**PRO-C37**

Tema: Juego de la Base de Datos

**Descripción:** Los alumnos crean el juego de carreras de autos multijugador, que ocurre solo en la base de datos. Los alumnos presionan la tecla de flecha hacia arriba, para cambiar la propiedad de distancia recorrida de cada jugador, en la base de datos. Las

distancias recorridas por todos los jugadores, se muestran en la pantalla con el jugador que juega el juego resaltado.

**Objetivos:** ● Estructurar el código del juego para incluir más propiedades, y funciones para cada clase. ● Escribir el código de comportamiento, en el estado de juego del juego, cuando todos los jugadores están conectados. ● Mostrar a todos los jugadores con sus puntuaciones de distancia recorrida.

Primero, recordemos en dónde nos quedamos en la última clase. Creamos un formulario que permite a los jugadores iniciar sesión y ver el saludo. El nombre del jugador se registra en la

base de datos. **REA:** Creamos un formulario que permite a los jugadores iniciar sesión y ver el saludo. El nombre del jugador se registra en la base de datos.

Un código mal organizado, dificultaría la búsqueda de errores en tu código. También sería difícil escribir el código nuevo, en la parte superior de tu código escrito previamente, sin causar errores.

Se necesita un poco de tiempo para pensar en cómo estructurar tu código; pero todo el tiempo dedicado a hacer que tu código se vea bien, estructurado y más legible, ¡siempre vale la pena!

¡En la clase de hoy, crearemos un juego de carreras de autos que se ejecuta solo en la base de datos!

Comencemos.: Permítanme mostrarles rápidamente lo que estaremos construyendo el día de hoy.

El profesor clona el código del enlace de referencia, agrega la configuración de firebase para su propia base de datos, y hace los cambios necesarios en la estructura de la base de datos de firebase.

Asegúrate de que gameState y playerCount sean 0.

¿Sabes por qué es importante?: **REA:** Esos son los estados iniciales del juego.

*El profesor ejecuta el código en el servidor web, abre 4 pestañas diferentes, e ingresa los nombres de los jugadores para mostrar el inicio del juego.*

*El profesor presiona la tecla de flecha hacia arriba en los diferentes navegadores, para mostrar el número que se muestra en los diferentes jugadores.*

*El profesor también muestra las entradas en la base de datos.*

¿Qué está sucediendo aquí? **REA:** Tan pronto como 4 jugadores inician sesión, el formulario y el saludo desaparecen. Cuando comienza el Juego, todos los jugadores y la distancia recorrida por ellos se muestran en la pantalla. Se resalta el jugador que inició sesión. La distancia actual recorrida por el jugador, aumenta cuando se presiona la tecla de flecha hacia arriba.

¿Qué crees que podría estar sucediendo en el código?. ¿Cómo crees que conseguiremos el resultado que acabas de ver?

Codifiquemos para esto. Mientras codificamos, siempre intentaremos mantener nuestro código

estructurado.

*Guíe al alumno para que cambie firebaseConfig en index.html.*

*Guíe al alumno para que cambie la estructura de la base de datos de firebase como se muestra en la siguiente imagen.*

Comencemos con la Clase Player (Jugador). Agreguemos nuevas propiedades y funciones a nuestra clase Player, para satisfacer las necesidades de nuestro nuevo programa.

El constructor en nuestra clase Player está vacío. ¿Cuáles son las propiedades de un objeto de un

jugador?: **REA:** nombre, distancia recorrida, índice del jugador.

Exactamente, creemos estas propiedades dentro del constructor del jugador.

**Player.js**

*constructor(){*

*this.index = null;*

*this.distance = 0;*

*this.name = null;* }

Usemos la función update() para actualizar tanto el nombre, como la distancia del jugador.

*getCount(){*

*console.log("Indice del nuevo jugador A: ", this.index);*

*console.log("Nombre del nuevo jugador A: ", this.name);*

*console.log("Distancia del nuevo jugador A: ", this.distance);*

*var playerCountRef = database.ref('playerCount');*

*playerCountRef.on("value",(data) => {playerCount = data.val();}) }*

*updateCount(count){*

*console.log("PASO 4: player1.updateCount()")*

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

*console.log("Contador actualizado B: ", playerCount); }*

*update(){*

*console.log("PASO 3: player1.update() ");*

*var playerIndex = "players/player" + this.index*

*database.ref(playerIndex).set({name: this.name, distance: this.distance})*

*console.log("Indice del nuevo jugador B: ", this.index);*

*console.log("Nombre del nuevo jugador B: ", this.name);*

*console.log("Distancia del nuevo jugador B: ", this.distance); }*

Además, necesitamos cambiar la referencia a la base de datos, ya que todos los jugadores ahora estarán dentro de "jugadores" en la base de datos.

Usaremos la propiedad 'índex - índice' del jugador, para actualizar los valores de ese jugador en particular en la base de datos.

**Sketch.js**

*var allplayers;*

*El alumno escribe la función estática getPlayerInfo() para capturar datos de todos los jugadores en una variable llamada "allPlayers" (todos los Jugadores).*

Escribamos una función para obtener toda la información de los jugadores. Esta función no se adjuntará a ningún jugador en particular. Podemos declararlo como una función estática.

Las funciones estáticas no se adjuntan a cada objeto de la clase. Estamos tratando de obtener toda la información de los jugadores aquí; el trabajo no involucra ningún objeto en particular.

Las funciones estáticas son aquellas funciones comunes, que son llamadas por la propia clase en lugar de por los objetos de la clase. Usamos la palabra clave 'estática' antes de la función, para convertirla en una función estática. Pronto aprenderemos a usarla.

Primero, declaremos una función estática que obtiene todos los datos de los jugadores y los almacena. Los datos de los jugadores se almacenarán como JSON, ya que la estructura de la base de datos firebase es de tipo JSON. ¿Recuerdas la estructura de datos JSON? ¿Dónde se usa?: **REA**: Se utiliza para almacenar datos de forma organizada.

**Player.js**

*static getPlayerInfo(){*

*var playerInfoRef = database.ref('players');*

*playerInfoRef.on("value", (data)=> {allplayers = data.val()});*

*console.log("playerInfoRef: ", playerInfoRef);*

*console.log("allplayers: ", allplayers); }*

Abramos la Clase Form (Formulario). Podemos definir todas las propiedades del formulario: entrada, botón, elemento de saludo dentro del constructor de la clase. La entrada, botón y saludo dentro de la función mousePressed() debe cambiarse a this.input, this.button y this.greeting.

Además, "this – esto" se refiere al objeto que llama a la función. Queremos que 'this' dentro de la

función mousePressed, se refiera al objeto formulario. Sin embargo, el elemento de botón está llamando a la función mousePressed() y 'this' aquí, se refiere al elemento de botón.

Esto no es lo que queremos. Para que 'this' continúe refiriéndose al objeto formulario, usamos funciones de flecha.

Las funciones de flecha unen la función al objeto original que la llama.

Aquí se llama mousePressed dentro de la función display (visualización), que es llamada por el objeto formulario. ()=> La función de flecha asegura que 'this' permanece vinculado al objeto formulario.

*También guíe al alumno a escribir una función hide() que será llamada cuando cambie el estado del juego, y cuando queramos ocultar el formulario.*

**Form.js**

*class Form{*

*constructor(){*

*this.input = createInput("Nombre");*

*this.button = createButton("Play");*

*this.greeting = createElement("h3"); }*

*hide(){*

*this.greeting.hide();*

*this.button.hide();*

*this.input.hide(); }*

*display(){*

*var title = createElement("h2");*

*title.html("Car Racing Game");*

*title.position(130,0);*

*this.input.position(130,60);*

*this.button.position(250,200);*

*this.button.mousePressed(()=>{*

*this.input.hide();*

*this.button.hide();*

*player1.name = this.input.value();*

*playerCount+=1;*

*player1.index = playerCount;*

*player1.update();*

*player1.updateCount(playerCount);*

*this.greeting.html("Hellouuuu " + player1.name);*

*this.greeting.position(127,157); }); } }*

Pasemos a la Clase Game (Juego). Escribamos una función play() que se llamará cuando el estado del juego alcance el estado play o 1. Dentro de la función play(), oculta el formulario.

Obtén todos los datos de los jugadores, y muéstralos en la pantalla. Cambia la distancia, y actualízala en la base de datos cuando se presione la tecla de flecha 'ARRIBA'.

**Nota:** usamos 'plr' porque el jugador ya está definido. ¿Puedes escribir el código para hacer que el jugador actual aquí sea "rojo"?.

Abrir el formulario y presionar jugar rápidamente,, arrojará un error porque la consulta de la base

de datos para obtener playerCount aún no se ha completado. Para evitar este error, podemos

configurar un oyente asíncrono .once() que obtendrá los datos playerCount solo una vez, y luego

ejecutará getCount() para configurar un oyente permanente.

La función asincrónica esperará el valor playerCount, y solo entonces, se creará y mostrará el formulario.

**Game.js**

*class Game{*

*constructor(){}*

*getState(){*

*var gameStateRef= database.ref('gameState');*

*gameStateRef.on("value",function(data){gameState = data.val();}) }*

*update(state){*

*database.ref("/").update({gameState: state}); }*

*async start(){*

*if(gameState === 0){*

*player1 = new Player() //Crera un nuevo objeto para el jugador*

*var playerCountRef = await database.ref('playerCount').once("value");*

*if(playerCountRef.exists()){*

*playerCount = playerCountRef.val();*

*player1.getCount(); }*

*form1 = new Form();*

*form1.display(); } }*

*play(){*

*form1.hide();*

*textSize(27);*

*text("Game Start", 120,100);*

*Player.getPlayerInfo();*

*if(allplayers !== undefined){*

*var display\_position = 130;*

*for(var plr in allplayers){*

*if(plr === "player" + player1.index)*

*fill("red");*

*else*

*fill("black");*

*display\_position +=20;*

*textSize(17);*

*text(allplayers[plr].name + ": " + allplayers[plr].distance, 117, display\_position); }}*

*if(keyIsDown(UP\_ARROW) && player1.index !== null){*

*player1.distance +=50;*

*player1.update(); } } }*

Finalmente, modifiquemos el archivo sketch para escribir condiciones para cambiar el estado del juego, y llamar a la función play()

*El alumno escribe el código para actualizar el estado del juego, cuando el número de jugadores es 4, y llama a game.play() cuando el estado del juego es 1.*

**Sketch.js**

*var gameState =0;*

*var playerCount=0;*

*var database;*

*var form1, game1,canvas;*

*var allplayers, player1;*

*var distance=0;*

*function setup(){*

*canvas = createCanvas(400,400);*

*database = firebase.database();*

*game1 = new Game();*

*console.log("PASO 1: game1.getState()");*

*game1.getState();*

*console.log("PASO 2: game1.start()");*

*game1.start(); }*

*function draw(){*

*background(205, 92, 92);*

*if(playerCount === 4){game1.update(1); }*

*if(gameState === 1){*

*clear();*

*game1.play(); }}*

¿Cómo te sentiste con la clase del día de hoy?

Aunque parezca imposible de creer, ¡ya hemos hecho la mayor parte del trabajo duro en este juego!

¡Todo lo que tenemos que hacer ahora, es dibujar el juego de autos usando sprites, y otras funciones de p5.play que ya conoces!

*¡*Felicidades por tu excelente trabajo!

Haremos esto en la próxima clase. ¡¡¡Excelente!!!