**PRO-C36**

Tema: Estructurar antes de Codificar

**Descripción:** Los alumnos diseñan un formulario usando p5 dom para permitir que los jugadores inicien sesión, y registren los nombres de los jugadores en la base de datos. También se registran el estado del juego y el recuento de los jugadores. Los alumnos usan el estilo de programación OOP para escribir el código

**Objetivos:** ● Crear un formulario para registrar el nombre de los jugadores en el juego. ● Actualizar playerCount y gameState en la base de datos. ● Utilizar el estilo de programación de OOP.

Necesitamos crear algún tipo de formulario, donde diferentes usuarios puedan iniciar sesión con su nombre e ingresar al juego. Cada vez que un nuevo usuario inicia sesión, se debe crear un nuevo Jugador.

También, debemos tener en cuenta la cantidad de jugadores en el juego y el estado del juego. Por ejemplo, cuando el estado del juego es 0 (WAIT - ESPERA), queremos que los jugadores vean el formulario de inicio de sesión donde registran su nombre como jugadores.

¿Alguna idea sobre cómo hacemos esto?. Hay varias formas en las que podemos hacer esto.

Podemos empezar a escribir el código inmediatamente. Pero los buenos programadores, antes de escribir el código, piensan en cómo estructurar su código.

¿Qué estilo de programación estamos usando en nuestros códigos hasta ahora?. REA: POO/OOP – Programación Orientada a Objetos (o por sus siglas en inglés, Object-Oriented Programming)

Para esta pequeña parte de nuestro juego, en el que les pedimos a los jugadores que inicien sesión, ¿cuáles son los diferentes objetos que pueden estar en nuestro juego? ¿Cuáles serán sus propiedades y funciones?

Necesitamos tener al menos 3 objetos-

**1.** **Form - Formulario:** El formulario debe contener el cuadro de entrada y un botón para iniciar sesión.

● Cuando se presiona el botón, el nombre del jugador debe registrarse en la base de datos, y debe crearse un nuevo jugador.

**2.** **Player - Jugador:** Se debe crear un nuevo objeto de jugador cada vez que un nuevo usuario inicia sesión.

Debe contener toda la información sobre el jugador: nombre, posición en el juego, etc.

● Por ahora, solo puede tener la propiedad name (nombre). También debería poder leer y escribir información del jugador en la base de datos, por ejemplo, el número o el nombre del jugador.

**3. Game Object**: Objeto del Juego: el objeto del Juego debe poder mantener el estado del juego. Debería poder mostrar la forma cuando el estado del juego es 0 (WAIT - ESPERAR) o el juego cuando el estado del juego es 1 (PLAY - JUGAR) o la tabla de clasificación cuando el estado del juego es 2 (END - FIN). ● Por ahora, solo consideraremos el caso en el que el estado del juego sea 0.

Ahora que sabemos cómo será la estructura básica de nuestro programa, ¡será bastante fácil escribir nuestro código! Sin esta estructura, escribir el código puede parecer complejo.

Con esta guía en mente, ¿por qué no comienzas con el ejercicio de codificación? Te estaré guiando en el

ejercicio.

Creemos nuestra nueva base de datos. Guíe al alumno para que inicie sesión en console.firebase.google.com con una estructura de base de datos equivalente a la siguiente:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Agregar la configuración de la DB en **index.html**

c*onst firebaseConfig = {*

*apiKey: "AIzaSyAPSSR7d4i63CRGSSpj5bISOkv\_dp0lmYQ",*

*authDomain: "carreras-multijugador-v1-1.firebaseapp.com",*

*databaseURL: "https://carreras-multijugador-v1-1-default-rtdb.firebaseio.com",*

*projectId: "carreras-multijugador-v1-1",*

*storageBucket: "carreras-multijugador-v1-1.appspot.com",*

*messagingSenderId: "54484927637",*

*appId: "1:54484927637:web:fbe16cfe5fa135eea7079c"*

*};*

*firebase.initializeApp(firebaseConfig); </script>*

Creemos una nueva carpeta en nuestro directorio llamada js. Este contendrá el plano de los 3 objetos de nuestro juego: Game (Juego), Form (Formulario) y Player (Jugador).

Creemos archivos para éstos e inclúyelos en el archivo **index.html.**

*<script src="./Js/Form.js"></script>*

*<script src="./Js/Game.js"></script>*

*<script src="./Js/Player.js"></script>*

Comencemos con el archivo **sketch.js** e incluyamos todas las variables globales que serán necesarias.

*var gameState =0;*

*var playerCount;*

*var database;*

*var form, player,game,canvas;*

*function setup(){*

*canvas = createCanvas(400,400);*

*database = firebase.database();}*

*function draw(){background(205, 92, 92);}*

*Guíe al alumno para crear las variables globales utilizadas en el programa, cree un lienzo y conéctese a la base de datos de Firebase.*

Primero escribamos la clase Game (Juego). Nuestro objeto Game debería poder leer el gameState y actualizar gameState. También debería poder iniciarse y mostrar el juego en la pantalla dependiendo del gameState.

El constructor de una clase se usa para dar propiedades a un objeto cuando se crea. Por ahora, podemos

mantener el constructor vacío. Escribamos funciones dentro de la Clase Game para getState y actualizar el estado.

- getState() simplemente leerá el estado del juego de la base de datos.

- update(state) actualizará el gameState en la base de datos a un valor que se le pasa entre paréntesis.

-> databaseReference.on() crea un oyente que sigue escuchando el gameState de la base de datos.

Cuando se cambia gameState en la base de datos, se ejecuta la función que se le pasa como argumento.

Nota: Aquí la función se escribe directamente dentro del oyente .on().

-> databaseReference.update () actualizará la referencia de la base de datos. Aquí "/" se refiere a la base de datos principal dentro de la cual se crea gameState.

También podemos crear una función start() que iniciará el juego, y se mostrará en la pantalla dependiendo

del estado del juego.

Por ahora, cuando el estado del juego es 0, queremos que se cree un formulario y un objeto de jugador.

Queremos mostrar el formulario y obtener playerCount.

Escribiremos el código para crear estos objetos, aunque el plano aún no esté definido. A esto se le llama

escribir el código usando abstracción

Escribiremos el código para estas clases y crearemos estos objetos después de esto.

**En Game.js**

*class Game{*

*constructor(){}*

*getState(){*

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

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

*console.log("Valor del gameState", gameState);*

*}*

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

*start(){*

*if(gameState === 0){*

*player1 = new Player()*

*player1.getCount();*

*form1 = new Form();*

*form1.display();}}}*

Vamos a escribir la clase de Form ahora. HTML se utiliza para crear cualquier contenido como un formulario en una página. HTML es similar a markdown en algunos aspectos. Un HTML contiene elementos que definen la estructura de una página.

Una página html simple contiene:

- head (cabeza): donde se agregan todos los scripts y hojas de estilo de

la página

- body (cuerpo): donde se agrega todo el contenido de la página. El cuerpo de una página HTML puede contener varios tipos diferentes de elementos:

- h1 , h2, h3: muestra encabezados de diferentes tamaños.

- input (entrada): para recopilar información del usuario.

- button (botón): para mostrar un botón.

Este modelo de una página HTML se llama Modelo de Objetos del Documento (o DOM por sus siglas en

inglés, Document Object Model). Usaremos la biblioteca Dom p5 para crear el formulario.

Mantendremos el constructor en la clase Form vacío. Escribamos una función display() que muestre el formulario.

Creamos un título para nuestro juego "Juego de Carreras de Autos":

- creamos un elemento h2.

- cambiamos el contenido html dentro del elemento.

- colocamos el título en el lienzo.

De manera similar, creamos la entrada y el elemento botón. Posicionamos la entrada y el elemento botón.

En **Form.js**

*class Form{*

*constructor(){}*

*display(){*

*var title = createElement("h2")*

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

*title.position(130,0);*

*var input = createInput("nombre");*

*var button = createButton("Play");*

*var greeting = createElement("h3")*

*input.position(130,60);*

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

Queremos saludar al jugador cuando el jugador escriba su nombre e inicie sesión. También, queremos actualizar el playerCount y el nombre del jugador en la base de datos button.mousePressed() se puede

usar para activar una acción cuando se presiona un botón del ratón. Espera una función como argumento.

Escribamos el código para mostrar un saludo y actualizar la base de datos cuando se presione el botón.

En **Form.js**

*button.mousePressed(function(){*

*input.hide();*

*button.hide();*

*var name = input.value();*

*playerCount=+1;*

*console.log("No. de jugadores: ", playerCount);*

*player1.update(name);*

*player1.updateCount(playerCount);*

*greeting.html("Holaaa " + name);*

*greeting.position(130,160); }); } }*

Finalmente, vamos a escribir el código para la Clase Player. Necesitamos escribir una función getCount() para obtener playerCount updateCount() para actualizar playerCount en la base de datos.

También, necesitamos actualizar el nombre del jugador en la base de datos. Para ello, necesitamos crear

nuevas entradas en la base de datos.

Podemos hacerlo mediante la concatenación de cadenas. Si playerCount es 1, creamos una entrada de base de datos para layer1 y le asignamos el nombre y así sucesivamente.

En **Player.js**

*class Player{*

*constructor(){}*

*getCount(){//Función para obtener playerCount*

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

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

*console.log("playerCountRef", playerCountRef);*

*console.log("playerCount", playerCount);}*

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

*update(name){*

*var playerIndex = "player" + playerCount;*

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

Finalmente, agreguemos algo de código en nuestro archivo sketch.js para crear un nuevo objeto Game,

obtener gameState y luego iniciar el juego.

En **sketch.js**

*function setup(){*

*canvas = createCanvas(400,400);*

*database = firebase.database();*

*game = new Game();*

*game.getState();*

*game.start(); }*

Probemos nuestro código. *El alumno ejecuta el código usando el servidor web 200 OK.*

*- El alumno abre el enlace en diferentes navegadores. - El alumno agrega los Nombres de los jugadores,*

*y observa los cambios en la base de datos de Firebase.*

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

**Reto:** Mascota virtual

**Objetivo:** En la clase 36, creaste un formulario para que los jugadores inicien sesión, agregaste una entrada para un nombre y un botón para jugar. También, creaste playerCount y gameState en la base de datos. Aprendiste a actualizar gameState y el recuento de jugadores en la base de datos. En este proyecto, tendrás que aplicar lo aprendido en clase y crear un juego de mascotas virtual.

**Historia:** Shreya realmente quiere una mascota. Pero nadie más en su familia quiere traer una mascota a

casa. Quiere crear un juego en el que pueda rastrear fácilmente las existencias de alimento (por ejemplo, leche) que tiene y el tiempo que alimenta al perro. También, debería poder agregar alimentos (botellas de leche) al almacenamiento de alimentos cuando esté terminado.

¿Puedes crear un juego de mascotas virtual para Shreya?