IES AUGUSTO GONZÁLEZ DE LINARES

CFGS DAM2º

DESARROLLO MÓVIL

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

Ramiro Gutiérrez Valverde

2023/2024

Índice

1. Criterios del RA1

1.1. Criterio de evaluación C: SE han identificado las tecnologías de desarrollo de aplicaciones para dispositivos móviles

En la actualidad (últimas décadas), el mercado de los dispositivos móviles se ha visto dominado por dos sistemas operativos que destacan significativamente sobre el resto. Estos son Android (Google) e IOS (Apple) y ocupan, respectivamente, el primer y segundo puesto de ventas de dispositivos a nivel mundial.

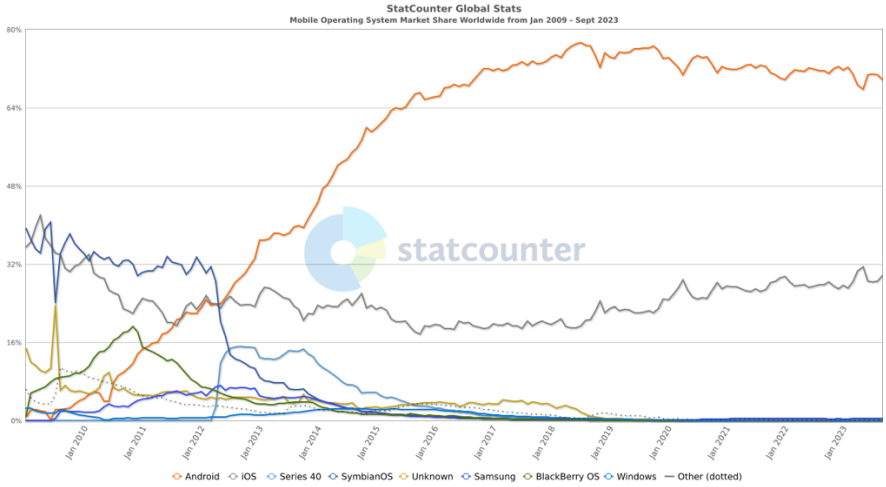

Ilustración 1.1.1. Mercado de los SO móviles desde 2009 hasta la actualidad. (Statcounter, s.f.)

Lamentablemente (sobre todo para los desarrolladores), dichos sistemas operativos se basan en lenguajes de programación diferentes, por lo que, cada uno posee una tecnología de desarrollo de aplicaciones con su propio lenguaje.

En el caso de Android, el lenguaje más utilizado es Kotlin, con el 50% de los desarrolladores, seguido de Java, utilizada por un 30% (Kotlin, s.f.) . Ambos disponen de su SDK. Java es un lenguaje multiplataforma per se, y Kotlin, recientemente, ha lanzado (en beta) el Kotlin Multiplatform, que pretende simplificar el desarrollo de aplicaciones mientras gana usuarios, frente a otros lenguajes más cerrados como puede ser el de IOS. El IDE más usado para Android es el Android Studio.

Apple utiliza Swift para el desarrollo de aplicaciones para todos sus dispositivos. Es un lenguaje que no permite migración a otros SO y que sólo puede ser desarrollado con el IDE Xcode, que funciona exclusivamente dentro del ecosistema de Apple. (Apple, s.f.)

Sin embargo, utilizando otros lenguajes como C#, Javascript o Dart, utilizando framewors como React Native, Flutter o Xamarin, se pueden crear aplicaciones multiplataforma. (Armadillo amarillo, s.f.)

1.2. Criterio de evaluación D: SE han instalado, configurado y utilizado entornos de trabajo para el desarrollo de aplicaciones para dispositivos móviles.

Como he mencionado antes, ambas plataformas poseen sus IDEs nativos pero, debido a que nuestro módulo es multiplataforma, nosotros utilizamos Unity, que es un motor de videojuegos que permite hacer builds para ambos sistemas.

He instalado el Unity Hub en dos equipos con diferentes SO. En el de clase, con Windows y en mi ordenador personal con MacOS. Desde el Unity Hub se puede controlar la instalación de las diferentes versiones del Unity Editor (se recomienda instalar las versiones LTS), así como los módulos necesarios para la creación y posterior exportación (o build), a las diferentes plataformas que se desee.

Uno de los módulos instalables es el Android Build Support, que viene con su JDK y SDK incluidos.

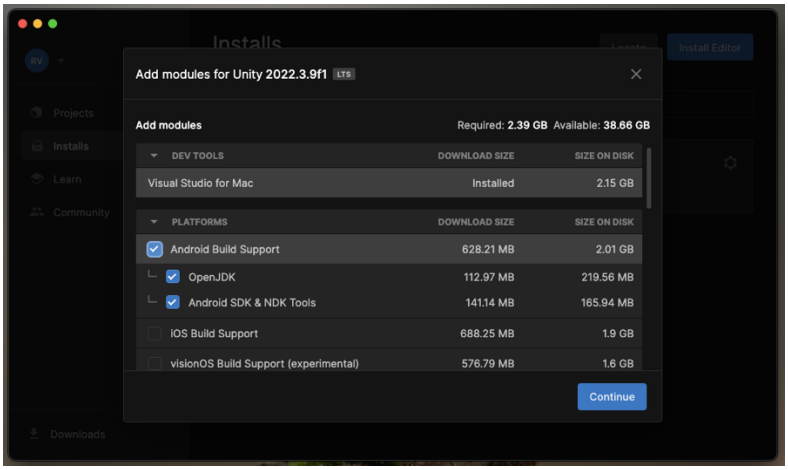

Ilustración 1.2.1. Pantalla de instalación de módulos de Unity Hub.

1.3. Criterio de evaluación E: Se han identificado configuraciones que clasifican los dispositivos móviles en base a sus características.

La clasificación de los dispositivos móviles según sus características más importantes podría resumirse en: Dispositivos de propósito general y dispositivos de entretenimiento.

En el primer grupo encontramos dispositivos ampliamente utilizados como teléfonos inteligentes, tabletas, phablets (término horrible utilizado para referirse a mezcla entre teléfono y tableta), y ordenadores portátiles. Cada vez es más difícil distinguir entre unos y otros. A su vez se clasifican por su tamaño y tipo de pantalla, procesador, sistema operativo, memoria, etc.

En los dispositivos de entretenimiento nos encontramos con las consolas portátiles. Algunas de las más conocidas del mercado en la actualidad son la Nintendo Switch (con juegos propios de la plataforma), la Steam Deck y la Razer Edge (que permiten ejecutar juegos de

escritorio), la Anbernic RG35XX (que permite emular juegos retro) o la Asus Rog Ally (que directamente funciona sobre Windows 11). (ComputerHoy, s.f.)

1.4. Criterio de evaluación G: se ha analizado la estructura de aplicaciones existentes para dispositivos móviles identificando las clases utilizadas.

Me referiré a la experiencia de la creación del primer juego en Unity:

Las clases en Unity se corresponden con los Scripts. Como se ve en la ilustración, yo he utilizado 4 diferentes:

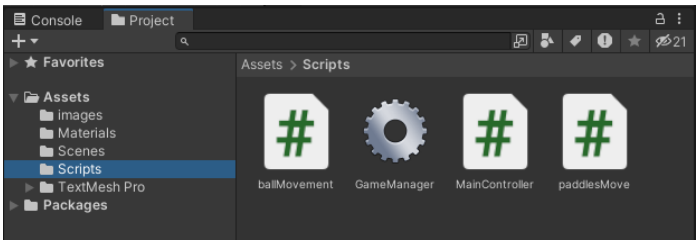

Ilustración 1.4.1. Scripts del proyecto de Unity.

BallMovement: controla el movimiento de la bola mediante métodos que dirigen el lanzamiento de esta y la respuesta ante las colisiones.

PaddlesMove: controla el movimiento independiente de las palas, definiendo su rango y velocidad y cómo interactúan con la bola al chocar.

Game manager: inicia el juego, controla pausas, puntos y marcadores.

MainController: me sirve para cambiar entre escenarios y salir de la aplicación.

1.5. Criterio de evaluación H: Se han realizado modificaciones sobre aplicaciones existentes.

He añadido dos escenas al juego. Un menú principal desde el que se puede elegir jugar, salir o ver los controles, y la escena en la que se explican dichos controles, desde la que se puede jugar o volver al menú principal (ilustraciones 1.5.1 y 1.5.2).

He añadido dos métodos que detienen el juego por 1 y 3 segundos respectivamente, para que el jugador tenga tiempo de reaccionar, al empezar el juego y al reiniciar tras marcar punto (ilustración 1.5.3).

También, como pequeño complemento a la jugabilidad, la pelota aumenta de velocidad cada vez que choca con una pala (ilustración 1.5.4).

Ilustración 1.5.1. Menú principal Pong.

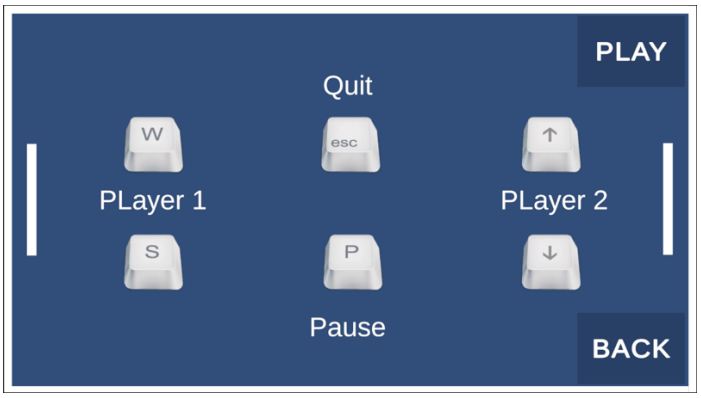

Ilustración 1.5.2. Menú de controles del Pong.

```
IEnumerator waitOneSec()
{
    ball.GetComponent<Transform>().SetPositionAndRotation(new
Vector3(0, 0, 0), transform.rotation);
    ball.GetComponent<Rigidbody2D>().velocity = Vector3.zero;
    yield return new WaitForSeconds(1);
    ball.GetComponent<ballMovement>().launch();
}
IEnumerator WaitFor3Secs()
{
    ball.GetComponent<Transform>().SetPositionAndRotation(new
Vector3(0, 0, 0), transform.rotation);
    ball.GetComponent<Rigidbody2D>().velocity = Vector3.zero;
    yield return new WaitForSeconds(3);
    ball.GetComponent<ballMovement>().launch();
}
```

Ilustración 1.5.3. Métodos para retardar el lanzamiento de la bola.

```
private void OnCollisionEnter2D(Collision2D collision2)
{
   if(collision2.gameObject.tag.Contains("Paddle"))
   {
      ballBody.velocity *= speedRise;

   }
}
```

Ilustración 1.5.4. Método que aumenta la velocidad de la bola tras impacto.

1.6. Criterio de evaluación I: SE han utilizado emuladores para comprobar el funcionamiento de las aplicaciones.

El propio programa de Unity tiene una pestaña en la que se despliega un menú que te permite escoger en qué tipo de dispositivo deseas realizar la emulación. He probado un par de dispositivos y lo he documentado en las siguientes ilustraciones.

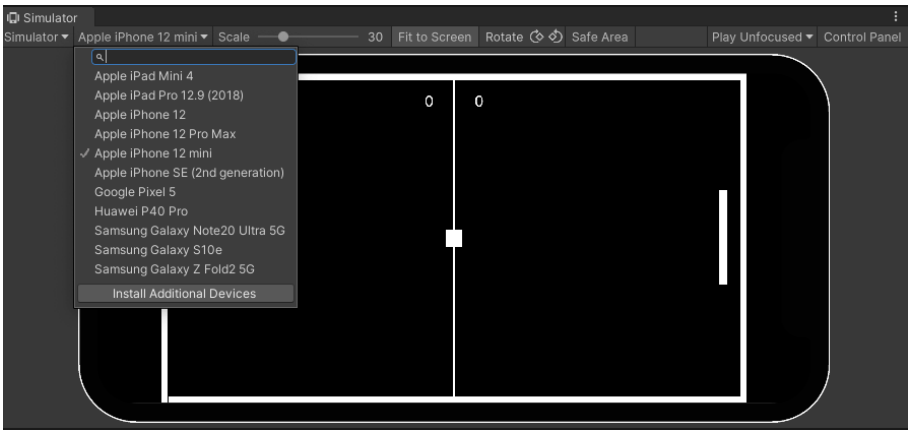

Ilustración 1.6.1. Ventana del simulador desplegada. Se observan las diferentes opciones.

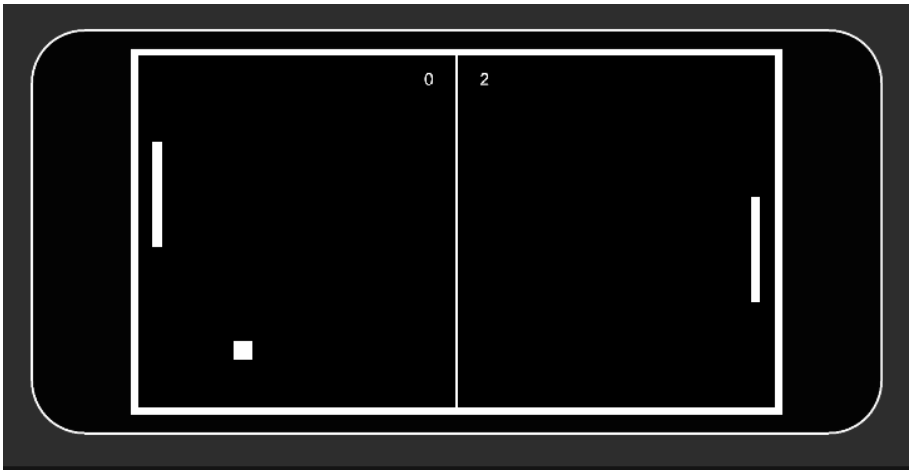

Ilustración 1.6.2. Ejecución del juego en el simulador del iPhone SE (segunda generación).

2. Criterios del RA 2

2.1. Criterio de evaluación A: Se ha generado la estructura de clases necesaria para la aplicación.

En el siguiente diagrama se muestra la relación entre los diferentes componentes, escenas y botones, y qué controla cada uno de ellos. Como dije anteriormente, he creado 3 escenas que se comunican entre ellas para el manejo del juego.

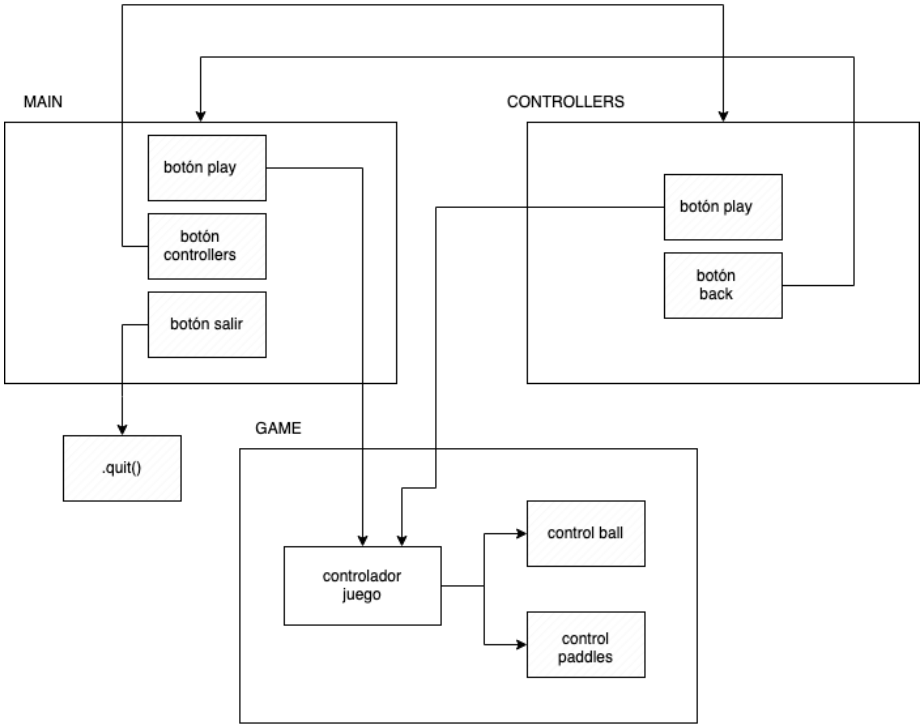

Ilustración 2.1.1. Diagrama que muestra la relación entre objetos del juego Pong.

2.2. Criterio de evaluación B: Se han analizado y utilizado las clases que modelan ventanas, menús, alertas y controles para el desarrollo de aplicaciones gráficas sencillas.

He creado dos gameObjects de tipo canvas utilizando elementos UI, que necesito para modelar los dos menús de los que he venido hablando hasta ahora.

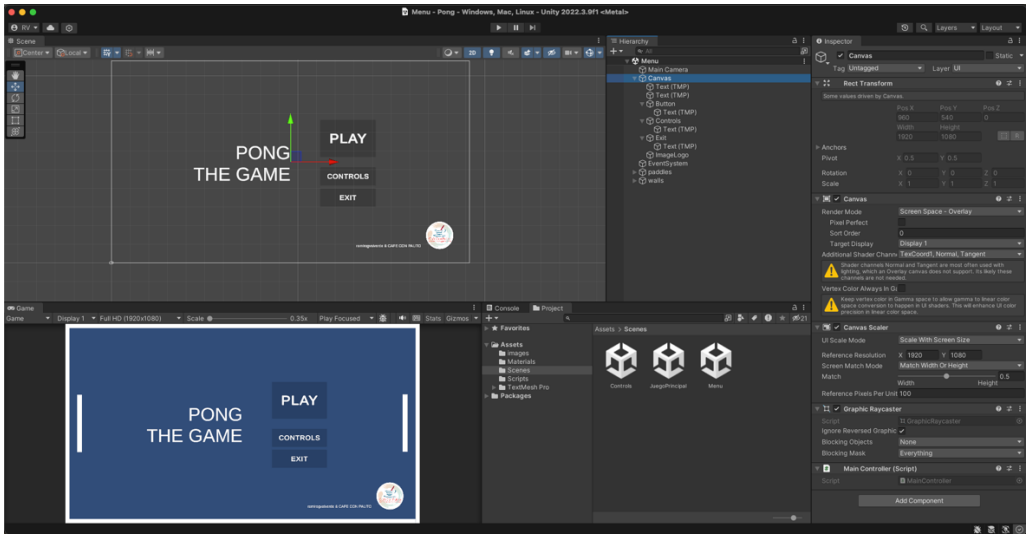

Ilustración 2.2.1. Captura de pantalla del programa en la que se aprecia el diseño del menú, sus componentes y su jerarquía.

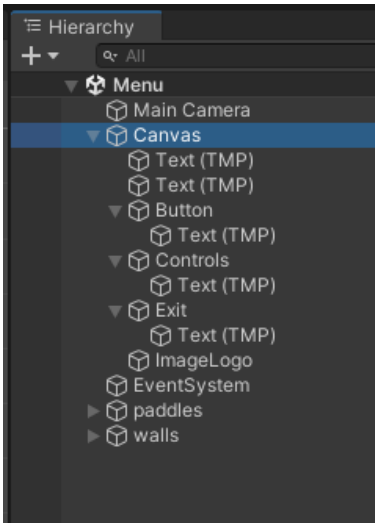

Ilustración 2.2.2. Jerarquía de elementos del menú. Elemento Canvas de UI con sendos botones y textos.

2.3. Criterio de evaluación C: Se han utilizado las clases necesarias para la conexión y comunicación con dispositivos inalámbricos.

Unity. Algunas de las clases utilizadas para la conexión inalámbrica:

NetworkManager: Esta clase proporciona una interfaz para administrar la conectividad de red en Unity. Puedes usarla para configurar y administrar conexiones inalámbricas en tu juego.

NetworkDiscovery: Esta clase te permite descubrir y conectarte a otros dispositivos en la red local. Puedes usarla para buscar y unirte a sesiones de juego inalámbricas.

Android Studio. Estas son algunas clases y métodos importantes que se pueden utilizar:

WifiManager: Esta clase te permite acceder y controlar el estado de la conexión Wifi en el dispositivo. Métodos:

- getWifiState(): Devuelve el estado actual de la conexión Wifi.
- isWifiEnabled(): Verifica si la conexión Wifi está habilitada.
- setWifiEnabled(boolean enabled): Habilita o deshabilita la conexión Wifi.

WifiInfo: Esta clase te proporciona información detallada sobre la conexión Wifi actual. Métodos:

- getSSID(): Devuelve el nombre de la red Wifi a la que está conectado el dispositivo.
- getIpAddress(): Devuelve la dirección IP asignada al dispositivo en la red Wifi.
- getLinkSpeed(): Devuelve la velocidad de conexión en Mbps.

WifiConfiguration: Esta clase te permite configurar y administrar las redes Wifi disponibles. Métodos:

- addNetwork(WifiConfiguration config): Agrega una nueva red Wifi a la lista de redes configuradas.

- enableNetwork(int networkId, boolean disableOthers): Habilita una red WIFI específica.
- disableNetwork(int networkId): Deshabilita una red Wifi específica.

En la documentación de Unity se pueden encontrar más indicaciones. (Unity , s.f.)

2.4. Criterio de evaluación D: Se han utilizado las clases necesarias para el intercambio de mensajes de texto y multimedia.

Unity. Con las clases NetworkManager y NetworkMessages podemos mandar mensajes en red.

Android Studio. Con los paquetes android.telephony.VisualVoicemailSms para SMS y android.provider.Telephony.Mms para MMS se pueden realizar comunicaciones entre dispositivos.

2.5. Criterio de evaluación E: Se han utilizado las clases necesarias para establecer conexiones y comunicaciones HTTP y HTTPS.

Unity. Utilizando la clase UnityWebRequest se pueden crear tres tipos de objetos, con varios métodos importantes cada uno.

UploadHandler. Maneja la transmisión de datos al servidor.

DownloadHandler. Maneja la recepción de dichos datos.

UnityWebRequest. Maneja los objetos anteriores con métodos para enviar y recibir datos a través de HTTP, como GET, POST, PUT, DELETE, etc. (Unity, s.f.)

2.6. Criterio de evaluación F: Se han utilizado las clases necesarias para establecer conexiones con almacenes de datos garantizando la persistencia.

La clase que utilizamos para las conexiones con la base de datos SQL es la Sqlite, para lo que necesitamos realizar una serie de importaciones y configurar una serie de variables. Incluyo un ejemplo de implementación encontrado en el foro de la comunidad de Unity del que muestro unas pequeñas pinceladas. La página enlazada es muy útil. (dklompmaker, s.f.)

```
Import        System.Data; //se importa la clase Data
Import        Mono.Data.SqliteClient; // importamos el cliente sqlite
```

// variables de conexión

private var connection : String;

private var dbcon : IDbConnection;

private var dbcmd : IDbCommand;

private var reader : IDataReader;

The Class

Code (csharp):

```
1.
2. import          System.Data;  // we import our  data class
3. import          Mono.Data.SqliteClient; // we import our sqlite client
4.
5. class dbAccess {
6.     // variables for basic query access
7.     private var connection : String;
8.     private var dbcon : IDbConnection;
9.     private var dbcmd : IDbCommand;
10.    private var reader : IDataReader;
11.
12.    function OpenDB(p : String){
13.    connection = "URI=file:" + p; // we set the connection to our database
14.    dbcon = new SqliteConnection(connection);
15.    dbcon.Open();
16.    }
17.
18.    function BasicQuery(q : String, r : boolean){ // run a baic Sqlite query
19.        dbcmd = dbcon.CreateCommand(); // create empty command
20.        dbcmd.CommandText = q; // fill the command
21.        reader = dbcmd.ExecuteReader(); // execute command which returns a reader
22.        if(r){ // if we want to return the reader
23.        return reader; // return the reader
24.        }
25.    }
26.
```

Ilustración 2.6.1. Captura de pantalla del foro en el que se muestra la implementación de una base de datos.

2.7. Criterio de evaluación G: Se han realizado pruebas de interacción usuario- aplicación para optimizar las aplicaciones desarrolladas a partir de emuladores.

Al realizar las diferentes pruebas en los distintos emuladores y tras “buildear” y ejecutar en diferentes plataformas, me vi forzado a realizar ajustes. En primer lugar, no era consistente con el tamaño de pantalla y su aspecto o ratio en toda la aplicación y tuve que corregirlo.

Después, algunos de los elementos se descolocaban al escalarse y, tras mucho probar y algún que otro tutorial descubrí que los elementos están anclados a una parte de la escena. Simplemente cambiando el anclaje se puede evitar que esos errores sucedan.

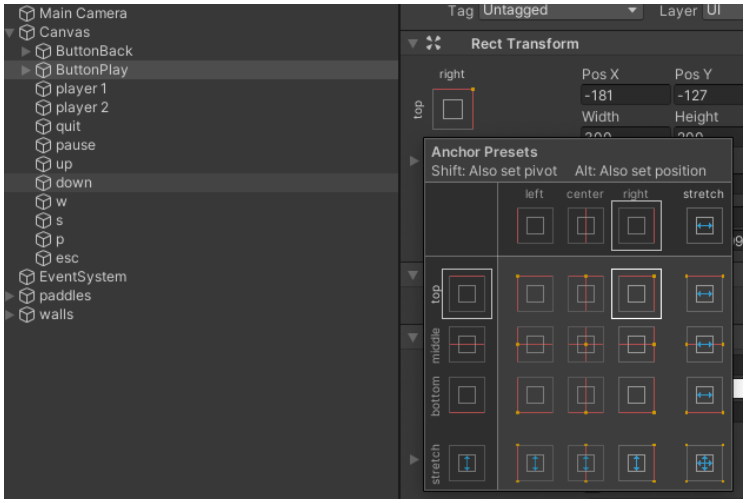

Ilustración 2.7.1. Modificando el punto del anclaje del botón ButtonPlay.

2.8. Criterio de evaluación H: Se han empaquetado y desplegado las aplicaciones desarrolladas en dispositivos móviles reales.

Mi móvil es un iPhone y no he podido desplegar nada para esa plataforma. Pero sí que he realizado builds para MacOS , Windows y Android, de una forma un poco sobria porque no he cambiado siquiera los iconos del proyecto. Los ejecutables de MacOS y Windows funcionan correctamente y, aunque es muy sencillo, es entretenido.

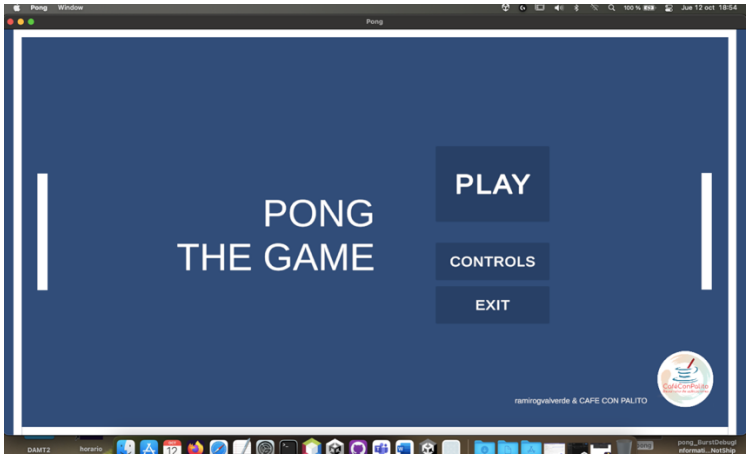

Ilustración 2.8.1 Ejecución del juego en macOS Monterrey.

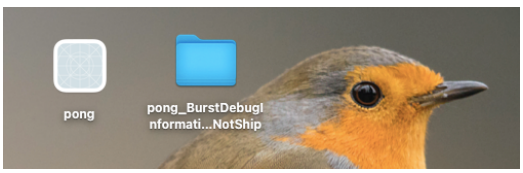

Ilustración 2.8.2 Archivos generados en el build para macOS.

En Android me he limitado a realizar unas fotografías que capturan la ejecución del juego en sus diferentes escenas.

Ilustración 2.8.3. Fotografía de la primera escena.

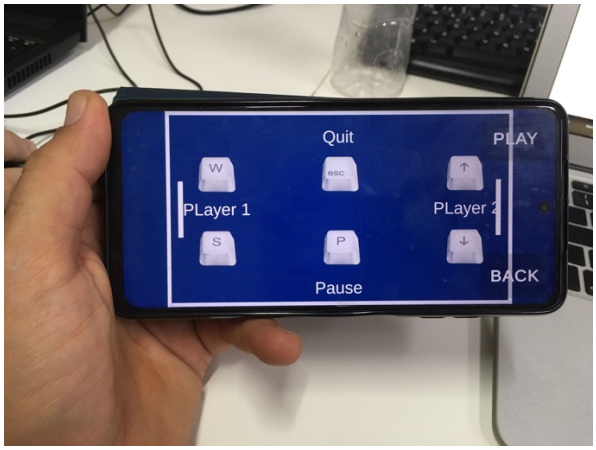

Ilustración 2.8.4. Fotografía de la segunda escena. Se aprecian los botones descolocados.

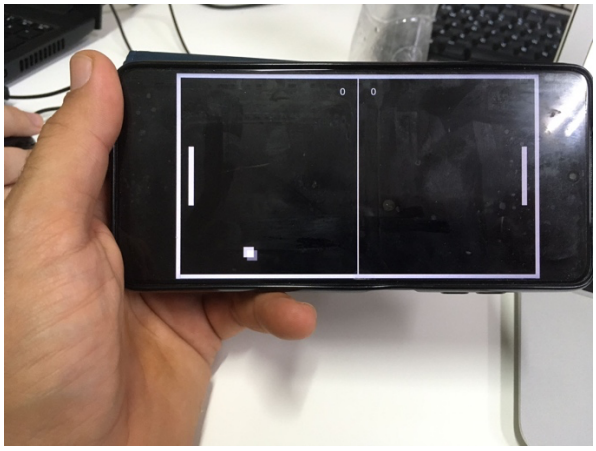

Ilustración 2.8.5. Fotografía del juego en ejecución.

Bibliografía

Apple, n.d. *Apple Swift.* [Online]
Available at: https://www.apple.com/es/swift/

Armadillo amarillo, n.d. *Armadillo amarillo.* [Online]
Available at: https://www.armadilloamarillo.com/blog/mejores-tecnologias-para-desarrollo-de-aplicaciones-moviles-2022/#tecnologiaapphibrido

ComputerHoy, n.d. *Computer Hoy.* [Online]
Available at: https://computerhoy.com/reportajes/mejores-consolas-portatiles-ofertas-descuentos-915015

dklompmaker, n.d. *Unity. Foro.* [Online]
Available at: https://forum.unity.com/threads/sqlite-class-easier-database-stuff.28500/

Kotlin, n.d. *Kotling for android.* [Online]
Available at: https://kotlinlang.org/docs/android-overview.html

Statcounter, n.d. *Statcounter GlobalStats.* [Online]
Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202309

Unity , n.d. *Unity documentation.* [Online]
Available at: https://docs.unity3d.com/es/2018.4/Manual/UNet.html

Unity, n.d. *Unity Documentation UnityWebRequest.* [Online]
Available at: https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html

Tabla de ilustraciones