**Game Camera:**

It allows players to follow the game characters beyond the screen.

We move the car forwards using the UP\_ARROW key

We will move the car when the game is in **play** mode

we have 3 objects, **form**, **player** & **game**. Our game execution code is in game.js. We also have the track.

We need to set the position of the players (cars) based on the position from the database. We will use player.getDistance() to get the positions of the players (cars) from the database.

Create a variable index as var index = 0. We will use the for-in loop to extract the positions of the players from the **allPlayers** object. For that, we will use the for-in loop.

This for-in loop has a different syntax. This is called a for-in statement; using this we look for the values inside a javascript object.

Syntax: rules to write code

Syntax

for (variable in object) { statement }

eg. for (var plr in allPlayers) { }

In the first iteration, plr value will be player1 and next iteration, plr value will be player2.

The for-in loop runs for a number of elements in the object. Here allPlayers variables contain the information of all the players, but we need to extract the info of each player so that we can set their position. So we write

for(var plr in allPlayers).

Inside the for loop, we need to manually increase the index by one. We need to define 2 variables x and y to store the positions.

To access the data from the allPlayers object we write the name of the **object[plr].positionX.** This value will go in var x.

The same process we will do for the y-direction but we need to subtract the y position from the height because we want to place the player at the bottom of the screen.

Then we need to set these positions to the position of the car's sprite as well. **‘Cars’ is an array** containing 2 car sprites; we need to set the position to both of them. The index value is 1 now, but we need to keep it 0 because the first element of the array is at 0. That's why we need to subtract 1 from the index. Hence using cars[index-1] we assign x and y positions to each car.

The next step is to make the player move by **keypress**, which we are implementing by using the **keyIsDown**() function of p5.js.

To make sure our code is structured we will create a separate function to add controls to the car. let us name it as **handlePlayerControls**(). because it is defined in the same file Game.js we can call it as this.handlePlayerControls().

Call this.handlePlayerControls() after for-in loop of play()

Create a condition to move the car upwards inside handlePlayerControl() of game.js.

The condition checks for the keypress event and once the key is pressed we are changing the y position of the player and we are calling the player.update() function so that player’s position will be updated in the database. We have called the update method here but we need to write this method in the class. First, create the update() method; in this, we are using playerIndex to update the x and y position of the player

In the **update**() method, we are referring to player1 & player2 to update their respective position on the canvas.

We use .**ref**() to refer to location in the database. .**update**() is used to update an existing field in the database with a new value. Here we are updating the x & y positions of the car.

**Retrieve the distance** from the database so that the other player knows where another car is.

create the **getDistance** method inside the player.js file. Here first we are getting the position values from the database using **database.ref(**). Then we are storing the data in the data variable.

**Data** coming **from** the **database** is in the form of a **JavaScript object(JSON)** so you need to extract the x and y position of the car. Create 2 variables this.positionX and this.positionY and we assign these to the x and y position values from the database.

Here we are using properties created in the constructor() to update the distance. call this function In the play() of game.js. however, in order to make sure cars are always placed at the right position when the game starts, we will add it in the handlemousePressed() function of Form.js.

To **highlight the active player** in a window draw a small circle below the playerCar

we draw the circle Inside play() in Game class as this is the function which is called when the game is in the ‘play’ state.

identifying the current player in the game by matching index with player.index inside the condition **- if(index===player.index).**

give instructions to add the circle at the x and y position of the car using **ellipse**() when the condition is true.

also use **fill**() to make an identifier red in **color** and **stroke**() to give its **border** **width**.

**issue are we facing:**

When we keep pressing the up arrow till the car goes out of the screen and we are not able to see the cars. All the players see the same thing in the game. This is where the concept of a Game Camera comes in.

The **game camera** allows us to change how and from where we are viewing the game. We want the Game Camera to be focused on each player's car. We can set the camera position in the game differently for each player.

Here we are setting the camera position x and position y according to the player’s car position. We will restrict the movement of cars to stay on track. For that, we can keep the x position of the camera to align with the x position of a car. Our cars move forward, so we will align the position of the camera with the position of the car.

Let's reset and run the code. Make sure gameState and playCount is 0 and delete all players.