




Topic	GAME STATES & PLAYER INFORMATION	
Class Description	Students will learn how to update and access the player's information from the database. They will also learn to change the game states based on player count.	
Class	C37	
Class time	55 mins	
Goal	<ul style="list-style-type: none"> • Update player count in the database. • Change game state. • Create player sprites. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ VS Code Editor ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen • Student Resources <ul style="list-style-type: none"> ○ VS Code Editor ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen 	
Class structure	WARM-UP Teacher-Led Activity Student-Led Activity WRAP-UP	5 Mins 15 Mins 30 Mins 5 Mins
WARM-UP SESSION - 5 mins		
<div>  </div> <p>Teacher starts slideshow from slides 1 to 14</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		

Teacher Action	Student Action
<p>Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?</p> <p>Run the presentation from slide 1 to slide 4</p> <p>Following are the WARM-UP session deliverables:</p> <ul style="list-style-type: none"> • Greet the student. • Revision of previous class activities. • Quizzes 	<p>ESR: Hi, thanks, Yes I am excited about it!</p> <p>Click on the slide show tab and present the slides</p>
Q&A Session	
Question	Answer
<p>Select the correct block of code to give HTML to the question created.</p> <div data-bbox="162 1068 980 1480">  </div> <p>A. <code>this.question("Question:- What starts and ends with the letter 'E', but has only one letter? ");</code></p> <p>B. <code>this.question.html("Question:- What starts and ends with the letter 'E', but has only one letter? ");</code></p> <p>C. <code>this.html("Question:- What starts and ends with the letter 'E', but has only one letter? ");</code></p> <p>D. <code>question.html(["Question:- What starts and ends with the letter 'E', but has only one letter? "]);</code></p>	<p>B</p>

<p>What will the following code block do?</p> <pre>this.message.html("Thank You, Your Answer Has Been Submitted"); this.message.position(350, 350);</pre> <p>A. It will create a message having a position of x = 350 only.</p> <p>B. It will create a message having position of x=350 and y =350</p> <p>C. It will create an input box.</p> <p>D. It will create a submit button.</p>	<p>B</p>
<p>Continue the WARM-UP session</p>	
Teacher Action	Student Action
<p>Run the presentation from slide 5 to slide 14 to set the problem statement.</p> <p>Following are the objective of the WARM-UP session:</p> <ul style="list-style-type: none"> • Appreciate the student. • Update the player information and player count to the database. 	<p>Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.</p>
<p>Teacher ends slideshow </p>	
<p>TEACHER-LED ACTIVITY - 15 mins</p>	
<p>Teacher Initiates Screen Share</p>	
<p><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Add players information to the database • Change Game State as per Player count. 	
Teacher Action	Student Action
<p><i>The teacher downloads the Teacher Activity 1 code and explains the function.</i></p>	

In the last class, we have created the **Form**, **Player**, and **Game** class.

Now, first of all, 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.

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

We will make a 2-player game. 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.

Any ideas on how to do this?

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).

Let us check the `Game.js` where we can see that **getState()** is already given to us.

Can you quickly explain the code?

Do you remember while synchronising a ball game we learned about **.ref()** and **.on()** methods?

Here we are using **.ref()** to pass the location of the 'gameState' field of the database and using **.on()** we are

ESR: Varied.

ESR: Varied

ESR: Yes

reading the value of the '**gameState**' field and saving it to the global variable **gameState**

*The teacher can show global variables declared for **gameState** and **playerCount** in the **sketch.js**.*

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.

```
class Game {
  constructor() {}
  //BP
  getState() {
    var gameStateRef = database.ref("gameState");
    gameStateRef.on("value", function(data) {
      gameState = data.val();
    });
  }
}
```

Now, we will go to **sketch.js** to see where we are calling this function to read the game state even before showing the form to the users.

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

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

```
start() {
```

```
player = new Player();  
playerCount = player.getCount();  
  
form = new Form();  
form.display();  
  
}
```

Now let us add player's information to the database.
Why are we creating this class Player?

ESR: To manage information about each player.

Yes, from this class we will store and retrieve player information to and from the database.

As you can see in the **constructor()**, we will be creating various properties for each player. Can you read aloud and we understand the use of each property?

The student reads from code and explains each property.

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

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.

We can also see two more methods in **player.js**.
getCount() & updateCount().

Do you recollect what these methods are doing?

ESR: Varied

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**'.

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

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

“/” is used on updateCount to refer to the root directory.	
<p>Now we will write a function, to give the position to each player and add their properties to the database.</p> <p>So which function do we use to add values in the database? <i>This has been covered in C35.</i></p> <p>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.</p> <p>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.</p> <p>If the 'playerCount' is 1, we create a database entry for player1 and we set the name for it, and so on.</p> <p>→ First we use .ref() to give a location in the database. --> databaseReference.set() will create and save the database reference.</p>	<p>ESR: .ref() and .set()</p> <p><i>The student listens, observes, and asks questions.</i></p>
Create a addPlayer() in player.js	


```

    }

    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 Hierarchy in
database

To Give x position to
both players, one on
left from
center(width/2) &
and other on right

Updating field in
database

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.

Update handleMousedPressed() in form.js

```
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);  
  });  
}
```

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.

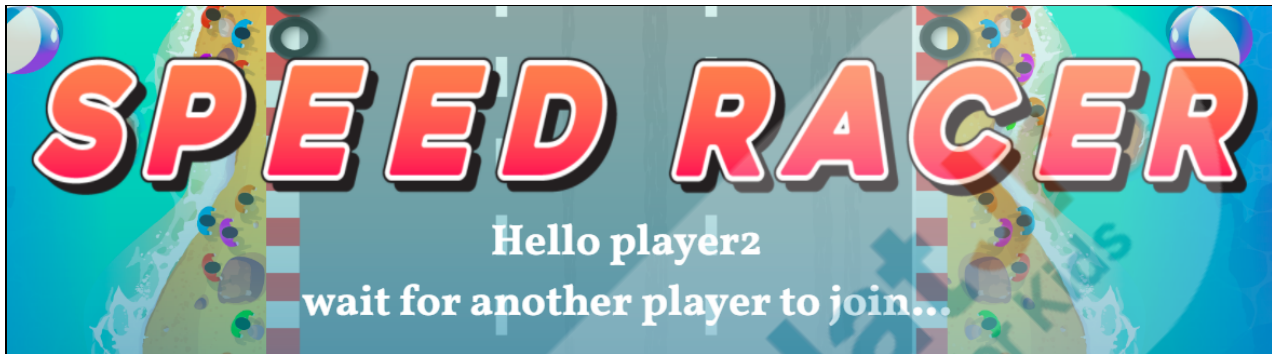
The player's name will be assigned to the property **player.name**.

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

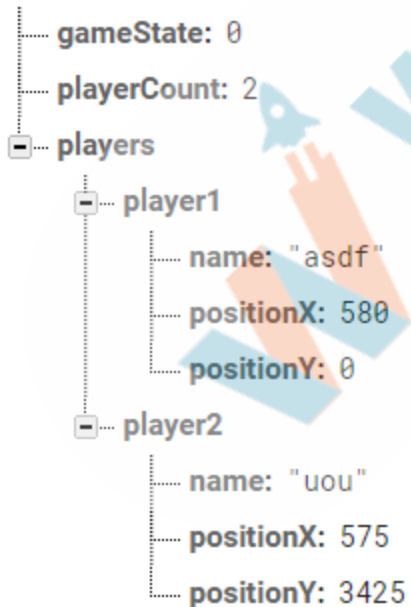
data in the database.

Let us run the code in two browsers to check if data is getting saved in the database.




Database:

multiplayer-car-racing-g-e094a-default-rtdb



The data is getting added to the database. We can see **'playerCount'** has also been updated to 2.

Let's say we are making a two-player game. Then what should we do to ' gameState ' once both players have joined?	ESR: It should change to 1.
<p>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.</p> <p>Would you create these methods?</p> <p>Now it's your turn. Please share your screen with me.</p>	ESR: Yes.
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY - 30 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Fullscreen. 	
<p style="text-align: center;"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Create updateState() method • Create the play() method. • Create a Static getPlayerinfo() method. 	
<div style="text-align: center;">  <p>Teacher starts slideshow for slide 15 and slide 16</p> <p>Refer to speaker notes and follow the instructions on each slide.</p> </div>	
Teacher Action	Student Action
<p><i>The teacher guides the student to download the Student Activity 1 code and opens it in the VS code editor.</i></p> <p><i>Replace the SDK code with the SDK from the student database in index.html.</i></p>	<p><i>The student downloads the Student Activity 1 code and opens it in the VS code editor.</i></p>

<p>Let us first start with the update() method in Game.js to update gameState.</p> <p><i>If the student is very slow, the teacher can allow him to copy updateCount() from player.js and make changes. Revise .ref() .update().</i></p>	<p><i>The student writes updateState() in Game.js.</i></p>
<p>Create update() in Game.js</p> <pre> update(state) { database.ref("/").update({ gameState: state }); } </pre>	
<p>When the update method is called we will pass a number to it which will be saved as an argument state. Post that, we are referring to the root database ('/') and using update() to change the gameState field with the value of the state.</p> <p>Awesome! Next, we need to update the gameState to 1 and play the game when two players are joined.</p> <p>How do we do that?</p> <p>Come to sketch.js, here inside the function draw() you can see if conditions are given to update State to 1 when 'playerCount' = 2 and call the play() method when 'gameState' is 1.</p>	<p>ESR: We can use the if condition to check playerCount.</p>

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

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

What should we do next?

Correct, go to **Game.js** and we will create a new method **play()**.

What do we need this method for?

We will use **image()** to show the terrain. If you check, the images of track and cars are preloaded.

ESR: create **play()** in **game.js**

*Student creates a method
play(){
}*

ESR: To show track and race cars.

The Student checks the code to preload the images.

Preload car and track images in sketch.js.

```
function preload() {
  backgroundImage = loadImage("assets/background.png");

  car1_img = loadImage("assets/car1.png");
  car2_img = loadImage("assets/car2.png");
  track = loadImage("../assets/track.jpg");
}
```

But we won't be able to see any output yet, because we have to write the **play()** method and create the track and cars as well.

Let us 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.

The teacher can show variables for car1, car2, and an array cars are created in sketch.js

You already know how to do that right?

ESR: Yes, using sprites

Now let's 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

The student writes code for

<p>car2 in that array.</p> <p><i>Remember to call <code>Player.getCount()</code> in <code>start()</code>.</i></p>	<p><i>creating sprites for cars and adds images to it.</i></p>
<p>Creates sprites inside start() of Game.js.</p> <pre> 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]; } </pre>	
<p>One more thing we need to do before we start creating a method.</p> <p>We need to fetch player's details from the database.</p>	<p>.</p>
<p>Let's 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.</p> <p>We are trying to get all the players' info here - the work</p>	<p><i>The student writes the static getPlayerInfo() function to capture data of all the players in a variable called "allPlayers".</i></p>

<p>doesn't involve any particular object.</p>	
<p>Create a static getPlayerInfo() in player.js.</p> <pre>static getPlayerInfo() { var playerInfoRef = database.ref("players"); playerInfoRef.on("value", data => { allPlayers = data.val(); }); }</pre>	
<p>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. We will learn how to use it soon.</p> <p>Let's first declare a static function that gets all the player data and stores it.</p> <p>The player's data will be stored as JSON - since the Firebase database structure is of JSON type.</p> <p>Do you remember the JSON data structure? Where is it used?</p>	<p>ESR: Yes, { key1: item1 , key2: item2, }</p> <p>It is used to store data in an organized way.</p>
<p>You can see one more method given in Game.js as handleElements()</p> <p>This function helps us to hide form once the game is in play mode.</p> <p>So now create a method play() in Game.js</p>	

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

The Student writes `play()` in `game.js` and calls `getPlayerinfo()` and `image()` command.

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

play() {
  this.handleElements();

  Player.getPlayersInfo();

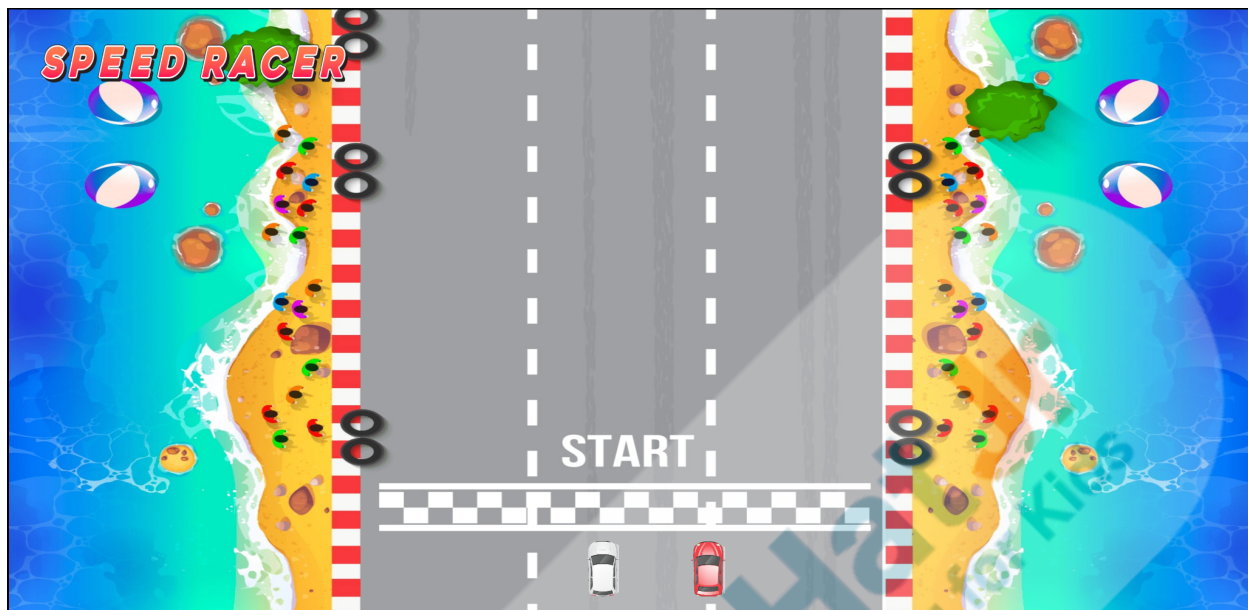
  if (allPlayers !== undefined) {
    image(track, 0, -height * 5, width, height * 6);

    drawSprites()
  }
}
```

We created **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)**

<p>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.</p> <p>If this condition is true then we will display the track image on the canvas using the image() method.</p> <p>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.</p> <p>For the width of the track, set as width and for the height keep it as height*6.</p>	
<p>Now let's run the code to see the output.</p> <p>This is our login screen, copy the local server URL and paste it in the new tab, and press enter.</p> <p>The game will run and you can enter the name of the 2nd player to start the game.</p> <p><i>Make sure to reset the 'gameState' and 'playerCount' to 0 and remove players' fields from the database every time you run the code.</i></p>	<p>ESR: <i>The student runs the code and observes the output.</i></p>
<p>OUTPUT:</p>	



Today we have made the basic functionality of our game.
We got the data from the database and made changes in
the database as well.

Great job!

Sounds fun right?

ESR: Varied.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins

Teacher starts slideshow








from slide 17 to slide 26

Teacher Action

Student Action

<p>Run the presentation from slide 17 to slide 26</p> <p>Following are the objective WRAP-UP sessions:</p> <ul style="list-style-type: none"> • Appreciate the student. • Revise the current class activities. • Discuss the quizzes. 	<p>Discuss with the student the current class activities and Student will ask questions related to the activities.</p>
<p>Quiz time - Click on in-class quiz</p>	
Question	Answer
<p>Firestore database stores the information/data in which of the following structures?</p> <p>A. parent-child node B. tabular C. sequential D. indexed</p>	<p>A</p>
<p>We can define all the properties of the Form - input, button, and greeting elements in the _____ of the class Form.</p> <p>A. setElementsStyle() B. hide() C. constructor() D. setElementsPosition()</p>	<p>C</p>
<p>Functions that can be called using class and do not require any object are known as?</p> <p>A. Dynamic B. Friend C. Static D. Async</p>	<p>C</p>

End the quiz panel	
FEEDBACK <ul style="list-style-type: none"> • Appreciate the student's efforts in the class. • Ask the student to make notes for the reflection journal along with the code they wrote in today's class. 	
Teacher Action	Student Action
<p>You get Hats off for your excellent work!</p> <p>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.</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div> <div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div> </div>
<p>* This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.*</p> <p>Project Overview MYQUIZ GAME</p> <p>Goal In Class 37, you have learned how to update and access the players' information from the database. In this project, you will apply what you have learned in the class to create a two player quiz game and store their response for the quiz question in the database.</p>	

<p>Story:</p> <p>Prakriti loves asking quizzes and she always tries to find unique questions and ask different people. But now, she is thinking of creating her own multiplayer quiz game where she can ask a quiz question to different people at the same time.</p> <p>Can you help her in creating the game?</p> <p>I am excited to see your project.</p> <p>Bye Bye!</p>	
<div>  Teacher ends slideshow </div>	
<div> Teacher Clicks  </div>	
ADDITIONAL ACTIVITIES	
<p>Additional Activities</p> <p><i>Encourage the student to write reflection notes in their reflection journal using Markdown.</i></p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> • What happened today? <ul style="list-style-type: none"> ◦ Describe what happened. ◦ The code I wrote. • How did I feel after the class? • What have I learned about programming and developing games? 	<p><i>The student uses the Markdown editor to write their reflections in a reflection journal.</i></p>

- What aspects of the class helped me? What did I find difficult?

Activity	Activity Name	Links
Teacher Activity 1	p5 Boilerplate for Multiplayer Car Racing Game	https://github.com/pro-whitehatjr/C37RV_SpeedRacer_TeacherActivity
Teacher Activity 2	Teacher Reference Code	https://github.com/pro-whitehatjr/C37RV_SpeedRacer_ReferenceCode
Student Activity 1	p5 Boilerplate for Multiplayer Car Racing Game	https://github.com/pro-whitehatjr/C37RV_SpeedRacer_StudentActivity
Teacher Reference	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Assessment/PRO_VD/PRO_C37_V3_lit_WithCues.html
Teacher Reference	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/6311b140-254e-40c1-a28a-adc0213b70cc.pdf
Project Solution	MyQuiz Game-2	https://github.com/pro-whitehatjr/9d115a05ebbe489bf61e1278508d0487