Tutorial 3

Author: Kia Kalani

Contributors: Ali Lezzaik, Alireza Teimoori

Introduction

In this tutorial the main focus would be on rectangles collision detection and by the end of this tutorial you would be able to make a tic tac toe game in p5js. For making tutorial more compact there would be no 'X' or 'O' in this game. Instead the color of the rectangles would change on click depending on the player's turn.

Rectangles collision detection

Logic

Each rectangle has four components in itself; a x position and a y position as well as width and height. For finding the rectangles' collision we would have to embrace the way that rectangles would collide in terms of their main components. In terms of the x position, two rectangles would collide when the x position of the second rectangle is more than or equal to the x position of the first one while being less than or equal the x position plus the width of the second rectangle. The same exact logic applies to the y components of both of the rectangles: [Insert the image here.]

In code

For applying this logic in code, we need a function with the four components of two rectangles as parameters and exactly check for the conditions provided in the logic. Therefore there would total of 8 parameters for rectangles collision detection. The code for detecting collision between two rectangles can be written as below:

```
function rectangleCollidedWithRectangle(x, y, width, height, x2, y2,
width2, height2) {
    return x <= x2 + width2 && x2 <= x + width && y <= y2 + height2 && y2
<= y + height;
}
```

Note: In case we want to check for collision between a point and a rectangle we just need to provide the components of the first rectangle and provide the x and y position that we want to check the collision for with the width and height set to 0 for the second element. That is because a point is theoretically a rectangle with width and height of 0.

Usages of the rectangle collision

When making various graphical projects, the rectangle collision is probably the most critical type of collision that should be checked for. In case of making a program from scratch without any button elements, checking for this type of collision would be necessary for knowing when a button gets pressed. There are

many applications to this type of collision detection excluding that specific case. For instance, one of usages for this type of collision can be in making a simple tic tac toe game.

Making a simple Tic Tac Toe Game in p5js

For making a simple tic tac toe game we need to setup a normal p5js project as demonstrated in the first and second tutorial. For this game, we are going to be using a 2d array of rectangle components with an indicator that specifies what kind of occupation the rectangle has. Another element that is necessary for this game is a boolean variable that specifies which players' turn it is. As always after defining the width and height of the canvas we need to define the rectangles and the turn variables and initialize the rectangles:

```
const rectangles = [];

let turn = true;

function setRectangles() {
    for (let r = 0; r < 3; r++) {
        rectangles.push([]);
        for (let c = 0; c < 3; c++) {
            rectangles[r].push({ x: c * (WIDTH / 3), y: r * (HEIGHT / 3),
width: WIDTH / 3, height: HEIGHT / 3, content: 0 });
        }
    }
}
```

In this case, the position of the rectangles are being calculated by a simple mathematical calculation where we divide the width and height by 3 and multiply them by the index of the row and column accordingly. Of course the width and height of each rectangle would be the width and height of the canvas divided by 3 respectively. In this case the variable content represents what occupation the rectangle has; i.e. if whether it is 'X' or 'O' or it is still an empty spot.

Mouse events

Inside p5js there are several built in methods for handling various events and they would be invoked when some sort of input event takes place. For instance for making this game we are going to be using the "mousePressed" event. For making certain events to take place when mouse is pressed, we would have to call them inside this built in method. We have several events to handle when the mouse is being pressed. The first one is to make a move by the current turn and then change the turn. For doing so, we would have to loop through all of the rectangle objects that we have defined earlier on and check for collision detection between a rectangle and mouse position. If there was a collision and the content of the rectangle is still blank, we are going to change the content according to the turn and then change the turn. Therefore in terms of code, it should look something similar to the following:

```
function makeAmove() {
    for (let r = 0; r < rectangles.length; r++) {
        for (let c = 0; c < rectangles[r].length; c++) {
            let curr = rectangles[r][c];
```

```
            if (rectangleCollidedWithRectangle(curr.x, curr.y, curr.width,
curr.height, mouseX, mouseY, 0, 0)) {
                if (curr.content == 0) {
                    if (turn) {
                        rectangles[r][c].content = 1;
                    } else rectangles[r][c].content = 2;

                    if (turn) {
                        turn = false;
                    } else turn = true;
                    return;
                }

            }
        }
    }
}
```

Resetting the game

By having this portion of the code, we are officially done with making the moves; however there are two elements remaining that we would have to check for; whether someone won the game or the game was a tie. In either of the cases we are going to restart the game for the players. For doing so, we just reset the content of all rectangles to value 0 which represents the empty position as well resetting the 'turn' variable. In terms of code, the following can be an example for what to expect:

```
function resetGame() {
    for (let r = 0; r < rectangles.length; r++) {
        for (let c = 0; c < rectangles[r].length; c++) {
            rectangles[r][c].content = 0;
        }
    }
    turn = true;
}
```

Win conditions

We are going to start by checking for the win conditions. There are three ways a player can win the game. They can either win diagonally, horizontally or vertically. Checking for the win conditions horizontally and vertically can both be done in a loop. We just have to have a loop from 0 to 3 and check to see if the elements of row have the same components for vertical and column components for horizontal win conditions. For diagonal win conditions, checking them manually without looping would be simpler as there are only two cases for it. Therefore in terms of code, it would end up looking similar to the following:

```
function checkWin() {
    for (let r = 0; r < rectangles.length; r++) {
        if (rectangles[r][0].content == rectangles[r][1].content &&
```

```
rectangles[r][1].content == rectangles[r][2].content && rectangles[r]
[0].content != 0) {
            window.alert("Game over");
            resetGame();
            return;
        } else if (rectangles[0][r].content == rectangles[1][r].content &&
rectangles[0][r].content == rectangles[2][r].content && rectangles[0]
[r].content != 0) {
            window.alert("Game over");
            resetGame();
            return;
        }
    }
    if (rectangles[0][0].content == rectangles[1][1].content &&
rectangles[1][1].content == rectangles[2][2].content && rectangles[0]
[0].content != 0) {
        window.alert("Game over");
        resetGame();
        return;
    } else if (rectangles[0][2].content == rectangles[1][1].content &&
rectangles[1][1].content == rectangles[2][0].content && rectangles[1]
[1].content != 0) {
        window.alert("Game over");
        resetGame();
    }

}
```

Checking the tie condition

Finally, we have to check for tie. For doing so, we only need to loop through all of the rectangles and check to see if any of them has a blank spot. If they do, then the game would still continue. In case of win with the last move, the check win conditions method would be called before this method. Therefore, that would be literally the only that we need to check for. In terms of code, it would look similar to this:

```
function checkTie() {
    for (let r = 0; r < rectangles.length; r++) {
        for (let c = 0; c < rectangles[r].length; c++) {
            if (rectangles[r][c].content == 0) {
                return;
            }
        }
    }
    window.alert("The game was a tie");
    resetGame();
}
```

Writing the mousePressed function

Finally after having all of the logic related components, we can put them all together inside the mousePressed function. It would end up looking something similar to this:

```
function mousePressed() {
    makeAmove();
    checkWin();
    checkTie();
}
```

Drawing the components to the Canvas

Drawing the components is the main part of making this game since they have to be visually demonstrated to the players. After setting the background color, the most important thing is to draw the rectangles themselves. For showing the different moves, we will be using the colors red and blue. In case that the content value is 1, we would change the color to red and if it is 2 we would change the color to blue. Otherwise it would stay as white. And by the end inside the loop for drawing the rectangles, we would use the built in 'rect(x: , y: , width: , height:)' method for drawing the rectangles with their main components. Therefore by the end the code would look similar to the following:

```
function draw() {
    background(100);
    for (let r = 0; r < rectangles.length; r++) {
        for (let c = 0; c < rectangles[r].length; c++) {
            let cur = rectangles[r][c];
            if (cur.content == 0) {
                fill(255, 255, 255);
            } else if (cur.content == 1) {
                fill(255, 0, 0);
            } else fill(0, 0, 255);
            rect(cur.x, cur.y, cur.width, cur.height);
        }
    }
}
```

That sums up the third tutorial.

For getting access to the source code, please visit: