Proyecto Fin de Ciclo

Debugging The History

Videojuego estilo Trivial/RPG para plataformas Android

Autor: Iván Juárez Rincón
Tutora: Patricia de Andrés
Ciclo: Desarrollo de Aplicaciones Multiplataforma
Centro: IES Enrique Tierno Galván (Madrid)
Fecha: 07/06/2021

Sumario

1 - Motivación

1.1 - Idea inicial y concepción del proyecto

La idea inicial que dio forma a mi proyecto, Debugging The History, al cual llamaré DTH para abreviar, se basa en la inexistencia de aplicaciones/juegos que enseñen y lleven temas históricos al gran público.

Adicionalmente, me di cuenta de que prácticamente todos los videojuegos de tipo Trivial existentes en el mercado actualmente (“Preguntados”, “¿Quién quiere ser millonario?”, etc.) carecen, a mi parecer, de un componente que incite a seguir jugando durante un tiempo prolongado, puesto que la gran mayoría de sus usuarios solo utilizan este tipo de aplicaciones para cubrir un interés inmediato y pasajero, ya sea porque un amigo le ha retado a una partida puntual o porque vio el juego anunciado y decidió probarlo.

Debido a los puntos anteriores decidí crear DTH, el cual es un juego estilo Trivial cuyas preguntas se centran únicamente en temas de historia, pero que cuyo núcleo central no son las preguntas, sino la historia propia y original que se encargará de contar, dándole al jugador un motivo para seguir jugando mas allá del interés básico que generan juegos de este estilo.

Además, el personaje que encarna el jugador deberá pelear contra diferentes enemigos en cada fase del juego, siendo las preguntas el medio para ello, ya que cada pregunta acertada hará que el jugador ataque al rival, mientras que los fallos harán lo propio pero por parte del enemigo.

El objetivo del proyecto es, por tanto, desarrollar un videojuego accesible y fácil de comprender, cuya jugabilidad estará basada en la contestación de preguntas sobre diversos temas históricos y que constará de elementos RPG como la subida de estadísticas (vida, ataque, etc.) y el desarrollo de una trama propia.

1.2 - Tecnologías utilizadas

Para la realización de este proyecto se utilizarán las siguientes tecnologías:

- Unity 3D: Se trata de un potente motor grafico que consta de todas las herramientas necesarias para la creación de aplicaciones y videojuegos en diversas plataformas, entre ellas Android.
- C#: Es un lenguaje de programación basado en objetos, muy similar a Java. Es el lenguaje utilizado por Unity 3D.
- SQL: Es un lenguaje de gestión de bases de datos que sirve para realizar consultas y modificaciones en bases de datos que lo utilizan.
- SQLite: Sistema de gestión de bases de datos escrito en C y que es utilizado principalmente en entornos móviles como IOS y Android. La base de datos del videojuego estará contenida aquí.

*Estas tecnologías y utilidades serán explicadas en detalle mas adelante.

2 - Planificación y Seguimiento

2.1 - Metodología de Desarrollo

En lo referente a la planificación y la metodología a utilizar durante el desarrollo del proyecto, he decidido usar una metodología de desarrollo software de las llamadas “ágiles” debido, principalmente, a que:

- Dispongo de poco tiempo para el desarrollo del proyecto, por lo que los hitos y avances del mismo deberán realizarse en periodos cortos de tiempo.
- Durante la realización del proyecto se irán generando versiones funcionales del mismo, las cuales me permitirán visualizar los progresos y darme cuenta de los fallos y errores que haya cometido. Estas versiones no serán versiones completas del juego sino que comenzarán teniendo unicamente los componentes más básicos de este y, conforme avance el desarrollo, su contenido aumentará hasta alcanzar la versión final del juego.
- Unity funciona mediante la creación de escenas, las cuales contienen las diversas pantallas y elementos del juego. Una metodología ágil me permitirá crear versiones básicas de cada escena de juego e ir probando sus funcionalidades sin necesidad de tenerlas completamente terminadas.

La forma de trabajo que he elegido es una versión simplificada de la metodología **SCRUM**. Está metodología consiste en:

- La realización de ciclos temporales cortos (desde 1 semana hasta 2 meses) de duración fija en los cuales se deben realizar las tareas que hayan sido planificadas para esa iteración del ciclo de desarrollo.
- Entregar al cliente una versión funcional del producto en un periodo de tiempo muy corto, obteniendo resultados rápidamente y realizando una revisión del producto y de los objetivos alcanzados al final de cada iteración para comprobar que lo obtenido es del agrado del cliente.
- Eliminar los imprevistos/sorpresas generadas por un mal entendimiento de las necesidades del cliente, puesto que este está viendo el progreso del proyecto en todo momento.

Para la realización de este proyecto voy a utilizar la planificación por iteraciones y la generación de versiones intermedias del juego. He de dejar fuera todos aquellos puntos de está metodología que impliquen el trabajo en equipo y las reuniones con el mismo, ya que este proyecto es unipersonal y, por lo tanto, no existe un equipo de desarrollo como tal.

2.2 - Planificación

La planificación de este proyecto estará dividida en ciclos de desarrollo (iteración) de 1 semana de duración, en los cuales se realizarán las tareas planificadas en cada uno de ellos. Las tareas se irán planificando según se vaya avanzando en el proyecto, empezando por los elementos más básicos en las primeras iteraciones y avanzando a funciones mas complejas en los siguientes ciclos.

Se partirá de una lista de tareas a realizar y en cada iteración se elegirá una o varias de ellas para su realización durante dicho ciclo. El desarrollo de estas tareas y el tiempo empleado para cada una será detallado mas adelante, en el apartado de **Diario de Desarrollo**.

LISTA DE TAREAS	
Tarea	**Observaciones**
Base de Datos	Esta tarea se realizara en varias partes, creando las tablas según se vayan necesitando
Escena de Juego	Deberá poder cargar las preguntas, comprobar las elecciones del jugador y controlar la vida de los personajes.
Animaciones	Se añadirán a la escena de juego e irán cambiando en función de las acciones que se estén realizando (esperar, atacar, recibir daño, etc)
Escena Pantalla de Inicio	En principio solo deberá contener el logo del juego y un botón para entrar al mismo
Escena Cinemática	En esta escena se contará la historia del juego mediante conversaciones entre personajes.
Escena Menú Principal	Este será el lugar al que los jugadores volverán después de cada partida. Aquí podrán mejorar las estadísticas del personaje, revisar las cinemáticas que ya hayan desbloqueado y cargar el siguiente nivel del juego.
Transiciones	Servirán para evitar que los cambios entre escenas sean bruscos.
Audio	Sonidos del juego.
Documentación	Esta tarea se realiza durante todo el desarrollo.

Las tareas detalladas en la lista anterior no tienen por que completarse al 100% antes de pasar a la siguiente, pues habrá funcionalidades de las mismas que no se podrán implementar hasta que otras tareas sean completadas hasta cierto punto. Por ejemplo, los cambios entre escenas y el control de la base de datos no estarán implementados del todo hasta que todas las escenas estén terminadas.

2.3 - Tablero SCRUM

Para llevar un control del progreso de cada tarea y tener una visión global del estado del proyecto he utilizado un Tablero SCRUM, cuyo fin principal es el de organizar el estado de cada una de las tareas que componen el proyecto.

Ilustración 1: Captura del Tablero Scrum del Proyecto

Secciones del tablero:

- Abiertas: Aquí se colocan todas las tareas al momento de comenzar el proyecto. Es el estado base para todas las tareas puesto que aun no se ha comenzado a trabajar en ellas.
- En progreso: Las tareas que estén aquí se encuentran en progreso actualmente, lo cual significa que se esta trabajando en ellas pero aun no están completadas.
- Revisión: Cuando una tarea ha sido completada, pasa a la fase de revisión, en la cual se comprueba que el funcionamiento de sus componentes es el esperado. Si todo esta en orden, la tarea ha sido completada con éxito, mientras que si se encuentran fallos en ella se la volverá a colocar en la columna de “En progreso”.
- Listo: Una vez completada y revisada, la tarea es colocada en esta sección para indicar que ya se ha terminado de trabajar con ella.

Hay que tener en cuenta que, aunque una tarea sea completada y se encuentre en la sección de “Listo”, esta puede volver a ser puesta en desarrollo si aparecen mejoras y añadidos que se le quieran agregar o por que se descubran fallos en otras partes del proyecto que afecten a esa tarea en particular.

* La utilidad empleada para la realización del tablero, así como para el seguimiento del proyecto esta incluida en la bibliografía de este documento.

3 - Estructura y Funcionamiento

3.1 - Esquema General

En el esquema siguiente se puede observar como esta estructurado el juego, ademas de la forma en la que interactúan los diferentes componentes que forman parte del mismo:

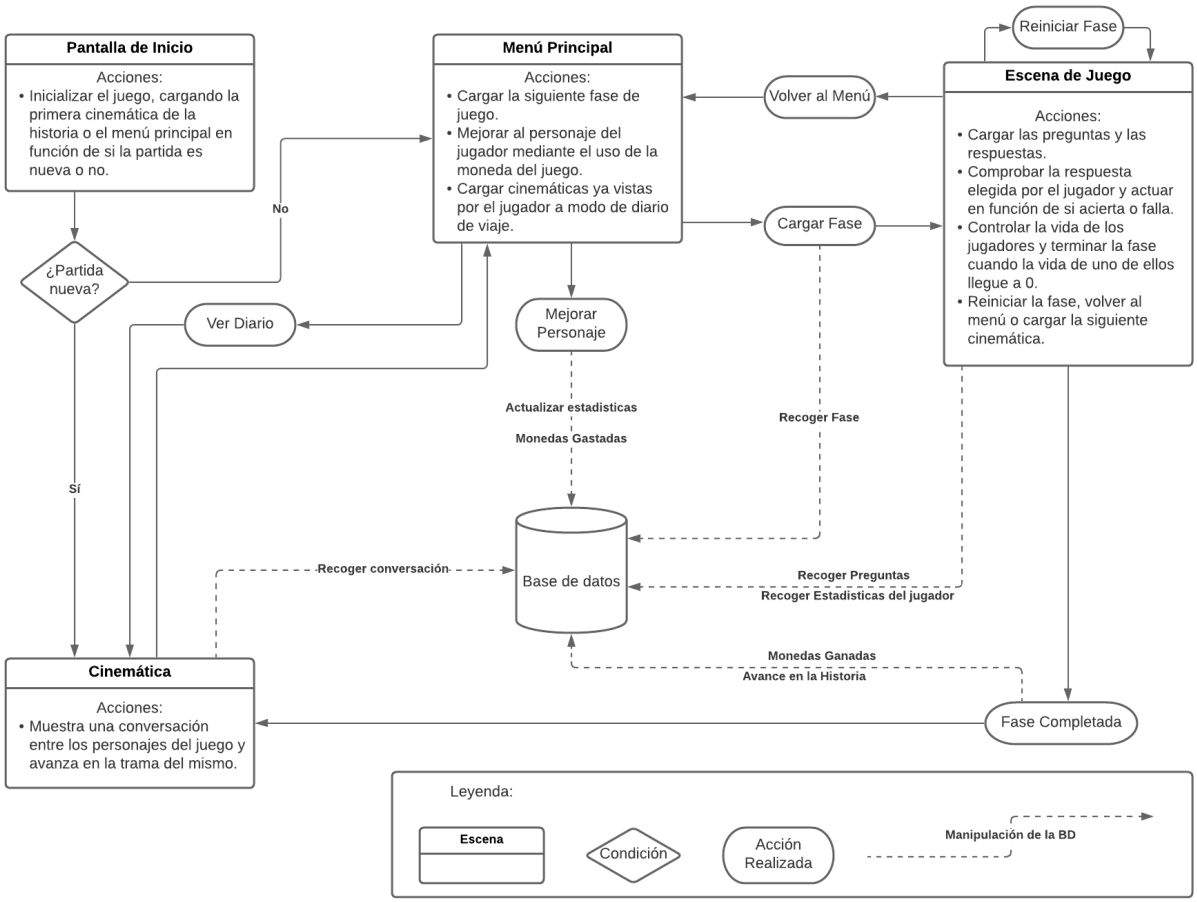

Ilustración 2: Esquema Relacional de los Componentes del Proyecto

Como se puede observar, el flujo del juego para una partida nueva es el siguiente:

1. Se abre la pantalla de inicio y el jugador comienza la partida.
2. Se lanza la primera cinemática de la historia del juego, en la que se le pone en situación al jugador. Tras esto, se abre la escena de juego.
3. En la escena de juego, si el jugador gana se le permitirá avanzar en la trama del juego, lanzando la siguiente cinemática. Si el jugador pierde puede elegir entre cargar de nuevo la fase o volver al menú principal.
4. Tras ganar el nivel o decidir dejarlo por el momento, el jugador volverá al menú principal. Aquí puede elegir mejorar las estadísticas de su personaje, ver conversaciones anteriores o ir a la siguiente fase.

3.2 - Estructura de la Base de Datos

Como ya se mencionó antes, para el acceso a la base de datos desde el sistema Android he decidido utilizar el sistema gestor de bases de datos llamado SQLite.

Las razones por las que he elegido esta utilidad para mi proyecto son las siguientes:

- Se trata de un sistema muy liviano y cuyo proceso de implementación al proyecto es sencilla.
- La base de datos del juego será local, de forma que no se necesitará una conexión a Internet para poder jugar.
- Es una utilidad de dominio público, lo cual significa que su uso es gratuito y libre.
- Utiliza el lenguaje de consultas SQL, el cual es el lenguaje que se ha utilizado a lo largo del curso y, por lo tanto, no necesito obtener conocimientos adicionales para poder usarlo.

La base de datos se estructura en las siguientes tablas:

CONVERSACIONES		
Campo	**Tipo**	**Descripción**
Orden	Text	Indica la posición que ocupa la frase dentro de la conversación
Escena	Text	Especifica la escena a la que pertenece la frase (esta escena debe estar contenida en un registro de la tabla Escenas)
Frase	Text	Contiene la frase en si
Narrador	Text	Es el personaje que dice la frase

ESCENAS		
Campo	**Tipo**	**Descripción**
ID	Integer	Identificador de cada registro de la tabla
Escena	Text	Nombre de la escena
Completada	Text	Indica si ha sido completada (Valores: Si / No)
Conversación	Text	Indica si la escena es una conversación (Valores: Si / No)
Escena Siguiente	Text	Nombre de la escena que se cargara una vez finalice esta
Carga Desde Diario	Text	Indica si la escena ha sido cargada desde la funcionalidad “Diario” del menú principal. Sirve para que una vez finalice la reproducción de la escena, se cargue el menú principal y no la escena siguiente. (Valores: Si / No)

ESTADÍSTICAS CLARA		
Campo	**Tipo**	**Descripción**
ID	Integer	Identificador de cada registro de la tabla
HP	Integer	Indica el valor de la vida del personaje del jugador
ATK	Integer	Indica el valor del ataque del personaje del jugador
Monedero	Integer	Contiene el número de monedas que posee el jugador
Coste Mejora	Integer	Establece el coste en monedas de cada mejora que realice el jugador
Mejora HP	Integer	Establece los puntos de vida que gana el jugador con cada mejora realizada en dicha estadística
Mejora ATK	Integer	Establece los puntos de ataque que gana el jugador con cada mejora realizada en dicha estadística

PREGUNTAS EGIPTO		
Campo	**Tipo**	**Descripción**
ID	Integer	Identificador de cada registro de la tabla
Pregunta	Text	Contiene la pregunta de este registro
Respuesta Correcta	Text	Contiene la respuesta correcta a la pregunta contenida en este registro
Respuesta Incorrecta 1	Text	Primera respuesta incorrecta a la pregunta
Respuesta Incorrecta 2	Text	Segunda respuesta incorrecta a la pregunta
Respuesta Incorrecta 3	Text	Tercera respuesta incorrecta a la pregunta

Las tablas Conversaciones y Preguntas Egipto se utilizan unicamente para realizar consultas, puesto que ninguno de sus registros variarán a lo largo del juego.

Las tabla Escena sufrirá las siguientes modificaciones en sus campos según avance el juego:

- Completada: Cuando una escena haya sido completada por el jugador (visualizada en el caso de una conversación y ganada en el caso de una fase de juego), su valor cambiará a “Si” para indicar dicho estado.
- Carga Desde Diario: Este valor unicamente cambiara cuando el jugador lance una escena desde el diario de viaje, siendo su valor “Si” hasta que se termine de visualizar la escena. Una vez terminada, su valor volverá a ser “No”. Solo puede haber un registro con el valor “Si” en este campo a la vez.

La tabla Estadísticas Clara tendrá las siguientes modificaciones:

- HP y ATK: Sus valores cambiaran cada vez que se les aplique una mejora.
- Monedero: Su valor se verá modificado cada vez que se complete una fase de juego, aumentando dicho valor con las monedas ganadas. A su vez, su valor disminuirá cada vez que el jugador mejore sus estadísticas.
- Coste Mejora: Este valor se incrementará con cada mejora de estadísticas que se realice.

4 – Diario de Desarrollo

En este apartado del documento procederé a exponer el proceso de desarrollo del proyecto en cada una de las tareas que se establecieron en la fase de planificación, explicando aquellas partes del código o de la interfaz que considere mas importantes o complejas, así como las dificultades y problemas que hayan ido surgiendo durante su desarrollo.

4.1 – Base de Datos

La estructura de la base de datos ya ha sido explicada en un punto anterior, por lo que aquí solo me centraré en como se conecta el juego a la misma y en como se realizan las consultas y las demás operaciones sobre esta.

Para poder conectarse a la base de datos desde un sistema Android, antes se debe copiar la base de datos a una ruta que permita su lectura y escritura. Una vez hecho esto, se deberá recoger dicha ruta para después poder generar la conexión:

```
string filepath;

if (Application.platform == RuntimePlatform.Android)
{
    //Para cargar la BD en Android
1   var loadingRequest = UnityWebRequest.Get(Path.Combine(Application.streamingAssetsPath, DatabaseName));
    loadingRequest.SendWebRequest();
    while (!loadingRequest.isDone)...
2   if (loadingRequest.isNetworkError || loadingRequest.isHttpError)
    {

    }
    else
    {
        if(!File.Exists(Path.Combine(Application.persistentDataPath, DatabaseName)))
3       {
            File.WriteAllBytes(Path.Combine(Application.persistentDataPath, DatabaseName), loadingRequest.downloadHandler.data);
        }
    }

4   filepath = Path.Combine(Application.persistentDataPath, DatabaseName);
    //Fin Parte de Android
}
```

Ilustración 3: Código para copiar y recoger la ruta de la BD

1. Se realiza una llamada a la ruta que contiene la base de datos. Esta ruta se encuentra en los archivos internos de la aplicación, por lo que no puede ser modificada. Esta ruta se obtiene mediante la variable streamingAssetsPath de la aplicación, a la cual se le añade el nombre de la base de datos para así poder seleccionarla.
2. Esta comprobación sirve para definir que hacer en caso de que la conexión con la ruta proporcionada anteriormente falle. Esta conexión siempre se realizara sin problemas

puesto que los archivos de la base de datos se encuentran almacenados de forma local en el dispositivo y es por ello que el contenido de este IF se encuentra vació.

3. Si la conexión se realiza sin problemas, se procederá a copiar la base de datos a una ruta de almacenamiento que si permite su modificación. Esta ruta se obtiene mediante la persistenceDataPath de la aplicación, la cual es una ruta que contiene todos aquellos archivos que se requiere que sean persistentes en el tiempo, es decir, que se mantengan guardados incluso cuando el juego se cierre. Si la base de datos ya existe, se omitirá este paso.
4. Una vez copiada (o comprobada su existencia), se procede a obtener la ruta de conexión a la base de datos mediante la combinación de la persistenceDataPath y el nombre de la misma.

* Para realizar la conexión desde el editor de Unity solo es necesario utilizar la ruta del equipo en la que esta almacenada la base de datos.

Una vez obtenida la ruta, se procederá a establecer la conexión con a base de datos mediante el siguiente código:

```
//Abrir conexion con la BD
conn = "URI=file:" + filepath;

Debug.Log("Stablishing connection to: " + conn);
SqliteConnection dbconn = new SqliteConnection(conn);
dbconn.Open();
//Fin Abrir Conexion con la BD
```

Ilustración 4: Código para realizar la conexión con la BD

- Se guarda la ruta de conexión en un string.
- Se abre la conexión mediante el uso de la clase SqliteConnection (también se puede utilizar la clase IDBConnection para este fin)

Con la conexión abierta ya se puede comenzar a operar sobre la base de datos. Estas operaciones se realizan mediante el uso de la clase SqliteCommand (o IDBcommand), la cual se encarga de carga la instrucción SQL Para ejecutar una instrucción se utilizara el método executoNonQuery para realizar modificaciones en los registros (INSERT, UPDATE,…) y la clase SQLiteDataReader (o IDataReader) para realizar consultas (SELECT).

Ejemplo de uso:

```
SqliteCommand dbcmd = dbconn.CreateCommand();
string query = "select * from PreguntasEgipto;";
dbcmd.CommandText = query;
SqliteDataReader reader = dbcmd.ExecuteReader();
while (reader.Read())
{
    nPreguntas += 1;
}
reader.Close();
```

Ilustración 5: Código que realiza una consulta a la BD

Dificultades encontradas: El código necesario para copiar la base de datos y obtener su ruta me fue bastante difícil de obtener puesto que todas las guiás que encontraba utilizaban un código diferente que esta obsoleto a día de hoy y que no funcionaba. El código utilizado finalmente, aunque en las versiones más modernas de Unity parece que ya esta obsoleto también, no da ningún problema por lo que decidí usarlo al ser el único funcional que encontré.

4.2 – Escena de Juego

Esta es la base de todo el proyecto y la que contiene el juego en si. En esta escena se muestra una pregunta en un panel y varias posibles respuestas y el jugador debe elegir una de ellas.

Si acierta, el personaje al que encarna realizara su movimiento de ataque, restando vida al enemigo de la fase. En caso de fallar, ocurrirá lo mismo pero a la inversa, siendo el enemigo el que atacará y el jugador el que recibirá el daño.

La fase terminará cuando la vida de alguno de los dos llegue a 0, momento en el cual aparecerá una animación de derrota para el que haya perdido. Tras esto, se mostrará un panel que contiene las opciones disponibles tras la partida:

- Si el jugador gana, este podrá continuar con la historia del juego, avanzando a la escena que venga después de la fase recién completada.
- Si el jugador pierde se le dará la opción de re intentar el nivel y la opción de volver al menú principal.

Para una mejor comprensión de la estructura de la escena, procedo a adjuntar una imagen de la misma:

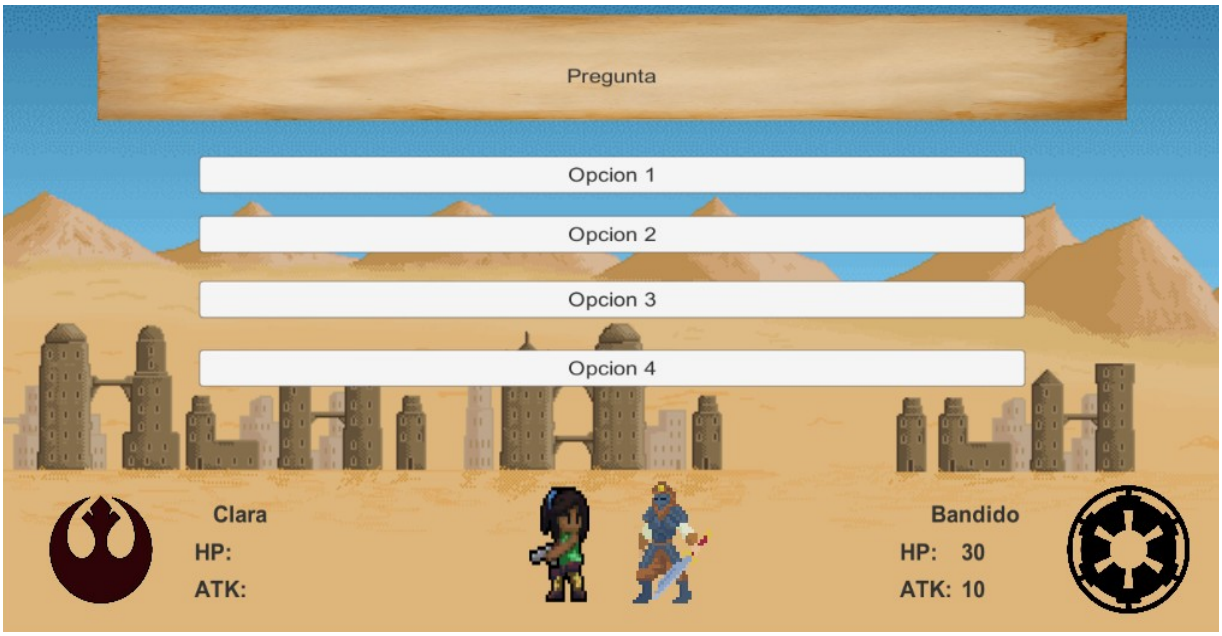

Ilustración 6: Visualización de la Escena de Juego

Como se puede observar, las estadísticas de la protagonista aparecen en blanco. Esto se debe a que sus estadísticas no estarán fijadas en la escena como las del enemigo (cada fase del juego estará contenido en una copia de la escena de juego a la que se le cambiara la estética en función de la apoce, hora del día, etc.), sino que serán cargadas desde la base de datos.

Aunque en una imagen estática no se puede apreciar, el fondo del escenario es móvil, al igual que los personajes.

El fondo está moviéndose hacia la izquierda de forma permanente e infinita para darle a la escena una sensación de movimiento. Esto se logra mediante el código siguiente:

```
void Update()
{
    float velocidadFinal = velocidad * Time.deltaTime;
    fondo.uvRect = new Rect(fondo.uvRect.x + velocidadFinal, 0f, 1f, 1f);
}
```

Ilustración 7: Método encargado de mover el fondo de la escena

En este código se modifica la posición del fondo en el eje X mediante una velocidad final que se obtiene a través de una velocidad ya determinada que es multiplicada por el tiempo que lleva la escena en funcionamiento. Este código se ejecuta una vez por cada frame (cuadro) del juego, es decir, cada vez que se actualiza la pantalla, el código es ejecutado, dando así la sensación de movimiento constante y fluido.

El movimiento de los personajes está controlado por animaciones, por lo que su funcionamiento se verá en su apartado correspondiente.

La carga de estadísticas se realiza mediante una consulta a la base de datos, con un código muy similar al visto en el apartado anterior.

Para la carga de las preguntas y de las respuestas, primero se genera aleatoriamente una serie de números, que están dentro de un rango predefinido, y que servirán para que la asignación de las respuestas a los botones sea totalmente aleatoria con cada carga. Esto se consigue de la siguiente forma:

```
int n1 = UnityEngine.Random.Range(2, 6);
int n2;
int n3;
int n4;

do
{
    n2 = UnityEngine.Random.Range(2, 6);
} while (n1 == n2);

do
{
    n3 = UnityEngine.Random.Range(2, 6);
} while (n1 == n3 || n2 == n3);

do
{
    n4 = UnityEngine.Random.Range(2, 6);
} while (n1 == n4 || n2 == n4 || n3 == n4);
```

Ilustración 8: Generación de la serie de números aleatorios

- Se genera un primer número dentro del rango establecido.
- Para el siguiente, se genera otro número dentro de ese mismo rango pero se impide que coincida con el número anterior.
- Se realiza el mismo proceso con los dos números restantes.

Con los números ya generados, se procede a realizar una consulta a la base de datos para recoger una pregunta.

Esta consulta también se realiza con un numero aleatorio, que indicara el ID de la pregunta a recoger.

El rango de generación de este ID viene dado por una consulta previa que recogió el número de preguntas existentes en la tabla consultada.

```
string query = "select * from PreguntasEgipto where id = " + idregunta + ";";
dbcmd.CommandText = query;
reader = dbcmd.ExecuteReader();
while (reader.Read())
{
    pregunta.text = pregunta.text + " Lector";
    pregunta.text = reader[1].ToString();
    resp1.GetComponentInChildren<Text>().text = reader[n1].ToString();
    if (n1 == 2)
    {
        b1Correcta = true;
        b2Correcta = false;
        b3Correcta = false;
        b4Correcta = false;
    }
    resp2.GetComponentInChildren<Text>().text = reader[n2].ToString();
    if (n2 == 2)
    {
        b1Correcta = false;
        b2Correcta = true;
        b3Correcta = false;
        b4Correcta = false;
    }
```

Ilustración 9: Asignación de respuestas

- Se realiza la consulta con el ID generado aleatoriamente.
- Cada botón tiene un numero aleatorio asignado.
- La posición 2 del data reader corresponde a la respuesta correcta, por lo que el botón que tenga asignado el numero 2 sera el que tenga la correcta y el resto cargara las respuesta incorrectas.

De esta forma las respuestas cargaran siempre de forma aleatoria y el juego siempre sabrá cual botón es el que contiene la respuesta correcta, puesto que el botón que la contiene también es asignado como el que tiene la repuesta correcta mediante el uso de variables booleanas.

Cuando el jugador contesta, un método comprueba si el botón pulsado es el que contiene la respuesta correcta. De ser así, se pinta de color verde y, en caso contrario, de color rojo. Al terminar este método se llama a una corrutina que se encarga de lanzar las animaciones necesarias en función de la situación, ademas de restar la vida al personaje golpeado.

* Las corrutinas son trozos de código que se ejecutan en Unity en un hilo alterno al hilo de ejecución principal. Las utilizo para lanzar las animaciones ya que me permiten parar la ejecución del código que contienen durante un tiempo establecido mientras la animación se muestra, sin que esto afecte al movimiento de otros elementos de la escena.

4.3 – Animaciones

Para su realización he utilizado el motor de animaciones que provee Unity, el cual consiste en el uso de sprites (Imágenes que contienen el diseño del personaje y los diferentes movimientos que realiza). Cada movimiento se compone de varias imágenes con pequeñas variaciones entre si. Esto funciona de la siguiente forma:

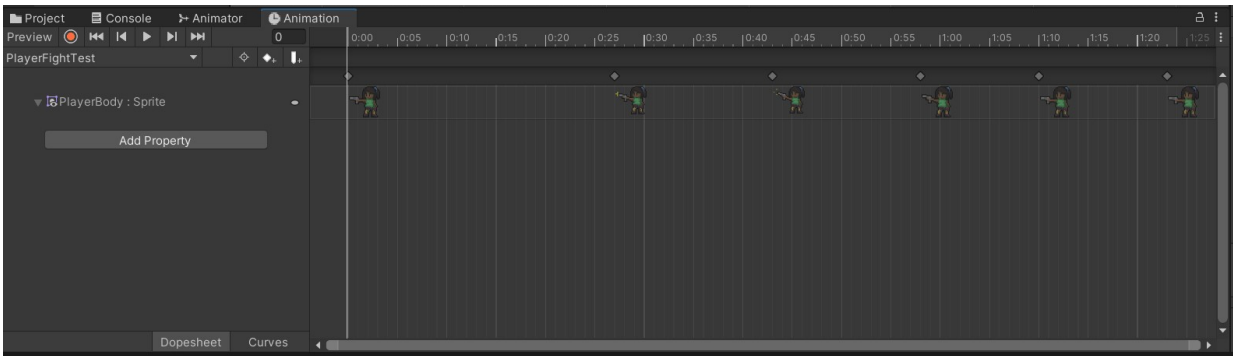

Ilustración 10: Ejemplo de animación 2D con Sprites

1. Se le añade un Sprite Renderer al objeto que se quiera animar.
2. Se crea el archivo que contendrá la animación.
3. Se elige el componente del objeto que se quiere animar, en este caso, el sprite del personaje.
4. En función del tiempo que se quiera dejar pasar entre cada cambio de sprites, se colocara cada uno de ellos en la regla de tiempo. El orden de colocación de los sprites ira en función de el efecto de movimiento que se quiera conseguir.
5. Se puede ver el resultado de la animación pulsando en botón play del Animator.

* Hay que tener en cuenta que la primera animación que se crea es la que Unity pondrá por defecto en el objeto, por lo que está se ejecutará todo el tiempo sin pausa. Lo mejor es crear primero la animación que el objeto utilizará cuando esta en reposo para así evitar este inconvenientes.

Para utilizar las animaciones en el momento en que se cumple una condición, deberemos llamar al Animator de su objeto y decirle que ejecute la animación que le pasamos por parámetro a su método play. Esto se consigue de la siguiente forma:

```
IEnumerator PantallaNegroDesvelar()
{
    transicion.Play("DesvelarPantallaEgiptoMovimiento");
    yield return new WaitForSeconds(1.75f);
    pantNegra.enabled = false;
}
```

Ilustración 11: Llamada a una animación desde una corrutina

- La ejecución de las animaciones se realiza en una corrutina por las razones que se explicaron en el apartado anterior.

En el código anterior, la parte de “yield return new WaitForSeconds(1.75f)” indica el tiempo que ha de esperar la aplicación antes de ejecutar el resto del código. Este tiempo es el que necesita la animación para completarse.

Cuando se realiza una animación con un personaje (atacar, por ejemplo), es necesario crear una transición en el Animator para que, cuando se termine de ejecutar dicha animación, el Animator pase de nuevo a ejecutar la animación que esta establecida por defecto. Si no se hace esto, la animación ejecutada se repetirá en bucle sin parar.

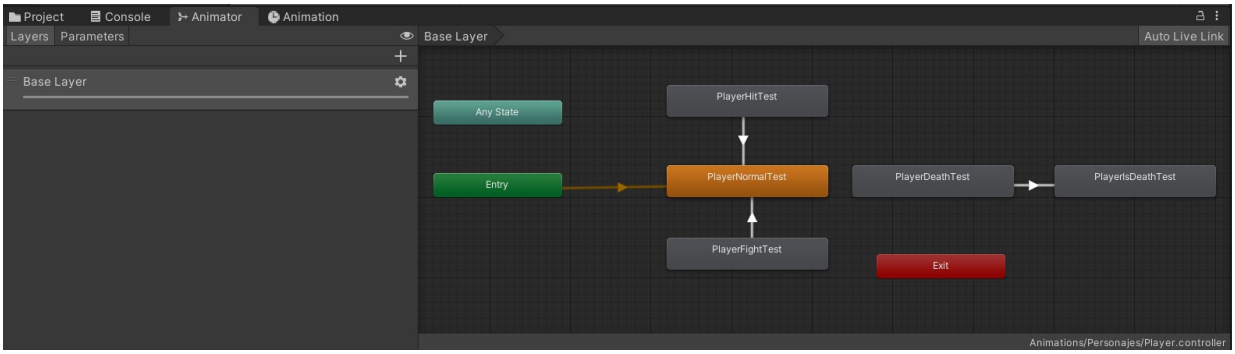

Ilustración 12: Estructura y transiciones de las animaciones de un objeto

4.4 – Escena Pantalla de Inicio

Esta escena es la más simple de todo el juego, puesto que unicamente consta de un fondo, el titulo del juego y un botón que servirá para comenzar la partida.

Su única función es la de comprobar si ya existe una partida guardada previamente, lo cual se logra mediante la realización de una consulta a la base de datos. En esta consulta se revisa si la primera cinemática del juego ha sido ya completada, dando como resultado dos posibles situaciones:

- No ha sido completada aun: El juego cargará esta escena nada mas pulsar sobre el botón de esta pantalla.
- Ya ha sido completada: Se carga la escena del menú principal.

Para cargar las diferentes escenas se utiliza una clase especifica de la API de Unity, el SceneManager:

```
public void CambiarEscena()
{
    if(partidaEmpezada == true)
    {
        SceneManager.LoadScene("MenuPrincipal");
    }
    else
    {
        SceneManager.LoadScene("CinematicaEgipto");
    }
}
```

Ilustración 13: Carga de escenas

4.5 – Escena Cinemática

Mediante está escena se reproducirán todas las conversaciones entre personajes que tendrán lugar a lo largo del juego, contando la historia del mismo. La escena consta de un fondo estático, un panel en el que se escribirán las frases de la conversación, un cuadro de texto que indicara el nombre del personaje que se encuentre hablando en ese momento y un botón invisible que ocupa toda la pantalla y cuya función es la de cargar la siguiente frase de la conversación cuando este es presionado.

Está escena puede ser cargada por dos motivos:

- Se ha completado una fase del juego y se ha procedido a cargar la siguiente cinemática para continuar con la trama.
- Se ha cargado una cinemática ya desbloqueada desde el diario del menú principal.

Para saber cual en cual de las dos situaciones nos encontramos, se utiliza este código:

```
dbcmd = dbconn.CreateCommand();
string query = "SELECT * FROM Escenas WHERE CargaDesdeDiario = 'Si';";
dbcmd.CommandText = query;
reader = dbcmd.ExecuteReader();

if(reader.Read())
{
    escenaActual = reader[1].ToString();
    escenaSiguiente = "MenuPrincipal";
    reader.Close();

    query = "SELECT * FROM Conversaciones WHERE Escena = '" + escenaActual + "';";
    dbcmd.CommandText = query;
    reader = dbcmd.ExecuteReader();

    while (reader.Read())
    {
        frases.Add(reader[2].ToString());
        narrador.Add(reader[3].ToString());
    }
    reader.Close();

    boton.enabled = false;

    query = "UPDATE Escenas SET CargaDesdeDiario = 'No' WHERE Escena = '" + escenaActual + "';";
    dbcmd.CommandText = query;
    reader = dbcmd.ExecuteReader();
    reader.Close();
}
```

Ilustración 14: Carga de una cinemática desde la función Diario del Menú Principal

En esta primera parte del código se realiza una consulta a la tabla Escenas, cuya condición para devolver un registro debe ser que el campo Carga Desde Diario tenga como valor “Si” ya que, si la cinemática ha sido lanzada desde el diario, la escena elegida tendrá ese valor en su campo.

Si se encuentra un registro con dicho valor, se procede a recoger en una lista todas las frases de la conversación mediante una consulta a la tabla Conversaciones. Después se actualiza el registro de la tabla Escenas para que el valor vuelva a ser “No”.

Si, por contra, la escena ha sido cargada por que se debe avanzar en la historia, el código ejecutado será el siguiente:

```
else
{
    reader.Close();

    query = "SELECT * FROM Escenas WHERE Completada = 'No' AND Conversacion = 'Si';";
    dbcmd.CommandText = query;
    reader = dbcmd.ExecuteReader();

    reader.Read();
    escenaActual = reader[1].ToString();
    escenaSiguiente = reader[4].ToString();
    reader.Close();

    query = "SELECT * FROM Conversaciones WHERE Escena = '" + escenaActual + "';";
    dbcmd.CommandText = query;
    reader = dbcmd.ExecuteReader();

    while (reader.Read())
    {
        frases.Add(reader[2].ToString());
        narrador.Add(reader[3].ToString());
    }
    reader.Close();

    boton.enabled = false;

    query = "UPDATE Escenas SET Completada = 'Si' WHERE Escena = '" + escenaActual + "';";
    dbcmd.CommandText = query;
    reader = dbcmd.ExecuteReader();
    reader.Close();
}
```

Ilustración 15: Carga de una cinemática siguiendo el flujo de la historia del juego

Como se puede observar, se realiza una consulta a la tabla Escenas mediante la cual se recogerá la siguiente escena que aun no haya sido completada por el usuario y que sea una conversación. Una vez recogida, se procede a guardar la conversación en una lista y a actualizar el registro de la escena para indicar que el jugador ya ha visto la escena y que, por tanto, ya tiene permiso para aparecer en el diario del menú principal.

4.6 – Escena Menú Principal

Esta es la escena desde donde el jugador podrá acceder al resto de escenas y funcionalidades del juego. Esta formada por una imagen de fondo que simula una interfaz de usuario futurista, en la cual se superpondrán 3 ventanas diferentes, en función de la opción del menú escogida. La pantalla que aparece por defecto es la del mapa de situación y en la parte superior de la interfaz se cargará el nombre de la fase en la que se encuentra el jugador actualmente.

Las opciones a elegir son:

- Viajar: Carga una escena de transición que simula el viaje de los personajes a la fase de juego. Después, abre dicha fase y comienza la partida.

Ilustración 16: Opción "Viajar" del Menú Principal

- Mejorar: Muestra la ventana de mejora de estadísticas, mediante la cual se pueden mejorar la vida y el daño del personaje del jugador.

Ilustración 17: Opción "Mejorar" del Menú Principal

- Cuaderno de Bitácora (Diario): Muestra la ventana del diario y permite visualizar cinemáticas ya vistas por el jugador.

Ilustración 18: Opción "Diario" del Menú Principal

Para realizar el cambio entre ventanas se utiliza una animación de carga que oculta la parte central de la escena mientras las pantallas cambian de posición. Esto se realiza mediante el uso de corrutinas (explicadas en los apartados 4.2 y 4.3) en las cuales se cambia la posición, orientación y estado de los elementos implicados.

```
IEnumerator HUDMovimiento(string pantalla)
{
    yield return new WaitForSeconds(1f);

    if (pantalla == "Diario")
    {
        visibilidadElementos(true);

        loadingAnimator.Play("CargaHUD");
        cargaHUD.enabled = false;
        tBarraDiario.text = "Mapa";
    }
    else if (pantalla == "Mapa")
    {
        visibilidadElementos(false);

        loadingAnimator.Play("CargaHUD");
        cargaHUD.enabled = false;
        tBarraDiario.text = "Cuaderno de Bitacora";

        tBarraDiario.enabled = true;
        barraDiario.enabled = true;
        bDiario.gameObject.SetActive(true);
        viajar.gameObject.SetActive(true);
    }

    bDiario.enabled = true;
    bDiario.transform.Rotate(0f, 0f, 180f, Space.Self);
    barraDiario.transform.Rotate(0f, 180f, 0f, Space.Self);
}
```

Ilustración 19: Código que cambia entre las pantallas "Mapa" y "Diario"

- En función del botón pulsado en el menú, se procede a cambiar entre una pantalla u otra.
- La pantalla “Mapa” esta siempre en su posición y son las otras dos las que cambian su localización y visibilidad a la hora de mostrarse, posicionándose encima del mapa.
- Los botones de la interfaz se mantienen desactivados mientras dura el cambio de pantallas.
- El botón de la opción que no ha sido elegida se oculta y el botón pulsado se rota para indicar que el jugador se encuentra en otra pantalla.

La mejora de estadísticas se realiza mediante la ejecución de una instrucción UPDATE, siempre y cuando el jugador disponga de las monedas suficientes para ello. Si el coste de la mejora supera al dinero del que dispone el jugador, el texto del coste se volverá de color rojo para indicar tal circunstancia.

La reproducción de cinemáticas se realiza mediante un UPDATE sobre la tabla Escenas para después cargar la escena encargada de mostrar las conversaciones. Cuando se finalice la reproducción, el jugador será llevado de vuelta al menú principal.

4.7 – Transiciones

Para evitar que el cambio entre escenas sea muy brusco, he decidido implementar una transición de pantalla negra que funciona de la siguiente manera:

1. Al cargar una escena (menú principal, por ejemplo), esta comienza estando tapada completamente por una pantalla negra que, al momento de iniciarse la escena, procede a desplazarse hacia la izquierda hasta quedar fuera de cámara, generando así un efecto similar al del telón de un teatro.
2. Cuando termina de moverse, la pantalla es posicionada fuera de cámara a la derecha de lo que el jugador ve.
3. Al cambiar de escena, la pantalla vuelve a moverse hacia la izquierda hasta tapar por completo la visión del jugador. Es en este momento cuando se produce el cambio de escena, la cual también tiene su propia pantalla negra que procede a moverse a la izquierda para mostrar su contenido.

Para que el movimiento de la pantalla negra se vea fluido, este se realiza mediante una animación en la que lo que se modifica es su posición a lo largo del tiempo.

Cabe mencionar que existe una escena que funciona como una transición especial desde el menú principal a la escena de juego. Esta escena simula el viaje de la nave de los protagonistas al lugar en el que se desarrolla la acción del juego, mostrando la nave volando por el cielo durante unos segundos.

Las técnicas utilizadas en esta escena son las mismas que se explicaron en el apartado 4.3 en lo que a animaciones se refiere, mientras que para el movimiento del fondo se utiliza el mismo código que en el apartado 4.2.

4.8 – Audio

La ejecución de sonidos en el juego, ya sea como música de fondo o efectos de sonido, se realiza mediante la adición de un elemento AudioSource, el cuál se encarga de ejecutar los clips de audio que se le pasen.

Los clips de audio, al igual que las imágenes y demás elementos visuales del juego, deberán estar almacenadas dentro del proyecto.

La ejecución de los sonidos de fondo se puede realizar sin necesidad de utilizar un código especifico, pues basta con añadir el clip al AudioSource y seleccionar las opciones necesarias. En este caso, se seleccionan “Play On Awake” para que el audio se ejecute nada más se cargue la escena, y “Loop”,que sirve para que el clip sea repetido una vez este finalice su reproducción.

Para los efectos de sonido (pulsar botones, realizar un golpe, etc), se deberá colocar el siguiente código en el momento que se quiera que se ejecute el clip de audio:

```
reproductor.PlayOneShot(clip);
```

- Para poder utilizarlo se deberán enlazar un AudioSource y un AudioClip al script correspondiente.

5 – Conclusiones

Para la realización de este proyecto me propuse crear una versión demo del juego, que contuviera las funcionalidades básicas del mismo como son la escena de juego, la mejora de estadísticas y el contar la historia mediante conversaciones entre personajes.

En ese aspecto, creo que he logrado mi objetivo puesto que todas esas funciones están presentes en la demo y su correcto funcionamiento ha sido probado y confirmado.

En lo referente al diseño visual, debido a que el tiempo disponible para su desarrollo ha sido muy corto, he tenido que utilizar Sprites, fondos y diseños gratuitos que estaban disponibles en Internet, aunque algunos diseños de la interfaz si son de mi autoría.

La mayor dificultad que he encontrado para llevar a cabo este proyecto ha sido el tener que aprender no solo un entorno de trabajo completamente ajeno a mis conocimientos como es Unity, sino que también he tenido que aprender a utilizar un lenguaje de programación nuevo.

C#, a pesar de ser similar a java, tiene ciertas peculiaridades que, sumado a la necesidad de tener que aprender a utilizar la API de Unity, han hecho que el avance del proyecto fuera más lento y complejo.

Otra de las dificultades encontradas fue la poca información actualizada que existe sobre la implementación de SQLite en un proyecto Unity, ya que en su mayoría se tratan de guías obsoletas que no funcionan para las versiones actuales del sistema Android.

Un punto que considero muy positivo es que las escenas que controlan el juego en si (escena de juego) y las conversaciones (escena cinemática) pueden ser clonadas de una forma fácil y rápida, posibilitando así la creación de diferentes escenarios de juego sin tener que emplear tiempo y recursos en crearlas desde cero, puesto que solo se cambiaría su aspecto, no su funcionalidad a nivel programático.

A pesar de las dificultades y problemas encontrados, considero que el resultado final ha sido muy similar al que tenia en mente y que conforma una base solida desde la cual seguir desarrollando el proyecto en un futuro, mejorando su diseño visual y las posibilidades que el juego ofrezca al jugador.

Para terminar, me gustaría señalar que este proyecto esta pensado como un producto de entretenimiento pero que a la vez sirva como material didáctico. Esto se debe a que la mecánica principal de este juego se basa en contestar pregunta sobre historia universal, el cual no es un tema muy popular entre niños y adolescentes.

El objetivo de este juego es acercar dicho tema a ese público, haciendo que aprendan historia sin que ellos se den cuenta, la cual es la mejor forma de aprender.

6 – Mejoras Futuras

Debido a que este proyecto se basaba en la realización de una base jugable y de sus funciones básicas, es decir, una demo, aun existen multitud de añadidos y mejoras que se le pueden agregar para completar así el proyecto y que este pueda ser lanzado al mercado, los cuales son:

- Adición de retratos de los personajes durante las cinemáticas, mostrando sus estados de animo y sus expresiones mientras se desarrollan las conversaciones.
- Adición de dichos retratos en la escena de juego, al lado de sus estadísticas, tanto para el personaje del jugador como para los enemigos.
- Mejora del diseño visual en general, con escenarios más variados e interfaces mas elaboradas.
- Adición de uno o mas personajes jugables, cada uno con sus propias estadísticas y habilidades especiales, las cuales ayudaran al jugador de diferentes formas durante la partida, ya sea eliminando una de las respuestas incorrectas o saltando una pregunta.
- Adición de narradores orales que interpreten a cada uno de los personajes del juego durante las conversaciones y aumentar así la accesibilidad del juego.
- Creación de varias ranuras de guardado para posibilitar que varios jugadores puedan tener una partida propia en un mismo dispositivo móvil.
- Creación de un modo de juego alternativo cuya meta sea la de contestar el máximo número de preguntas posibles tratando de superar la marca de otros jugadores o la suya propia, tanto a nivel local como global mediante la subida de las puntuaciones a un servidor online.
- Creación de un modo multiplicador que enfrente a los personajes de dos jugadores en partidas online cuya meta sea la de reducir la vida del oponente a 0.
- Adición de un sistema de logros para incentivar al jugador a realizar determinados hitos.

Otra vía de acción para continuar expandiendo el proyecto, una vez se hubieran completado todas las mejoras anteriormente descritas, sería la creación de una versión del juego para equipos informáticos de sistema operativo Windows, ya que de esta forma se podría expandir el alcance del videojuego creado, ademas de que se podrían explorar nuevas opciones y añadidos que no fuera posible añadir en la versión Android.

También se podría añadir un sistema de monetización en la versión Android mediante la inclusión de anuncios o la venta de skins dentro del juego, de forma que no se afectara a la jugabilidad.

7 – Bibliografía

Todos aquellos recursos, documentos y guías consultadas y utilizadas durante la realización de este proyecto son enumeradas a continuación.

Manual de Unity (Incluidos todos sus apartados):

- Autor: Unity Technologies
- Fecha de realización: 2016
- URL: https://docs.unity3d.com/es/530/Manual/UnityManual.html

Android, Conexión SQLite en Unity

- Autor: Alex Estudillo
- Fecha de realización: 19/03/2020
- URL: https://abigsite.blogspot.com/2020/03/android-conexion-sqlite-en-unity.html

Metodologías Ágiles, SCRUM:

- Autor: ProyectosAgiles.org
- Fecha de realización: No consta
- URL: https://proyectosagiles.org/que-es-scrum/

Tablero SCRUM online:

- Creador de la utilidad: meister task
- URL: https://www.meistertask.com/

Curso Udemy de introducción al desarrollo de videojuegos:

- Autor: Héctor Costa Guzmán
- Fecha de realización: No consta
- URL: https://www.udemy.com/course/unity-5-primer-videojuego-2d-multiplataforma/learn/lecture/6750822#overview

Recursos para la realización de las animaciones de los enemigos:

- Autor: Sven Thole
- Fecha de realización: 17/08/2020
- URL: https://assetstore.unity.com/packages/2d/characters/bandits-pixel-art-104130

Recursos para la realización de las animaciones del personaje principal:

- Autor:GoPro UK
- Fecha de realización: No consta
- URL: https://www.vhv.rs/viewpic/wmmxbb_preview-pixel-art-character-sprite-sheet-hd-png/