect first, if the object in question was in fact null and then if the objects active parameter was true, only then would if call the mt_onClick method. It would also set clicked to true although I believe this use would later be deprecated.



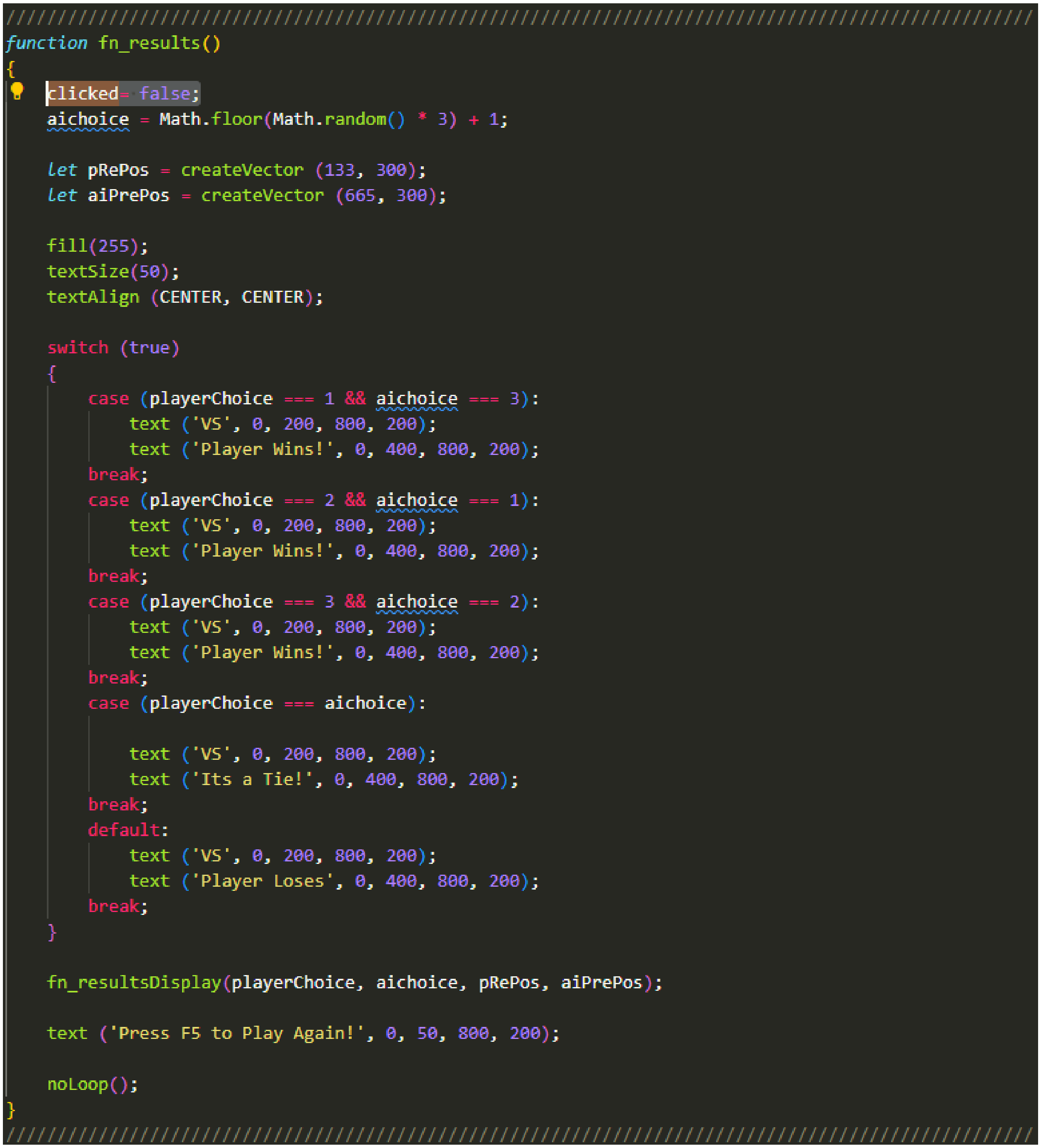
```
////////////////////////////////////////////////////////////////////////////////////////////////////
function fn_rockSetup ()
{
    rock = new cl_rock(133,300,150);
}

function fn_paperSetup ()
{
    paper = new cl_paper(324,225, 150);
}

function fn_scissorsSetup()
{
    p1 = createVector(590, 375);
    p2 = createVector(740, 375);
    p3 = createVector(665, 225);

    scissors = new cl_scissors(p1, p2, p3);
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

The setup functions assign the objects to variables and are used for their initial drawing at the start of the main menu function. The createVector functions for the triangle can also be found here.



```
/////////////////////////////////////////////////////////////////////////////////////////////////////
function setup ()
{
    createCanvas (800, 600);
}

function draw()
{
    background (100);
    switch (gameState)
    {
        case (1):
            fn_mainMenu();
        break;
        case (2):
            fn_results();
        break;
        default:
            fn_mainMenu();
    }
}
/////////////////////////////////////////////////////////////////////////////////////////////////////
```

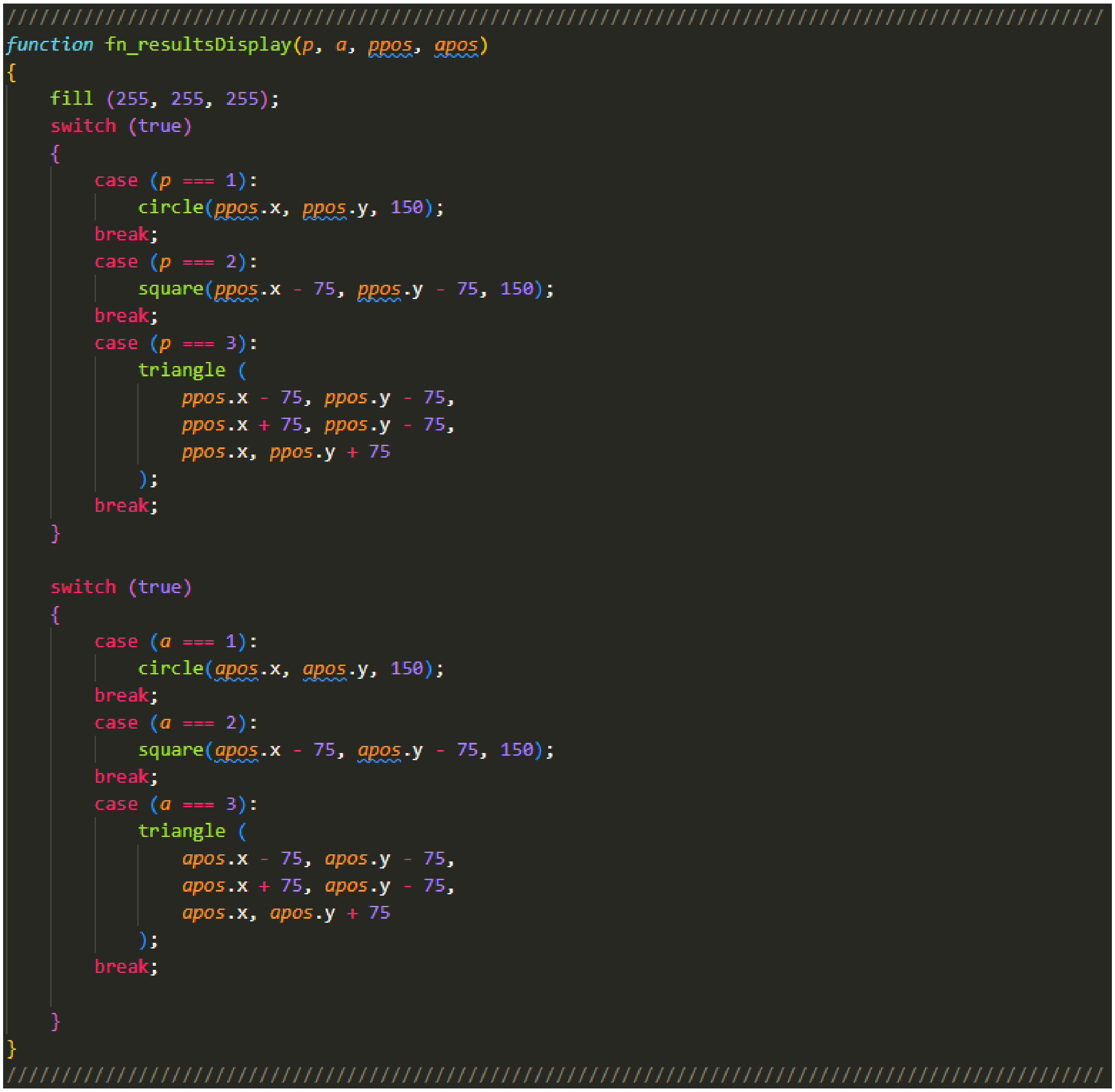
In the mt_onClick methods, gameState would be set to 2, within the draw function is a switch statement that runs every frame checking the value of gameState and changing the current running function based on it.



```
////////////////////////////////////////////////////////////////////////////////////////////////////
function fn_results()
{
    clicked= false;
    aichoice = Math.floor(Math.random() * 3) + 1;

    let pRePos = createVector (133, 300);
    let aiPrePos = createVector (665, 300);

    fill(255);
    textSize(50);
    textAlign (CENTER, CENTER);

    switch (true)
    {
        case (playerChoice === 1 && aichoice === 3):
            text ('VS', 0, 200, 800, 200);
            text ('Player Wins!', 0, 400, 800, 200);
        break;
        case (playerChoice === 2 && aichoice === 1):
            text ('VS', 0, 200, 800, 200);
            text ('Player Wins!', 0, 400, 800, 200);
        break;
        case (playerChoice === 3 && aichoice === 2):
            text ('VS', 0, 200, 800, 200);
            text ('Player Wins!', 0, 400, 800, 200);
        break;
        case (playerChoice === aichoice):

            text ('VS', 0, 200, 800, 200);
            text ('Its a Tie!', 0, 400, 800, 200);
        break;
        default:
            text ('VS', 0, 200, 800, 200);
            text ('Player Loses', 0, 400, 800, 200);
        break;
    }

    fn_resultsDisplay(playerChoice, aichoice, pRePos, aiPrePos);

    text ('Press F5 to Play Again!', 0, 50, 800, 200);

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

The function fn_results is the second game state. The variable clicked is set to false, again a feature that is deprecated, and the aichoice is calculated using a floored Math.random function returning a random value between 1 and 3, inclusive of both. pRePos and aiPrePos are declared assigned vector values of 133, 300 and 665, 300 respectively, the relevance of these numbers is explained below. The fill is changed to 255 or white, the text size to 50 and the text alignment is centered. A switch statement that always runs check the playerChoice against the aichoice. 1 is rock, 2 is paper and 3 is scissors, the first three cases account for player win scenarios, and print out text saying, ‘VS’ positioned in the middle of the canvas and ‘Player Wins!’ near the bottom of the canvas.

The fourth case is that of a draw, and in addition to the 'VS' text prints 'It's a Tie!'. The default case is that of a player loss and in addition to the 'VS' text prints 'Player Loses!'. The function fn_resultsDisplay is called and takes the values playerChoice, aichoice, pRePos, aiPrePos. The text 'Press F5 to Play Again!' is displayed near the top of the screen.



```
////////////////////////////////////////////////////////////////////////////////////////////////////
function fn_resultsDisplay(p, a, ppos, apos)
{
    fill (255, 255, 255);
    switch (true)
    {
        case (p === 1):
            circle(ppos.x, ppos.y, 150);
        break;
        case (p === 2):
            square(ppos.x - 75, ppos.y - 75, 150);
        break;
        case (p === 3):
            triangle (
                ppos.x - 75, ppos.y - 75,
                ppos.x + 75, ppos.y - 75,
                ppos.x, ppos.y + 75
            );
        break;
    }

    switch (true)
    {
        case (a === 1):
            circle(apos.x, apos.y, 150);
        break;
        case (a === 2):
            square(apos.x - 75, apos.y - 75, 150);
        break;
        case (a === 3):
            triangle (
                apos.x - 75, apos.y - 75,
                apos.x + 75, apos.y - 75,
                apos.x, apos.y + 75
            );
        break;

    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

The function fn_resultsDisplay uses the values passed to it to display the player and ai selected icons to the canvas, after setting the fill to white two switch statements are run, they check the values of p and a, or the playerChoice and aichoice, and then draws the appropriate shape using the pRePos vector and the aiPrePos vector for the x and y positions of the shape.