Tutorial 4

Author: Kia Kalani
Contributors: Ali Lezzaik, Alireza Teimoori

Introduction

In this tutorial the main focus would be on using the object oriented implementation of the game components in JavaScript in order to make a more meaningful and neat code in terms of readability and functionality. Finally by the end of this tutorial, you should be able to make a game menu and giving the player different options to choose from. Also being able to change the current menu according to the selected menu. Note that the last three sections of this tutorial series would be focusing on making a space invader game in `Javascript` using `p5js`.

Why Object Oriented Approach?

Using objectification approach has a great application in making games. This is due to a couple of reasons:

1.Game components are literally an object in real life therefore, using object oriented approach would be beneficial.

2.Utilizing design patterns would make it easier to write codes with similar structures but different contents.

3.The most important reason is that coding the stages related to the game objects would make them look more understandable.

The Template for the menu

While making games, every component has three behaviours in general; they have to be rendered in order to show the contents to the user and they should be updated. Some objects would have to behave according to the specific action in terms of the input that the

users are providing. Therefore in abstraction, every single object would have to be acting according to these methods. However, it is also important to remember that the handling part can be by the keyboard key being pressed or released or by mouse being pressed or released etc.

In terms of code, the idea behind the menus can look similar to the following:

```
class Menu {
    constructor() {

    }
    handleKeyPressed() {

    }
    handleKeyReleasedd() {

    }
    handleMousePressed() {

    }
    render() {

    }
    update() {

    }
}
```

Note that not all of the handling elements are presented here. That is because we would be only using the `mousePressed()` and keyboard events.

Thus, we have the main idea behind making the menu; However, we still have to use it inside our `sketch.js` file as we presented the idea behind it in the previous tutorials.

For doing so, we are going to create a variable of the class menu and call the render and update inside our `draw()` method since it is theoretically a loop that keeps updating the components of the game.

We would also adjust the components of handle methods with the built in methods of the `p5js` library for handling. Finally we are looking to change the scenes when certain events take place and therefore a method would be required to change the current scene to another one.

By the end, the code for `sketch.js` file would look similar to this:

```
let currentScene = new Menu();

function changeCurrentScene(scene) {
    currentScene = scene;
}

function setup() {
    createCanvas(MENUWIDTH,MENUHEIGHT);
}

function keyReleased() {
    currentScene.handleKeyReleased();
}

function keyPressed() {
    currentScene.handleKeyPressed();
}

function mousePressed(){
    currentScene.handleMousePressed();
}

function draw() {
    currentScene.render();
    currentScene.update();
}
```

Now by having that done, we can get started with making the menus related to the game.

Game Menus

Template Menu

We are going to start with creating a main menu. In order to give all of the menus a space invader feeling, we are going to put some stars on the background.
For doing so, we need an array of certain elements with x and y position assigned randomly inside the canvas.

For doing so, let us first create a method that would provide the random number within the provided boundaries.

```
function nextInt(start, end) {
    return Math.floor(Math.random()*(end-1))+start;
}
```

Now inside the constructor of the menu class we would create the stars by an idea similar to the following:

```
this.stars = [];
for (let i = 0; i < nextInt(100, 200); i++) {
    this.stars.push({x: nextInt(0, MENUWIDTH), y: nextInt(0, MENUHEIGHT)});
}
```

Also we are going to create a class that would take care of the operations related to collision detection. For more information about this part, please check out tutorial 2 and 3.

Additionally, we would modify the `render()` method to demonstrate the circles with a random diameter from 1 to 4 as well setting the background color. The code would look similar to the following:

```
background(5);
for (let i = 0; i < this.stars.length; i++) {
    circle(this.stars[i].x, this.stars[i].y, nextInt(1,4));
}
```

Now by opening the `index.html` file you would notice there are stars in the background!

Now that the menu template got the theme of the game, we move onto making the main menu.

Main Menu

First we need to create a main menu class that inherits the components of the menu class. This can simply be done like the following:

```
class MainMenu extends Menu {

}
```

And we would adjust the value of the current scene to show a main menu rather than our template menu.

As for the constructor we need to have a `super()` call in order to make sure the components of the stars would also exist in this menu.

Additionally, in every main menu we have a bunch of options with a currently selected option; so we would implement this concept as well.

In this case, in terms of code, it would look like the following:

```
super();
this.options = ["Start Game", "Instructions", "About This Workshop"];
this.selectedPosition = 0;
```

Now coming to the render portion of the menu, we need to demonstrate the content of the project with an appropriate name.

In this case we would be using `Space Invader Game`.

The method for writing a text to the display is `text(content: , x: , y: )` which content represents the content of the string we are writing and x and y represent the position we are writing the content to.

Additionally the method `textSize(size: )` would set the size of the text that we are writing. For making the code more organized, we would create a method for rendering the title and call it `renderMainText()`. The writing methodology may differ according to personal preference.

Another method that we need for rendering the components is a method that would show the options to the users. This method would just render the options and selected position. In terms of code, a way of implementing it is the following:

```
renderOptions() {
        for (let i = 0; i < this.options.length; i++) {
            let fillcolor = 'blue';
            if (i == this.selectedPosition) {
                fillcolor = 'red';
            }
            textSize(25);
            fill(fillcolor);
            text(this.options[i], 230, 250 + (i * 50));
        }
}
```

Now that we have all of the components for rendering we are going to go ahead and write the `render()` for the main menu. First of all, we need to call `super.render()` to make sure the stars would be shown to the user. Afterwards other components would come. The code would end up like this:

```
    render() {
        super.render();
        this.renderMainText();
        this.renderOptions();
    }
```

Moving on from the rendering portion, we would reach the handling portion. The events associated with this menu is just moving to another menu by the keyboard and mouse events.

Therefore, we would want the exact the same code for mouse pressed and key released.

So we create a method called `doActionBySelectedIndex()` and indicate what events should take place by selecting each option. The event would be just moving from the scenes.

The code would look similar to the following:

```
doActionBySelectedIndex() {
    switch (this.selectedPosition) {
        case 0:
            changeCurrentScene(new GamePage());
            break;
        case 1:
            changeCurrentScene(new Instructions());
            break;
        case 2:
            changeCurrentScene(new About());
            break;
    }
}
```

This method would be essentially called inside both mouse and keyboard events. However, we still did not figure out a way to move from options.

We can use the built in `key` variable that would indicate which key just had an event. Therefore, if that value indicates up or right, we would decrease the selected position by 1 and if the value of it presents right or down, we would add one to the selected position. Of course this would case out of bounds if we avoid handling it.

Therefore, we make another method for handling out of bounds. The code for that would look like the following:

```
checkKeyBoundaries() {
    if (this.selectedPosition < 0) {
        this.selectedPosition = this.options.length - 1;
    } else if (this.selectedPosition == this.options.length) {
        this.selectedPosition = 0;
    }
}
```

Finally by having that we can put all the logic together and write the method for handling the key released events:

```
handleKeyReleased() {
    super.handleKeyReleased();
    switch (key) {
        case "ArrowUp":
            this.selectedPosition -= 1;
            break;
        case "ArrowDown":
            this.selectedPosition += 1;
            break;
        case "ArrowLeft":
            this.selectedPosition -= 1;
            break;
        case "ArrowRight":
            this.selectedPosition += 1;
            break;
        case "Enter":
            this.doActionBySelectedIndex();
            break;
    }
    this.checkKeyBoundaries();
}
```

Finally, for demonstrating the application of the collision detection, we would use the mouse for moving the selected option as well. For doing so, we would create a method called `mouseHitOption()` which would move the selected option. We just need to multiply the size of the texts that we are using with the total characters used. Therefore, we would end up with a code similar to the following:

```
    mouseHitOption() {
        for (let i = 0; i < this.options.length; i++) {
            let x = 230;
            let y = 230 + ((i) * 50);
            let width = 25 * this.options[i].length;
            let height = 25;
            if (this.collisionDetector.collidedRectWithPoint(x, y, width,
height, mouseX, mouseY)) {
                this.selectedPosition = i;
                break;
            }
        }
    }
```

The only thing that is remaining is to modify the `update()` method. We first call the `super.update()` to ensure the stars would keep moving and afterwards, we make a call to the `mouseHitOption()`.

By having that done, the basic structure of the menus are done once all of the menus that are indicated inside the main menu are created and have inherited the main menu.

This sums up the fourth tutorial!

For gaining access to the source code please visit: https://github.com/kiakalani/P5JSTutorial/tree/main/tutorial4/code