Tutorial 5

Author: Kia Kalani
Contributors: Ali Lezzaik, Alireza Teimoori

Introduction:

The main focus of this tutorial is to create a player object that would be capable of moving around inside the scene. Note that this tutorial would not be including the logic behind the bullets and the mobs coming from the other side. Instead, the main focus would be on loading the image and setting up an express server for the game.

Why Express JS server?

The main reason for utilizing this server is because loading components for images and related materials are dependant on using a server.

Otherwise when trying to load images there would be an error indicating that the image we are trying to find does not exist. The way of getting around this issue is setting up a server.

Additionally, having an express server would be useful in case that we are trying to add more functionalities to the server and for instance making a multiplayer game etc.

The last and final reason is due to express being simple to use and easy to setup.

How to set up an Express JS Server:

Setting up an express server is really simple. For doing so, we need to browse to the directory that the project would be included in and open it in terminal.

Inside the terminal, we simply type npm init -y.
Doing so would assume the default cases for generating the package.json file.

Express.js

By having that now, we can go ahead and start installing the required packages. We need three packages for this tutorial and one of them is optional but it is beneficial to use it:

1. The first one is `express` package itself which would be responsible for serving the components that we have made.
2. The second package is path which would be responsible for loading static components inside the server.
3. The last and the optional package is nodemon which would be responsible for updating the server by any changes made to it.

Therefore, we would type the script as the following:

```
npm install --save express path nodemon
```

After doing so, we need to setup the basic structure of the code. For that we are going to make the index.js file in the root directory of the project.

After having that setup, we are going to modify the package.json file. After finding the "debug" option, we would put a ',' and then go to the next line and type:

```
"start": "nodemon index.js"
```

Now we get to actually make the server itself. For doing so, we go to the index.js file and start typing the server components.

We needed two components in order to serve the static webpage as you recall.

There were express and path. Now in order to require the packages we simply create a variable and set them equal to requiring the packages we have installed.

This procedure would look as the following:

```
const express = require("express");
const path = require("path");
```

Then the same way we setup an express application, we type:


```
const application = express();
```

Now in order to use express statically, we would have to use the path package that we required beforehand which would be done like the following:

```
application.use(express.static(path.join(__dirname, "public")));
```

Now we need to define a port that the application would be running on. For doing so, we just type the following:

```
const port = process.env.PORT || 4000;
```

Where 4000 would be the port that we would be running our server on. Feel free to adjust the value of it to your preferred port.

Now we move onto loading the components statically. For doing so, we are going to copy and paste what we currently have from tutorial 4 inside the public directory. Then, we can go back to making the server inside the index.js file. We would want to get the index.html file that we have created before.

In terms of code, we would have to make a get call to our application in order to get the components off the static page by going to the main directory. In order to send the static page as a response, we would need a code similar to the following:

```
application.get("/", function(request, response) {
   response.sendFile(path.join(__dirname, "./public", "index.html"));
});
```

Essentially by typing sendFile method the server would transfer the contents of the static site to the user.

Now finally by getting the components of the webpage, we need to make the server actually go live on the port we have defined earlier on as well as indicator that says the server is running on the given port. For doing so, we can write a code as following:

```
application.listen(port, function() {
    console.log("Server is up and running at port", port);
});
```

By having this part over, we are essentially done with setting up the server for our p5js project.

Now as indicated before, for being able to see the components that we have had so far, we make sure that the components are copied inside the index.html file.

Additionally, the same way that we made a template for menus, we are now going to make a template object that has exactly the same method. The exception is that the components of each object would be loaded within the menu that we are using it for.

Additionally, for getting started, you can try to replace the stars that we made earlier on with a CoolStar object that follows up with the same logic as the GameObject.

Note that the name Object is already used and therefore is not available for usage and hence we are going to stick with GameObject.

Note that each object in the game has an x and a y coordinate. Therefore, we would utilize the constructor for initiating the positions.

Now we are going to make the CoolStar by extending the GameObject for practicing the implementation of the GameObject.

After doing so, we are now going to make the player and load an image for it. For doing after setting the inheritance, we are going to initially load the image. This process can be done by using the built in method loadImage() and putting the relative to public directory as the parameter.

For instance in the case of this code, we are going to write it as the following:

```
this.image = loadImage("img/ship.png");
```

Now, the player also has the width and height and score components. They can be set through the constructor.

Additionally, we would need an accelerator in order to make the player moving smoothly and not necessarily dependent on keyPressed events.

Moving on from setting the normal components of the game, we are going to draw the player according to the x, y, width, and height components of the player. For doing so, we are going to move on to the `render` method.

For loading the image with the specified components, we are going to use the `image` built in method.

Loading the image is as simple as the following code:

```
image(this.image, this.x, this.y, this.width, this.height);
```

Finally, updating the player is as simple as saying:

```
this.x += this.accelerator;
```

Finally we want to make sure the player does not go out of bounds, so, we would update the player position with an if statement that it can't go more than screen width subtracted by its height or lower than 0 before updating the x position.
Now moving on to handling section of the code, we are going to make the key pressed and keyReleased. In case of button a being pressed, the accelerator would equal to -1 and if it is d the accelerator would equal to 1. In case of these keys being released the values would be set to 0 so the player would stop moving.

These events can be done within the methods and using the built in key value that indicates which key is currently being pressed.

By having the player moving around now, we come to the end of this tutorial!

For accessing to the source code please refer to: https://github.com/kiakalani/P5JSTutorial/tree/main/tutorial5/code