CS-PRO-C37(V3)

GAME STATES & PLAYER INFORMATION

What is our GOAL for this MODULE?

In this class, we learned how to update and access the player's information from the database. We also changed the game states based on the player count.

What did we ACHIEVE in the class TODAY?

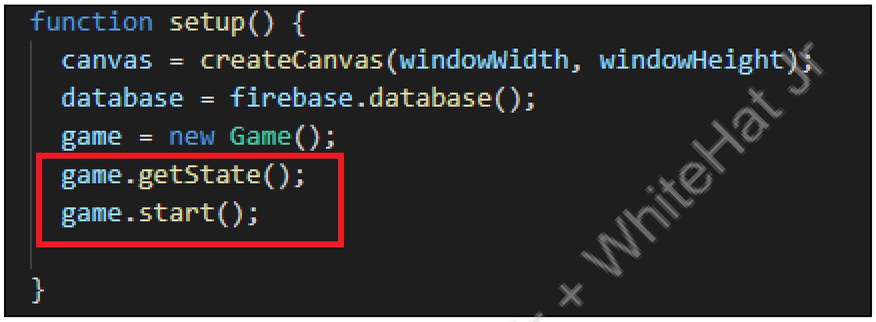
- Updated both player's details, **playerCount** & **gameState** to the database.
- Created listener to read changes from the database.
- Used the **Image()** function to show terrain for the race.

Which CONCEPTS/ CODING BLOCKS did we cover today?

- Database Queries
- Static Methods
- Receiving and interpreting JSON data from the database
- Creating an image beyond screen size.

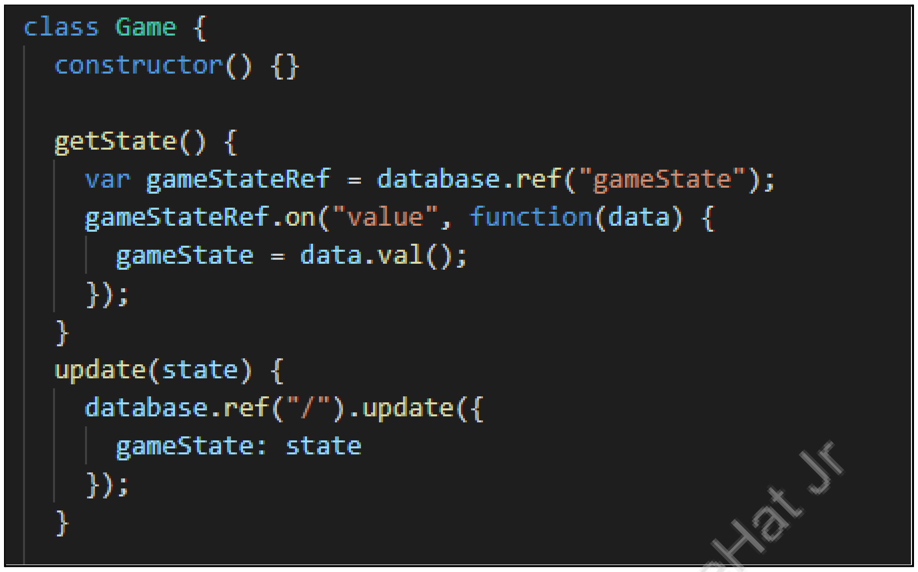
How did we DO the activities?

1. Create functions in Game.js for gameState:
    - **getState()** to read gameState values from the database.
    - **updateState()** to write the gameState value into the database.
    - Call **game.getState()** in function setup to know the current state of the game.



```
function setup() {
  canvas = createCanvas(windowWidth, windowHeight);
  database = firebase.database();
  game = new Game();
  game.getState();
  game.start();

}
```

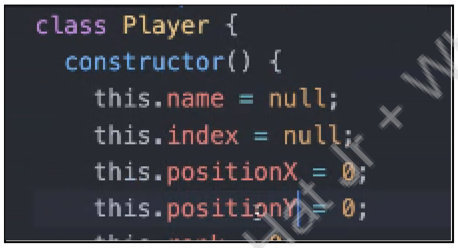
2. For database query we make use of four basic functions:
    - **.ref()** to Provide location of the database from where we need to write or read data.
    - **.on()** is a listener, which continuously check for any changes in the value of the location given in **.ref()**
    - **.val()** copies the value from database field to global variable.
    - **.update() / .set()** to update the values in database field whose location is given in **.ref()**.



```
class Game {
  constructor() {}

  getState() {
    var gameStateRef = database.ref("gameState");
    gameStateRef.on("value", function(data) {
      gameState = data.val();
    });
  }
  update(state) {
    database.ref("/").update({
      gameState: state
    });
  }
```

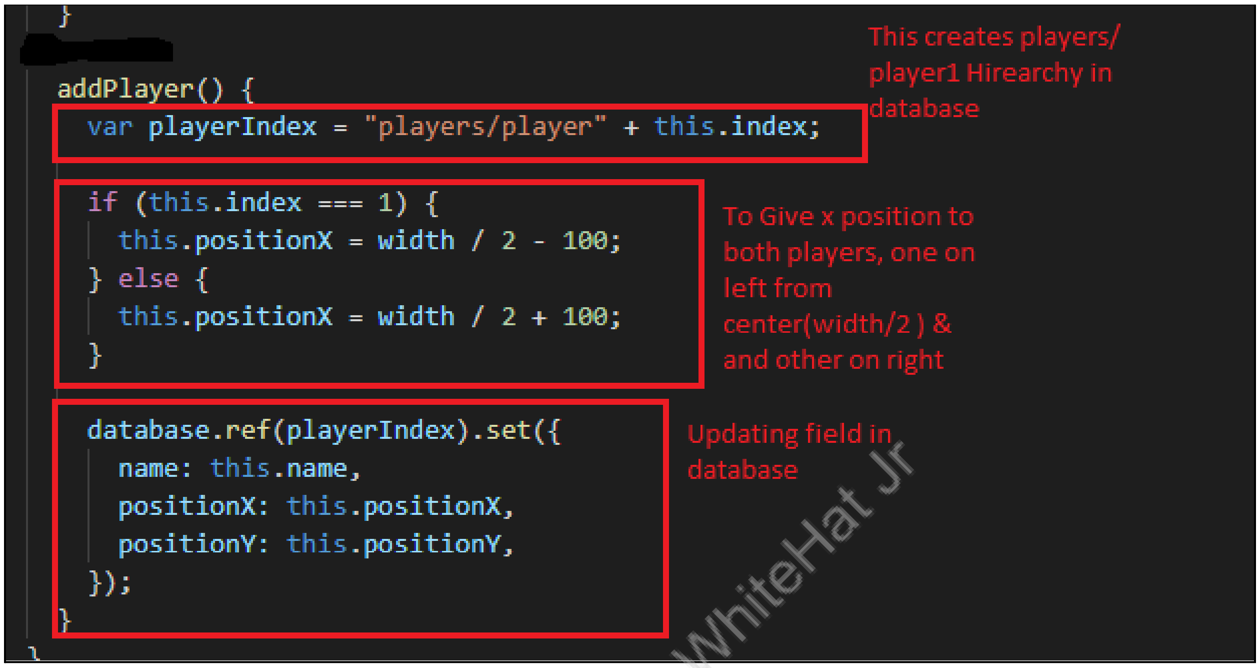
- '/ ' is the location of the root database; while updating values, we need to give the location of the parent node.

3. Use Player Class **Constructor()** to create properties of a player.



```
class Player {
  constructor() {
    this.name = null;
    this.index = null;
    this.positionX = 0;
    this.positionY = 0;
```

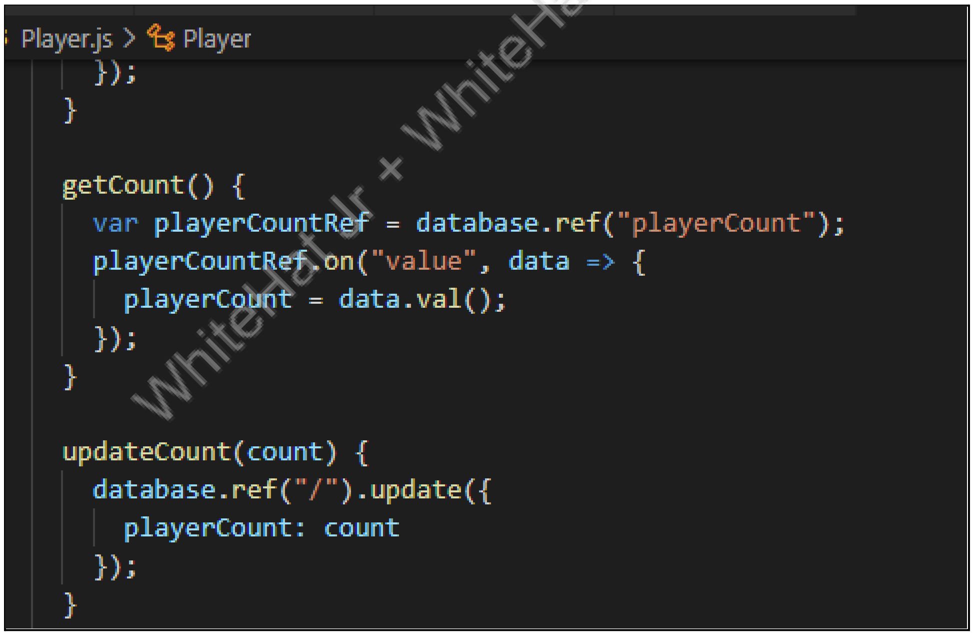
4. Write **addPlayer()** to update the player name in the database:
   - Create new entries in the database.
   - Use string concatenation. If the player index was 1, the database entry will be created as player1.



```
  }
[redacted]
  addPlayer() {
    var playerIndex = "players/player" + this.index;

    if (this.index === 1) {
      this.positionX = width / 2 - 100;
    } else {
      this.positionX = width / 2 + 100;
    }

    database.ref(playerIndex).set({
      name: this.name,
      positionX: this.positionX,
      positionY: this.positionY,
    });
  }
}
```

This creates players/ player1 Hirearchy in database
To Give x position to both players, one on left from center(width/2) & and other on right
Updating field in database

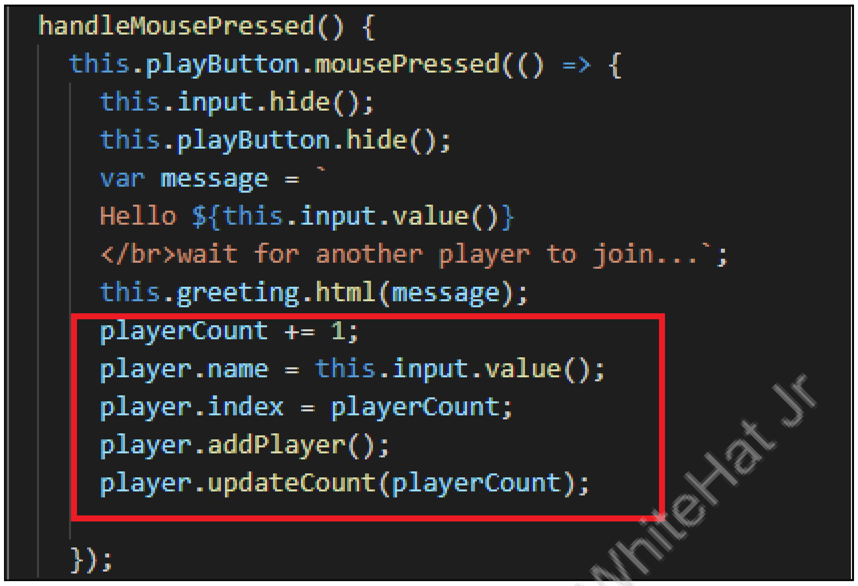
5. Write database queries to read and update **playerCount** in **player.js.**



```
    });
  }

  getCount() {
    var playerCountRef = database.ref("playerCount");
    playerCountRef.on("value", data => {
      playerCount = data.val();
    });
  }

  updateCount(count) {
    database.ref("/").update({
      playerCount: count
    });
  }
```

6. Track the number of players joined when the Play button is pressed in **form.js.**
   - Modify the function **handleMousePressed()** as below: (Shown in red box)
   - Increase **playerCount.**

- Call **updateCount()** using a player object.

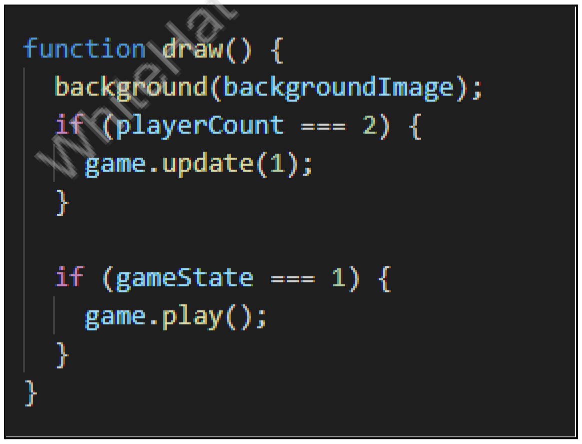


```
handleMousePressed() {
  this.playButton.mousePressed(() => {
    this.input.hide();
    this.playButton.hide();
    var message = `
    Hello ${this.input.value()}
    </br>wait for another player to join...`;
    this.greeting.html(message);
    playerCount += 1;
    player.name = this.input.value();
    player.index = playerCount;
    player.addPlayer();
    player.updateCount(playerCount);

  });
```

7. Write conditions in the **sketch. js** to update **gameState** once both players join.
    - Update the **gameState** to 1 once two players join the game.

    - Call **play()** from **game.js** when **gameState** is 1.

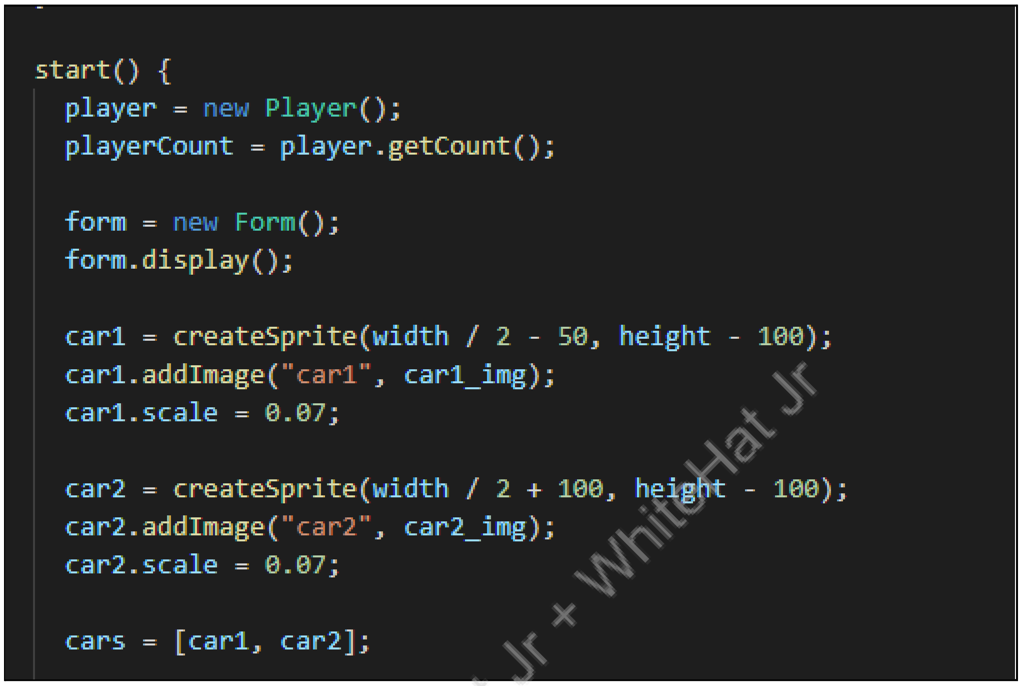
    - This method of calling a function before creating it is called **Abstract Programming.**



```
function draw() {
  background(backgroundImage);
  if (playerCount === 2) {
    game.update(1);
  }

  if (gameState === 1) {
    game.play();
  }
}
```

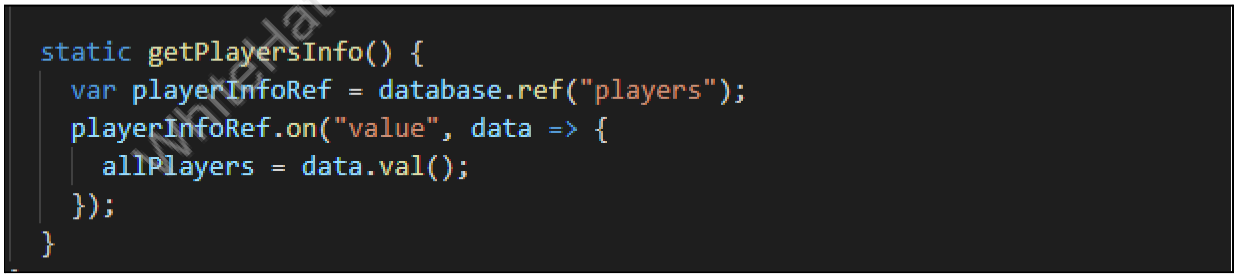
8. Create **start()** method:
    - Create sprites in **start()**.
    - Add preloaded images to both the sprites.
    - Create an array to contain both cars.

- Remember to preload Images in function **preload()** in **sketch.js.**



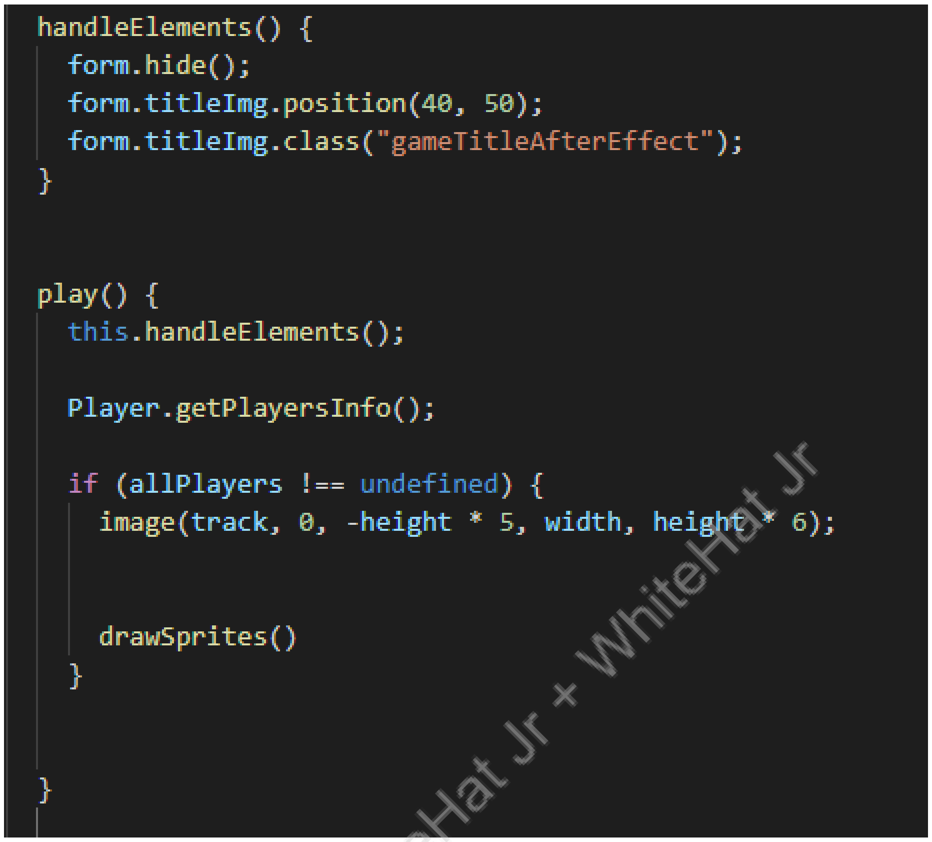
```
start() {
  player = new Player();
  playerCount = player.getCount();

  form = new Form();
  form.display();

  car1 = createSprite(width / 2 - 50, height - 100);
  car1.addImage("car1", car1_img);
  car1.scale = 0.07;

  car2 = createSprite(width / 2 + 100, height - 100);
  car2.addImage("car2", car2_img);
  car2.scale = 0.07;

  cars = [car1, car2];
```

9. Get information about players from the database.
    - Declare a global variable **allPlayers** in **sketch.js.**
    - Write queries to **getPlayerInfo()** in player.js.
    - **getPlayerInfo()** is a **static** function, it does not need to be called by each object.
    - Static functions can be called by the class itself.



```
static getPlayersInfo() {
  var playerInfoRef = database.ref("players");
  playerInfoRef.on("value", data => {
    allPlayers = data.val();
  });
}
```

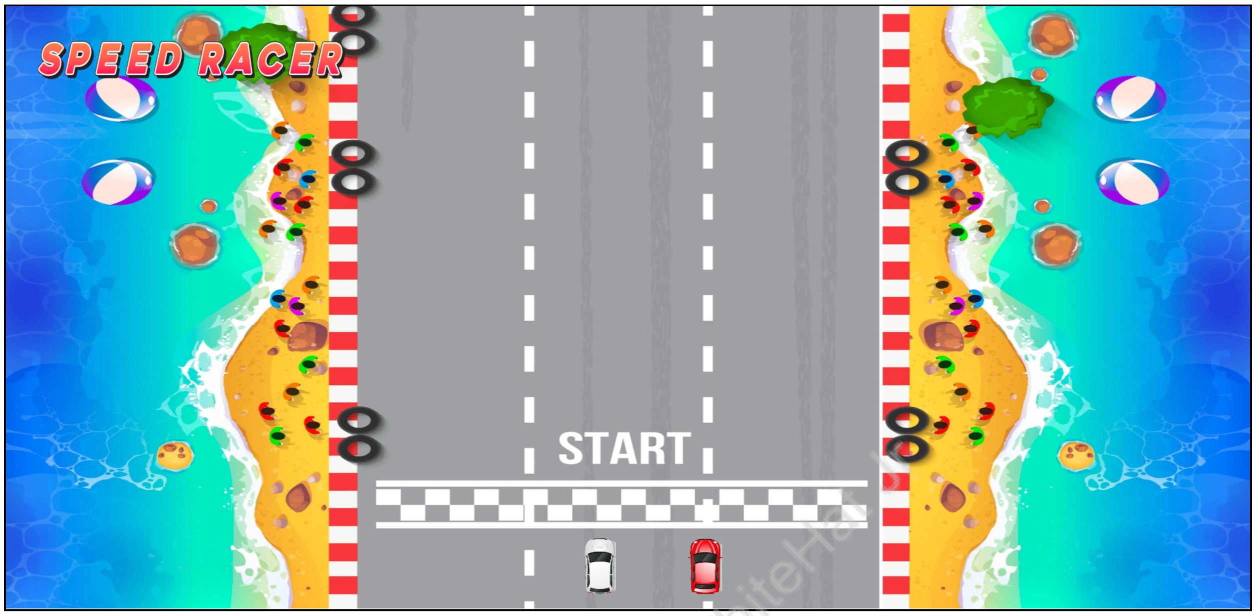
10. Create a **play()** method in **Game.js:**
    - In the **play()** function, we are displaying the track image to go beyond canvas size.
    - Display sprites in **play()** method.
    - Create **handleElements()** to hide the form object and call it in **play().**



```
handleElements() {
  form.hide();
  form.titleImg.position(40, 50);
  form.titleImg.class("gameTitleAfterEffect");
}


play() {
  this.handleElements();

  Player.getPlayersInfo();

  if (allPlayers !== undefined) {
    image(track, 0, -height * 5, width, height * 6);


    drawSprites()
  }


}
```

11. Make use of the **If** condition to make sure that sprites and track will be visible only once both players have joined.

OUTPUT:



What's next?

In the next class, you will move the cars forward with an arrow key and update the distance to the database. We will add an identifier for an active player. You will also be introduced to a Game Camera.

EXTEND YOUR KNOWLEDGE:

1. Know more about for-In loop from the following link created by Mozilla and individual contributors: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in

   About MDN by Mozilla Contributors is licensed under CC-BY-SA 2.5.