**Player.js, Game.js**

We will add information about players into the database.

Then we need the information about the game state and the number of players who joined.

We will make a 2-player game.

When the **game state** is 0 (**WAIT**), we want the players to see the login form where they register their names as players.

When the number of registered players reaches 2, we want the **game state** to become 1 (**PLAY**).

When the game state changes to 1, we would like the game to start.

We need a function to get the game state from the database. As 2 players join and the player count will increase, we will change the game state from 0 to 1(play).

while synchronising a ball game we learned about .ref() and .on() methods.

we are using .ref() to pass the location of the ‘gameState’ field of the database and using .on() we are reading the value of the ‘gameState’ field and saving it to the global variable gameState

Here instead of writing a separate function to read the value, we continued writing in .on() itself. This allows us to bind the function to read ‘gameState’ with getState() only.

We are first calling getState(), and then start() which in turn creates an object for the Form and Player class.

from Player class we will store and retrieve player information to and from the database

in the constructor(), we will be creating various properties for each player

Properties are created to save:

this.name - to save the name of the player

this.index - to give a unique id to each player,

this.positionX & this.positionY - to store the x & y position of each player.

two more methods in player.js. getCount() & updateCount().

We have created these methods to update the field ‘playerCount’ in the database, which we created in the last class, and also read the number of the ‘playerCount’ from the database and save it in the Global variable to use in the code. In order to read or write in the database, we use .ref() - to give the location of the field in the database.

Then we use:

.on() - to keep listening to the changes that happen in the ‘playerCount’ field of the database.

.val() - to copy the value from the database to the global variable of the code.

.update - to store value from global variable to the database field ‘playerCount’.

**“/” is used on updateCount to refer to the root directory.**

To give the position to each player and add their properties to the database, we will write functions : .ref() and .set()

We have two players in the game; we want the first player to stay on the left side of the screen and the 2nd to be placed on the right side of the screen before the race starts so let's give them the x & y position accordingly.

Now we will add all properties of players in the database. We will also create a player’s field on a real-time basis. We can do this using string concatenation. If the ‘playerCount’ is 1, we create a database entry for player1 and we set the name for it, and so on.

→ First we use .ref() to give a location in the database.

--> databaseReference.set() will create and save the database reference.

With this, we need to modify handleMousePressed() in form.js.

An object of the player class shall call an addPlayer() method to add information in the database and an updateCount() to increase the ‘playerCount’ field in the database.

We will increase the ‘playerCount’; initially, the ‘playerCount’ is 0 as we kept it 0 in the database. When the first player joins, it will be increased to 1, and when the second player joins, it will increment to 2. We need to update the count to the database simultaneously, hence we are calling player.updateCount(playerCount).

this.input.value() method is used to read the user input in the game name of the player. The player’s name will be assigned to the property player.name. Then we assign the playerCount to player.index which is used as a unique id for the players. So player.index will be 1 for the first player and 2 for the second player. This player.index property will be very useful once the race starts.

Lastly, we call addPlayer(). This method will store all the data in the database.

In a two-player game, once both players have joined ‘gameState’ should change to 1

We will write a condition in sketch.js to update the gameState when playerCount is equal to 2. For that, we will have to create similar methods in Game.js to those we created in player.js for the count.

inside the function draw(), if conditions are given to update State to 1 when ‘playerCount’ = 2 and call the play() method when ‘gameState’ is 1.

create play() in game.js

To show track and race cars

We will use image() to show the terrain. Preload the images of track and cars.

create two car sprites for both the cars at the start() of Game.js and add respective images to it. And then create an array of cars.

create the start() method, where we will create the car's sprites in game.js. We will create 2 cars using the createSprite() function from the p5.play library. We want the sprite to be close to the center of the screen in horizontal (x direction) so we are keeping the x position value as width/2 -50. In the y direction we will keep the car very close to the bottom of the screen hence we write height-100. We will not specify the width and height of the sprite in this function because we want to add the image to the sprite using the addImage() function. The sprite will take the size of the image, which is very large so we will scale the sprite down using car1.scale() function.

The same process we will follow to create the car2. In the end, we will create an array named cars and put car1 and car2 in that array.

To fetch(get) player’s details from the database

write a function to get all the players' info. This function will not be attached to any particular player. We can declare it as a static function. Static functions are not attached to each object of the class.

Static functions are those common functions that are called by the class themselves rather than by objects of the class. We use the 'static' keyword before the function to make it a static function.

The player's data will be stored as JSON - since the Firebase database structure is of JSON type.

handleElements() This function helps us to hide form once the game is in play mode. So now create a method play() in Game.js

We want to display terrain only once player’s information is received hence we will use if condition here.

play() method and inside this method, the first thing we want to do is position the title image on the canvas, which we will do using handleElements() function after that we will get the players info from the database and that we will do using Player.getPlayersInfo() function. This function is defined in the Player.js file and it gets the information of all the players from the database. We write a condition if (allPlayers !==undefined)

When we defined the variable allPlayers we have not assigned any value to it. This means at the start of the code it will be undefined; it will only have the values received from the database. If this condition is true then we will display the track image on the canvas using the image() method. While displaying the image we will keep the x position as 0 and y as (-height\*5); this will create the track outside the canvas, because we don't want to show the complete track at once we will show the track as the player will move the car. For the width of the track, set as width and for the height keep it as height\*6.