DANIEL CASTELAO

Tarea Unity PMDM – Firebase

2º DESARROLLO DE APLICACIONES MULTIPLATAFORMA

CPR Daniel Castelao

Curso 2023 – 2024

Ángel Castiñeira Durán

ÍNDICE

Enunciado

Utilización de Base de Datos Real Time con Unity

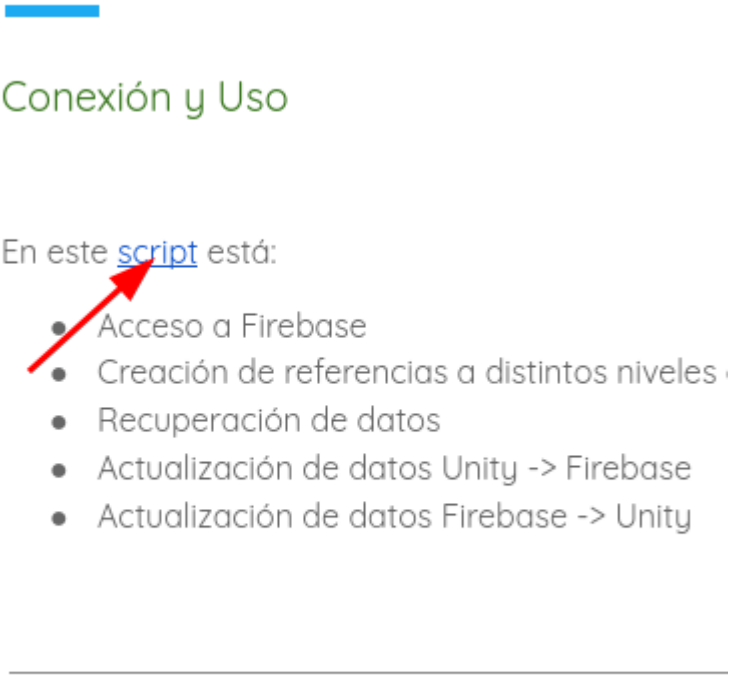
Implementa estas tres características:

1. En el inicio del juego, recoger las posiciones de varios elementos (los prefab) y posicionarlos en función de los datos de la base de datos
2. Actualizar datos en la base de datos segun avance el juego, por ejemplo, los puntos recogidos, las vidas, etc. Esto tiene que ser particular para cada jugador. Pensar que el juego lo juegan muchos jugadores y comparten la base de datos
3. Actualizar en tiempo real lo que ocurre en el juego. Guardar la posicion de un objeto, y usarla. Pensar en dos bolas, cada una, la mueve un jugador diferente. ¿Como sería moverlas tomando los valores de la base de datos?

Entrega un pdf explicando los pasos y el repositorio con los scripts

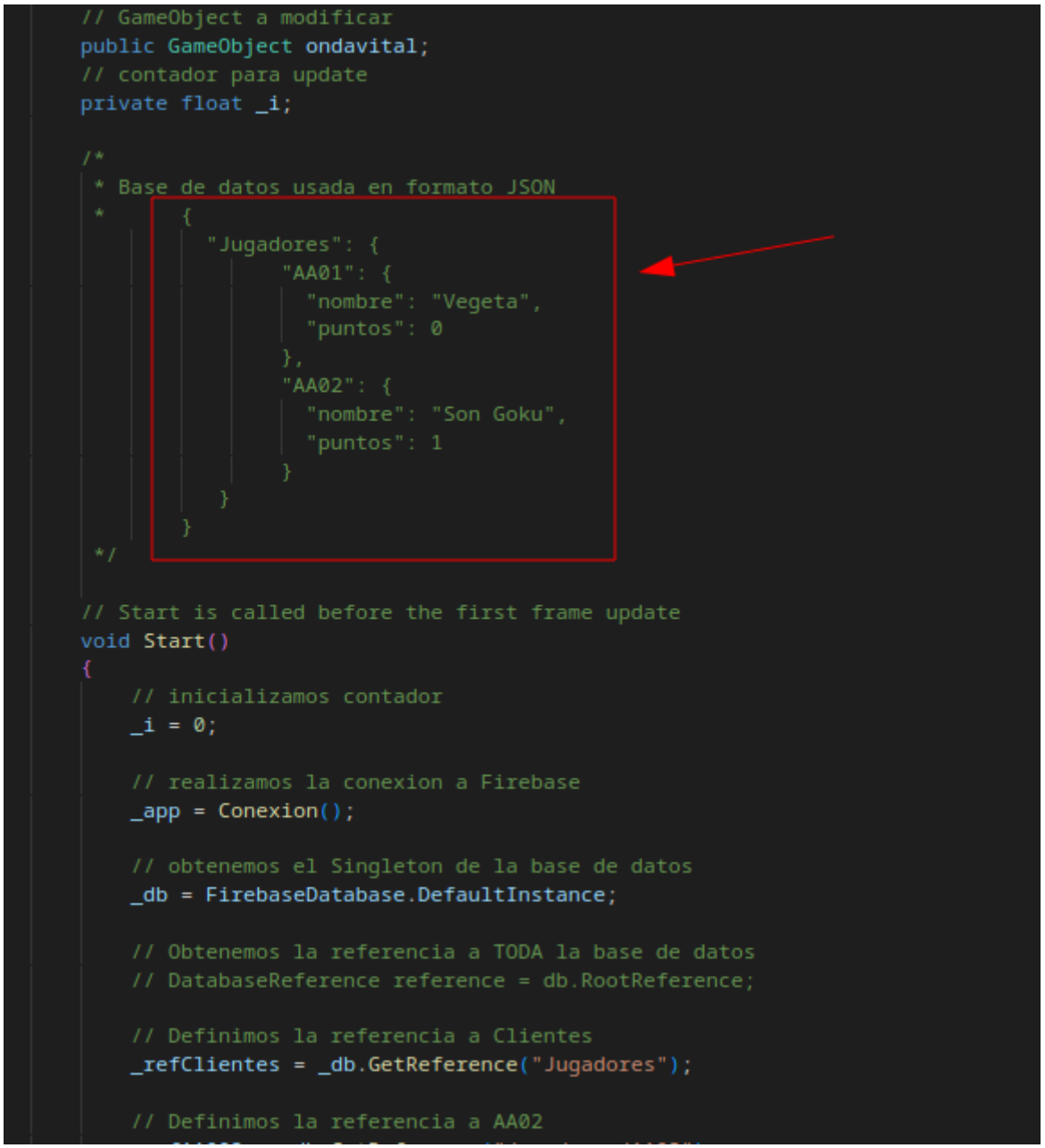
Previo

1. Voy a los apuntes de Esemtia de Firebase y copio las lineas del script.



2. En mi proyecto Unity creo un nuevo objeto vacío y le asigno un script. Pego el script copiado de los apuntes del script. Tanto el objecto como el script se llaman “RealTime”.

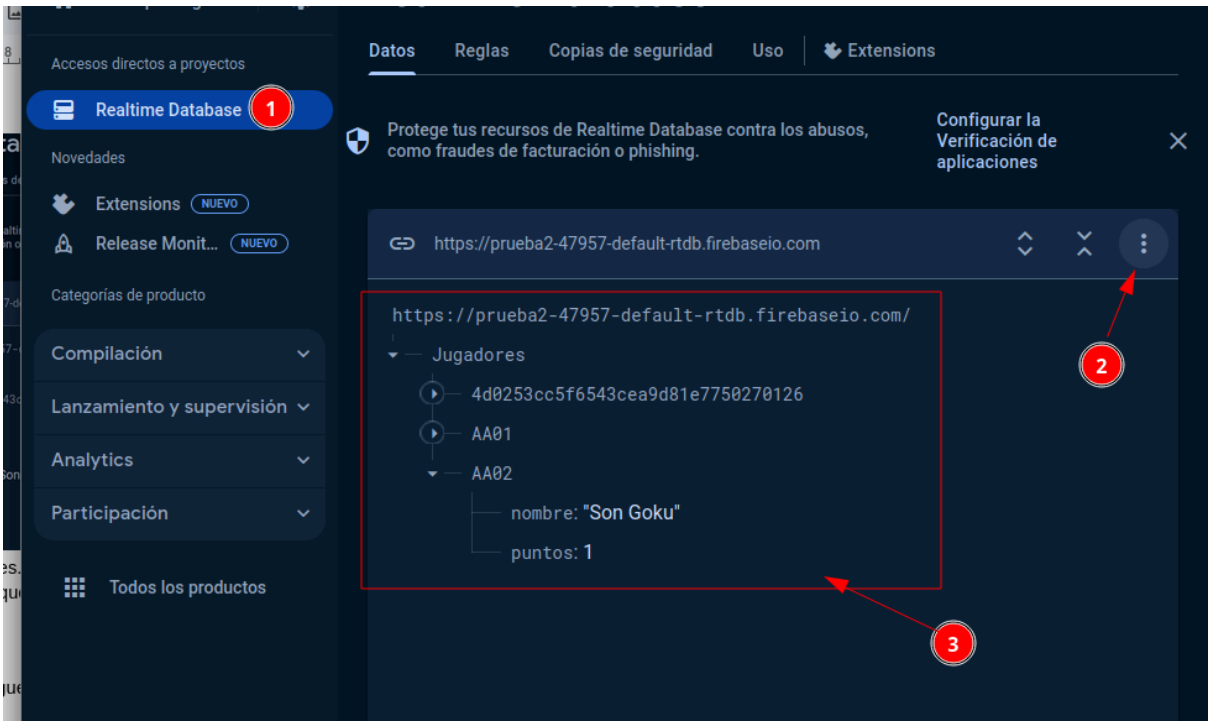
3. Dentro del script de RealTime copia las siguientes líneas del recuadro y las pego dentro de un nuevo archivo al que le llamaremos `datos.json`:



```
// GameObject a modificar
public GameObject ondavital;
// contador para update
private float _i;

/*
 * Base de datos usada en formato JSON
 *      {
          "Jugadores": {
              "AA01": {
                "nombre": "Vegeta",
                "puntos": 0
              },
              "AA02": {
                "nombre": "Son Goku",
                "puntos": 1
              }
          }
      }
 */

// Start is called before the first frame update
void Start()
{
    // inicializamos contador
    _i = 0;

    // realizamos la conexion a Firebase
    _app = Conexion();

    // obtenemos el Singleton de la base de datos
    _db = FirebaseDatabase.DefaultInstance;

    // Obtenemos la referencia a TODA la base de datos
    // DatabaseReference reference = db.RootReference;

    // Definimos la referencia a Clientes
    _refClientes = _db.GetReference("Jugadores");

    // Definimos la referencia a AA02
```

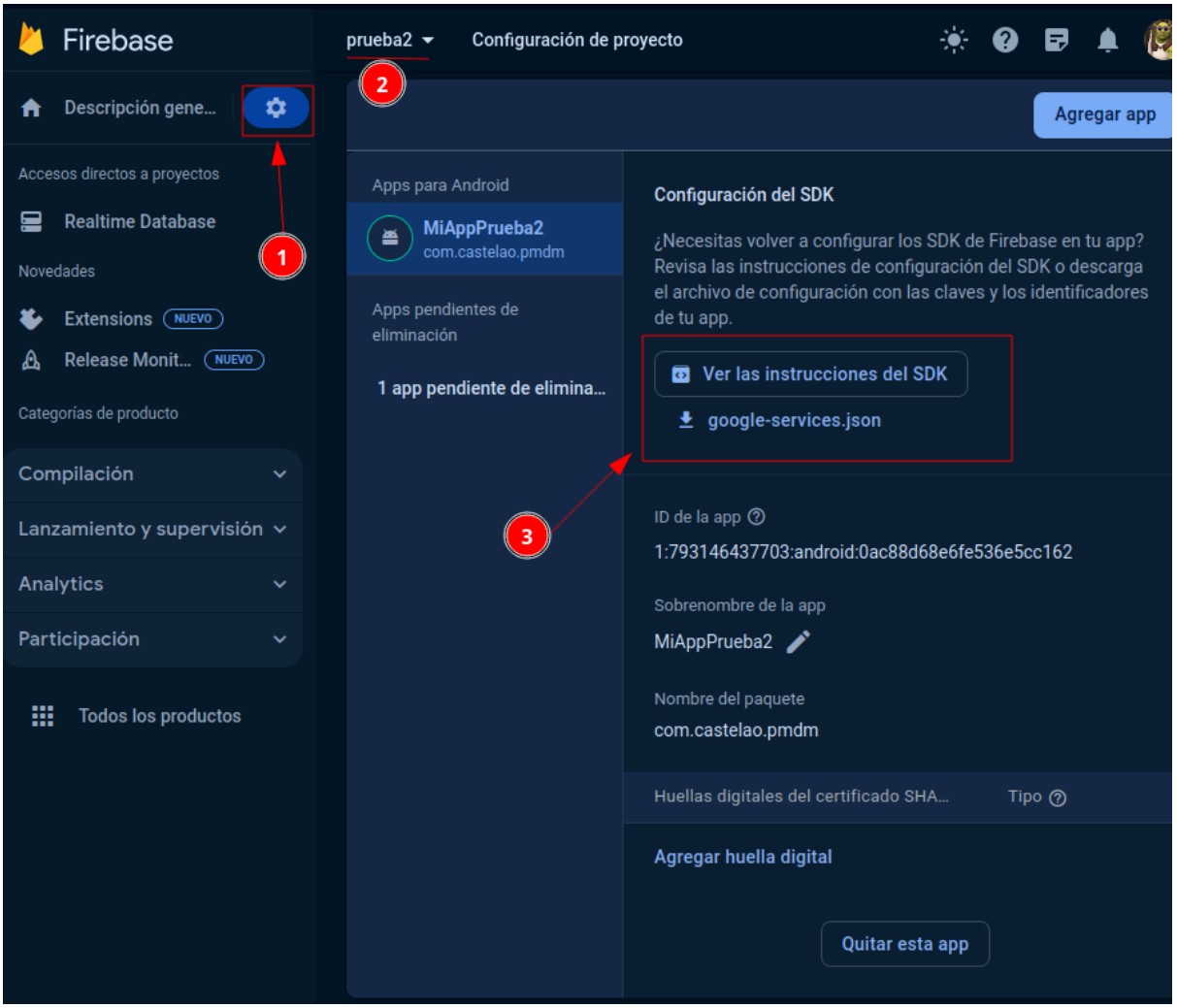
Líneas que hay que copiar correspondientes al script de `RealTime.cs`

El archivo JSON creado lo exporto a la base de datos creada el día anterior en clase.



Pasos para pegar el código del archivo JSON en el Firebase

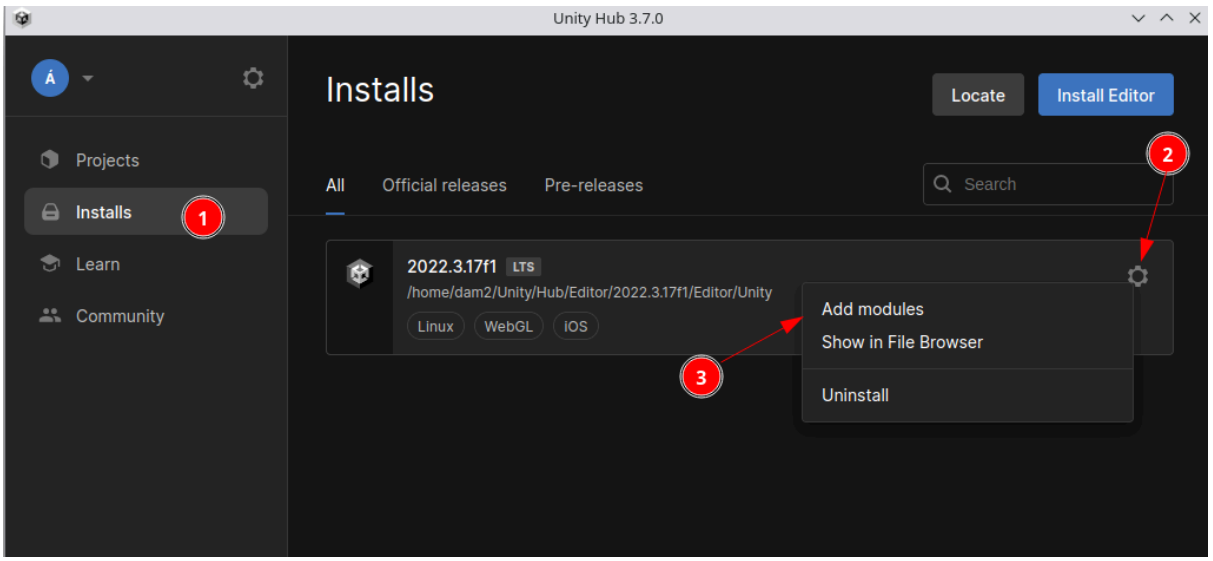
4. Descargar google.services.json y el SDK del Firebase.



Pasos para descargar SDK y JSON

5. Importar paquetes necesarios en unity.

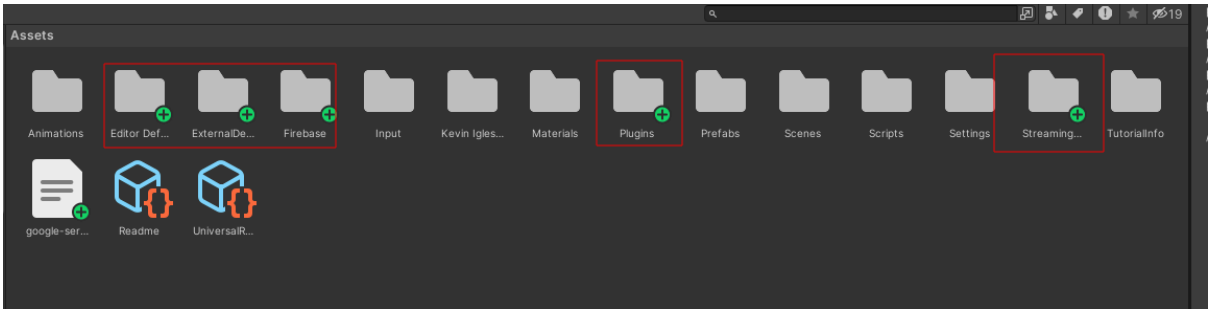
Previamente tenemos que importar el módulo “os build system” al proyecto.



Pasos para importar un módulo en Unity

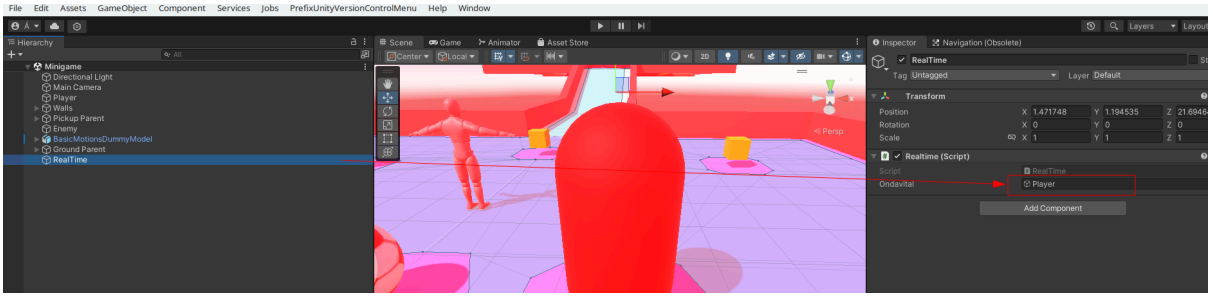
A continuación entramos en el proyecto, seleccionamos carpeta de los assets y dentro de esta:
click derecho → import package → custom package → seleccionamos la carpeta del jdk

Recordar que previamente la carpeta del JDK tiene que estar descomprimida.



Distribución de Assets después de importar el JDK

6. Añadir el objeto vacío al script del objeto creado de RealTime dentro del Inspector.

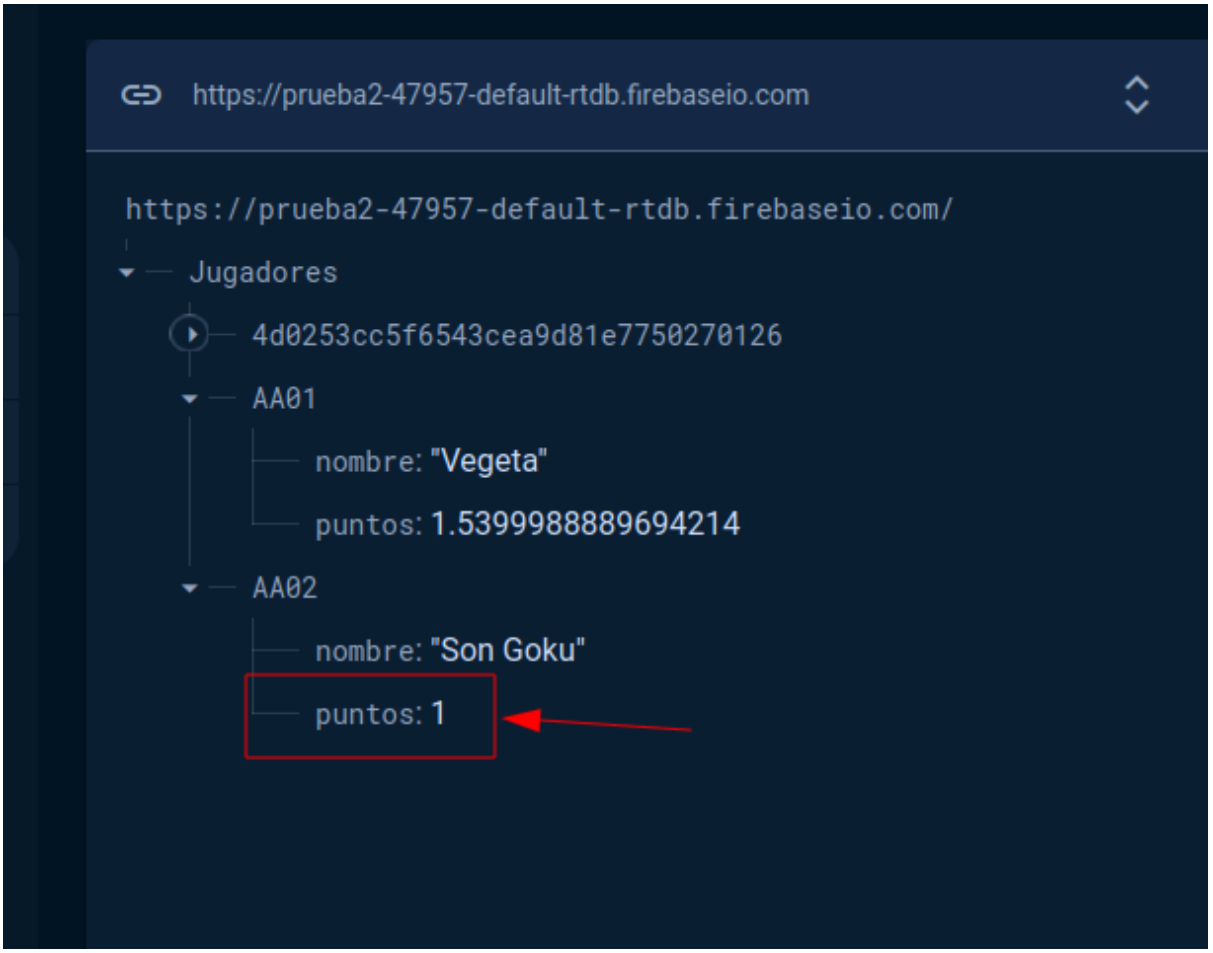


Player añadido al script

7. Añado en el proyecto el google.services.json (simplemente lo arrastro).

8. Guardar el proyecto y comprobar que funciona.

Para ello, me dirijo al firebase y cambio el parámetro `puntos`.



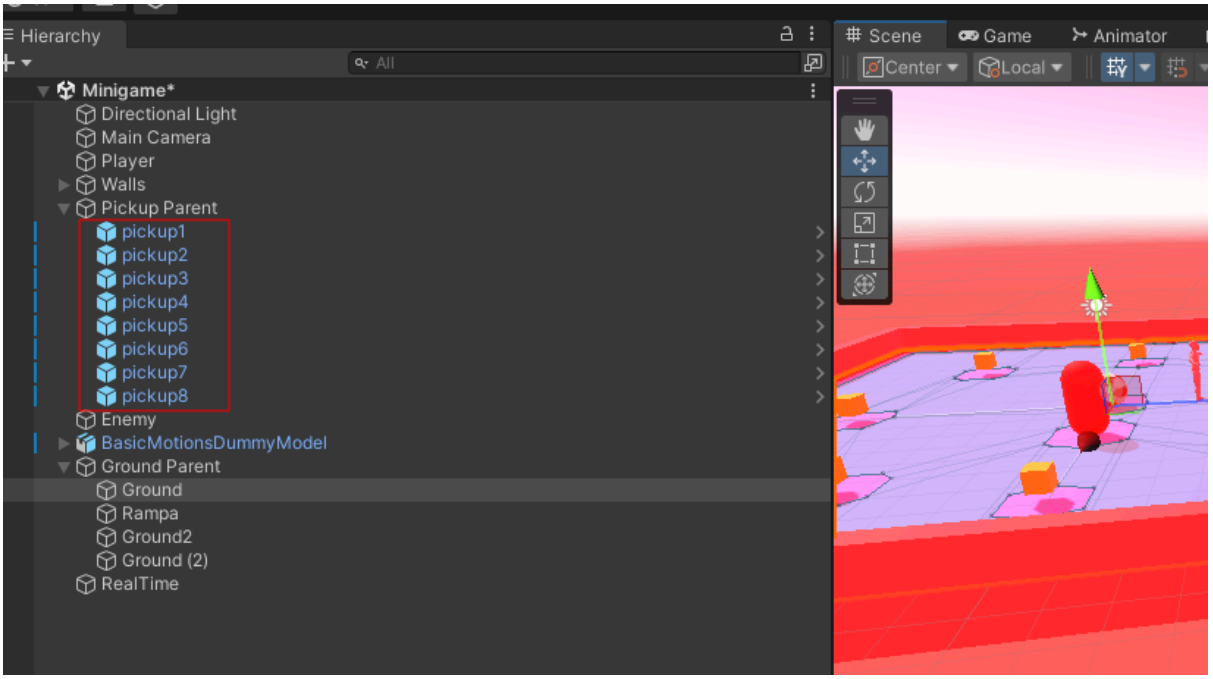
Parámetro `puntos` dentro del Firebase

Cambio el parámetro `puntos` a 15 por ejemplo, guardamos en proyecto de unity y ejecutamos. Observamos que funciona ya que el tamaño de la bola es mucho mayor nada más darle al “play”. Esto se debe a que hemos incrementado en el Firebase el parámetro `puntos`.

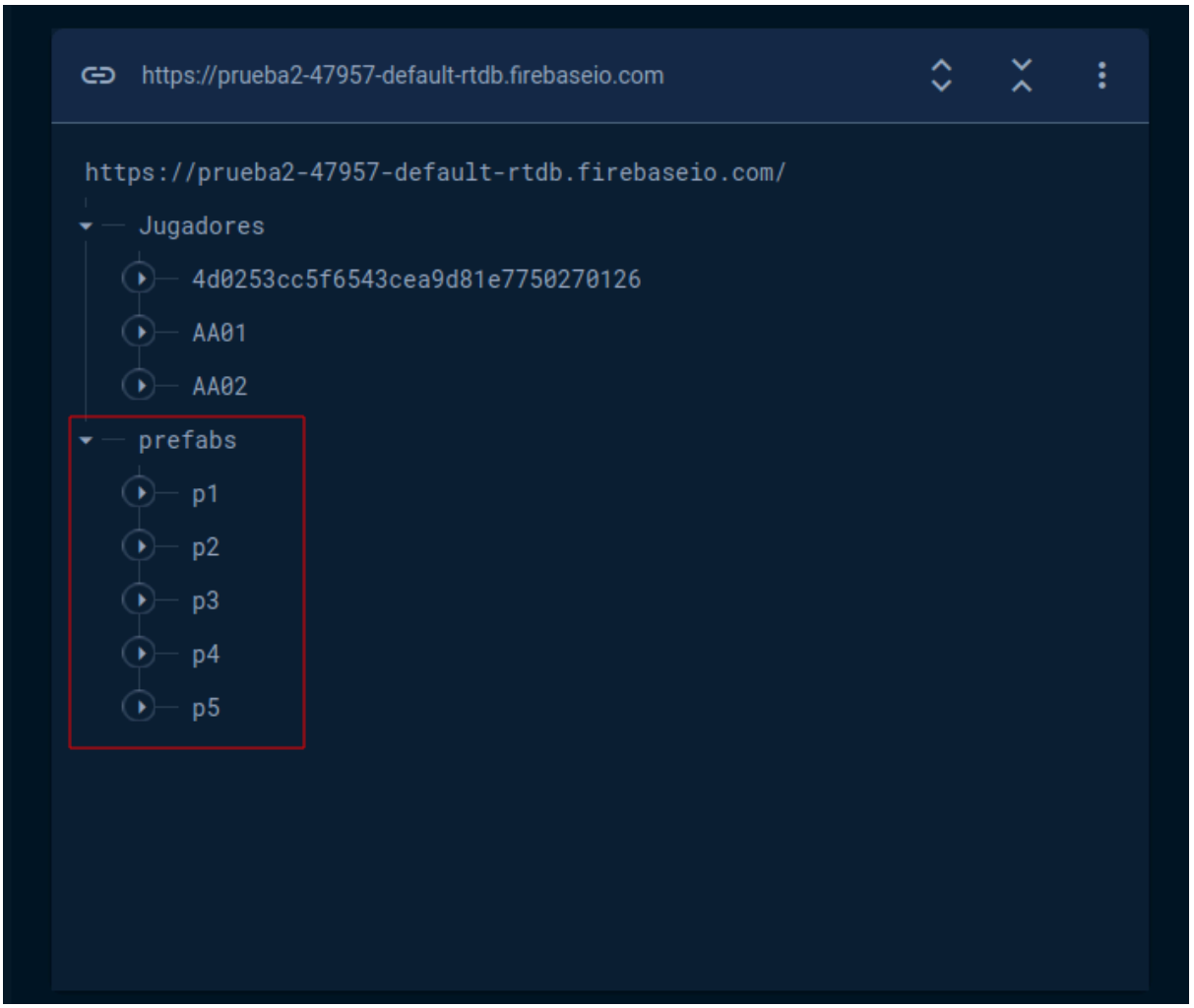
1. Posiciones en Firebase

Firebase

En la base de datos Firebase añadir nuevos valores. Estos servirán para modificar la posición de los pickups en el momento que se inicie el juego. En concreto en mi proyecto tengo un total de 8 objetos `pickup` pero para hacer este ejercicio sólo voy a representar a 5 de esos `pickup` el Firebase (simplemente para comprobar que funciona).

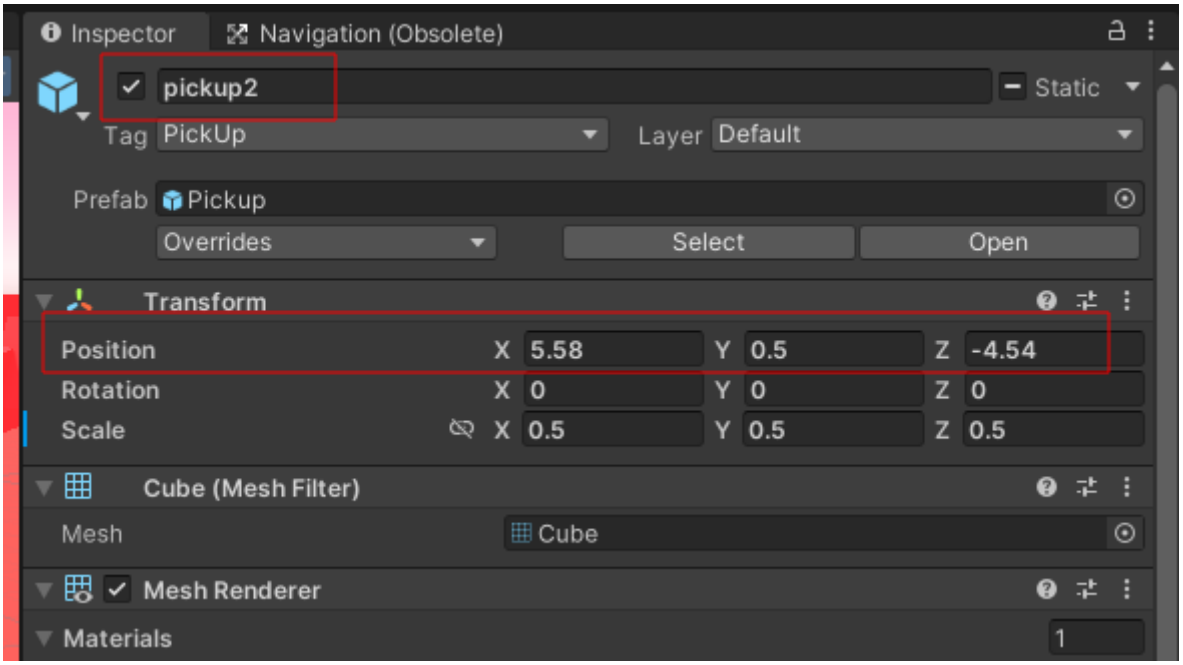


Menú Hierarchy del proyecto de Unity donde se señalan todos los objetos `pickup`

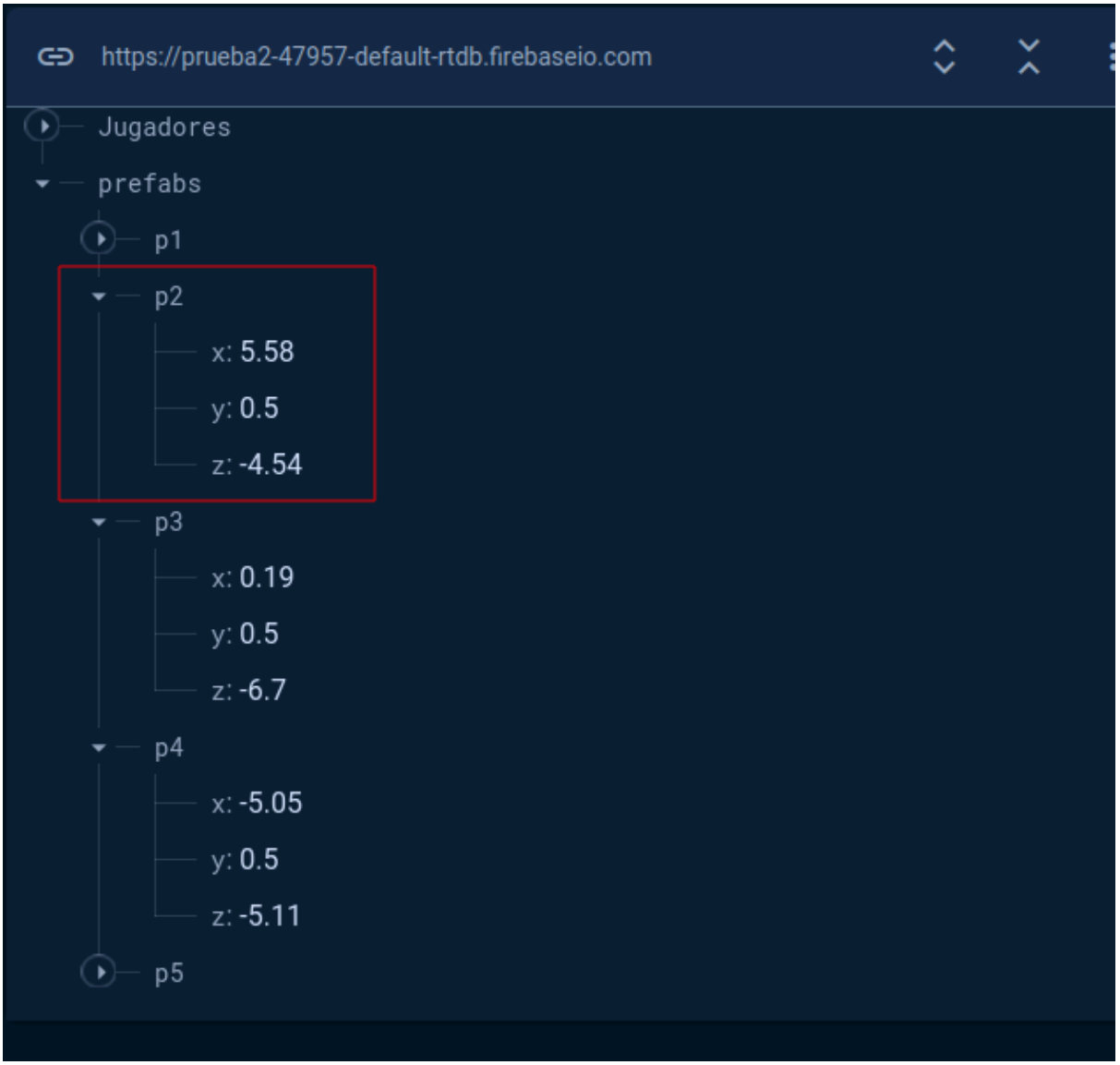


Base de datos Firebase con las variables pickup (p[i]) señalizadas

Por cada *p[i]* añado las 3 variables correspondientes al eje de coordenadas (x, y, z). El valor que le pongo a cada una de las coordenadas es el correspondiente al estado inicial de cada uno de los `pickup`.



Valores de las variables del eje de coordenadas del proyecto Unity correspondientes al objeto `pickup` llamado “pickup2”



Valores de las variables del eje de coordenadas del proyecto Firebase correspondientes al objeto `pickup` llamado "pickup2"

Script RealTime.cs

En primera instancia, hay que declarar al principio de la clase `Realtime` lo siguiente:

- Variables que referencian a `prefabs` y a `p[i]` (p1, p2, p3…) del Firebase.
- `GameObject` del objeto que vamos a modificar, que en este caso, sólo va a ser un o de los pickups, el `pickup1`.

```
private DatabaseReference _refPrefabs;
private DatabaseReference _refP1;
public GameObject pickup1;
```

Declaración de variables en el script RealTime.cs

Dentro del método `Start()` inicializo las variables anteriormente declaradas y recogemos todos los valores del Firebase correspondientes al `pickup1` (x, y, z).

El método que recoge los valores, en caso de que no haya errores durante el proceso, guarda toda la información de la variable `p1` del Firebase en `snaphost`. A continuación, llamo al metodo `RecorreResultado` el cual va a me va mostrar los valores de `snapshot` en el Debug. El siguiente paso es actualizar el valor de las coords de P1 de Unity a las del Firebase.

```
_refPrefabs = _db.GetReference("prefabs");
_refP1 = _db.GetReference("prefabs/p1");
_refPrefabs.GetValueAsync().ContinueWithOnMainThread(task => {
           if(task.IsFaulted) {
               // handle the error...
           }
           else if(task.IsCompleted) {
               DataSnapshot snapshot = task.Result;
               // mostramos los datos
               RecorreResultado(snapshot);
               //Debug.Log(snapshot.value);
           }
       });

// -- PREFABS --
// añadimos el evnto cambia un valor
_refP1.ValueChanged += HandleValueChanged_prefabs;
```

Método que guarda información de `p1` y llama a `RecorreResultado()` y métodos para actualizar posición de P1

```
   // recorro un snapshot de un nivel
   void RecorreResultado(DataSnapshot snapshot)
   {
       foreach(var resultado in snapshot.Children) // Clientes
       {
           Debug.LogFormat("Key = {0}", resultado.Key);  // "Key =
AAxx"
           foreach(var levels in resultado.Children)
           {
               Debug.LogFormat("(key){0}:(value){1}", levels.Key,
levels.Value);
           }
       }
   }
```

Método que muestra información de Debug

```
// -- PREFABS --
```

```
    // evento cambia valor en px
    // escalo objeto en la escena
    void HandleValueChanged_prefabs(object sender, ValueChangedEventArgs
args) {
        if (args.DatabaseError != null) {
            Debug.LogError(args.DatabaseError.Message);
            return;
        }
        // Mostramos lo resultados
        MuestroJugador(args.Snapshot); //?????

        // escalo objeto
        float x =
float.Parse(args.Snapshot.Child("x").Value.ToString());
        float y =
float.Parse(args.Snapshot.Child("y").Value.ToString());
        float z =
float.Parse(args.Snapshot.Child("z").Value.ToString());
        Vector3 cambioEscala = new Vector3(x, y, z);
        // cambio a position
        pickup1.transform.position = cambioEscala;

    }
```

Método que actualiza la posición de `p1`

Este ejemplo sólo lo hice con `p1` pero sería totalmente extrapolable a los demás pickup.

2. Actualizaciones en base de datos

En este apartado se tendría que actualizar la posición en el Firebase cada vez que mi Player se mueve en Unity. Sin embargo, en mi caso, voy a realizar el registro de la posición para el pickup P1, del anterior apartado, conforme el jugador se mueve. Realmente, en un caso real, no tiene sentido que el pickup se mueva con respecto al movimiento del Player, pero como ya tenemos la posición del pickup registrada en el Firebase y sólo quiero mostrar la actualización en la base de datos Firebase según la posición que tenga el Player, con eso me sirve.

Simplemente vamos al método `Update()` de nuestro script, obtenemos la posición del Player y con ello actualizamos la posición de P1. Este proceso se realiza cada frame.

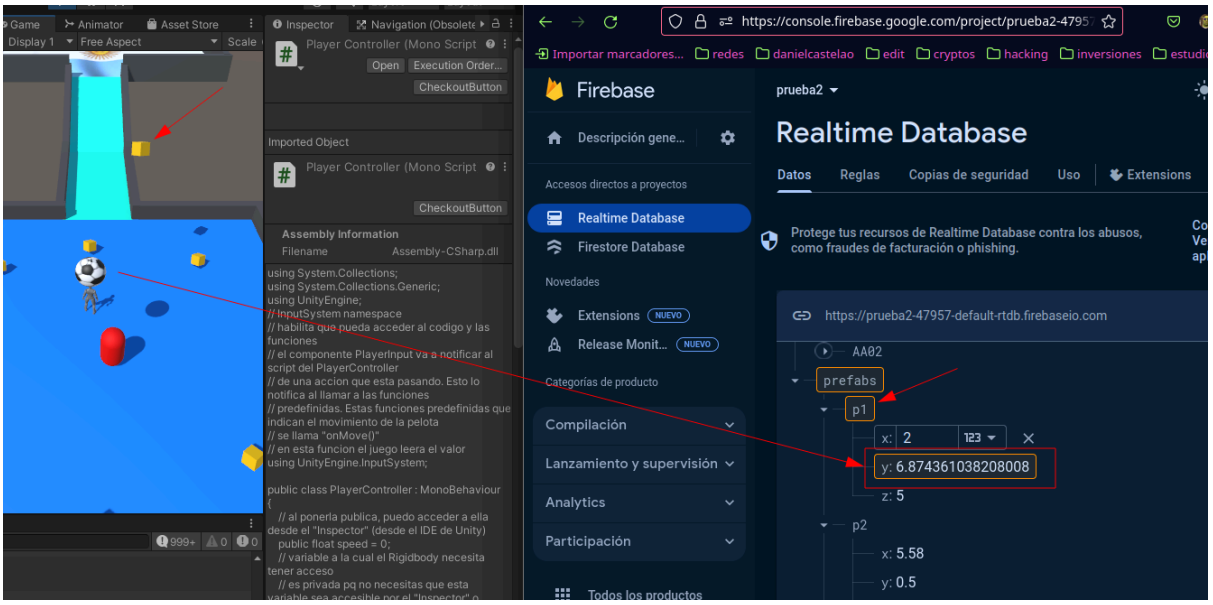
```
    void Update()
    {

        double playerx = ondavital.transform.position.x;
```

```
        double playery = ondavital.transform.position.y;

        _refPrefabs.Child("p1").Child("x").SetValueAsync(playerx+2);
        _refPrefabs.Child("p1").Child("y").SetValueAsync(playery+2);
    }
```

Actualización de P1 cada frame

Le sumamos dos a `playerx` simplemente para poder visualizar el pickup ya que si no, se vería dentro de la pelota.

En Unity observamos que simplemente al saltar con el Player, se actualiza el valor de Y en el Firebase y el pickup P1 se mueve hacia arriba.



Posición de P1 en el Firebase

3. Actualización en tiempo real

Creamos el nuevo objeto correspondiente al Player2 y hago los pasos necesarios para proporcionar movimiento a la bola.

En el Firebase añado los datos necesarios que servirán para rastrear el estado del segundo jugador.

Creamos una referencia al Player2 dentro del script. Añado las líneas de código para enlazar la posición del Player2 exactamente igual que lo hice con el Player1.