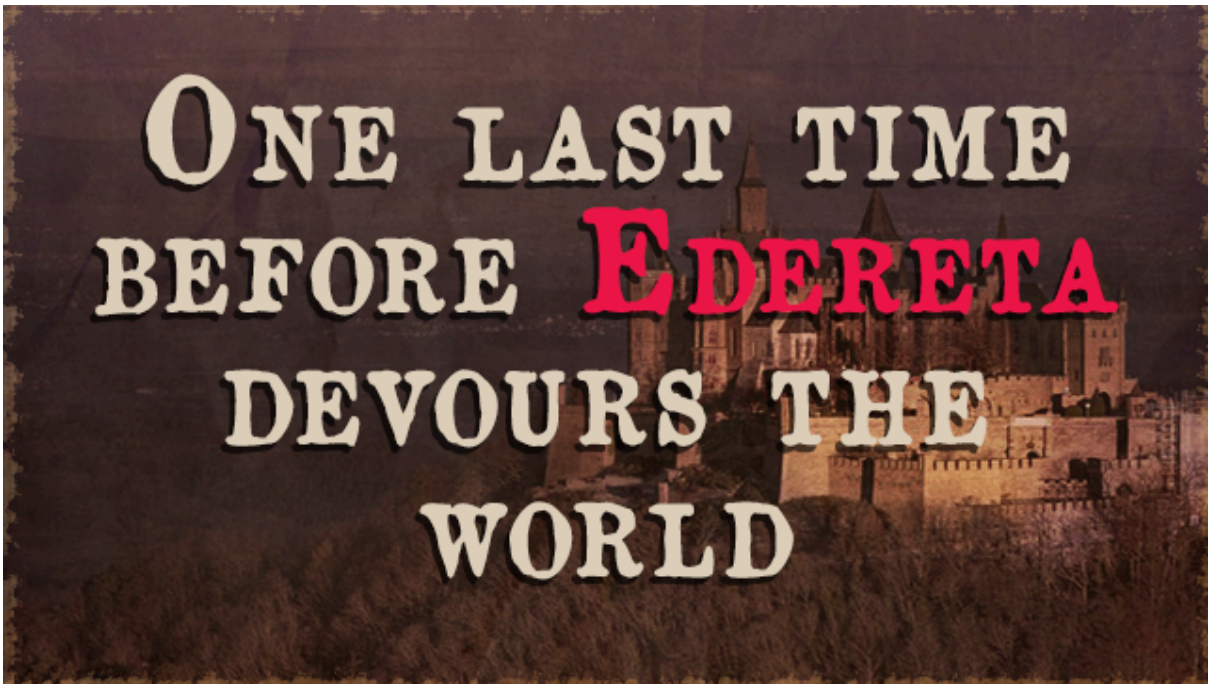
Proyecto fin de grado
Fernando Tarriño del Pozo
DAM-2023-2024



Juego Roguelike 2D en Unity

-

Análisis iniciales

ÍNDICE

1 - Concepto del proyecto

1.1 - Objetivos del proyecto

Se pretende crear un juego completamente funcional usando el motor Unity con perspectiva 2D de tipo roguelike con elementos de supervivencia y suerte, permadeath y sin persistencia entre diferentes partidas. Se busca además crear una sensación híbrida entre juego de ordenador y juego de mesa mediante la gestión de acciones (puntos de movimiento, por ejemplo) y eventos que suceden al final de tu turno de juego.

El juego debe tener un sistema completo formado por diferentes menús y submenús (menú inicial, menu de configuracion, menu de opciones ingame etc.) Dicho sistema debe permitir un flujo de uso sin necesidad de cerrar la aplicación en ningún momento o usar terceras aplicaciones para cualquier tarea.

Podemos dividir en juego en dos partes mejor explicadas más adelante en los puntos 3 (Base de datos) y 4 (Mapa de juego) que se pueden resumir en:

- **Flujo de creación de una nueva partida**
- **Juego con partida ya creada**

La diferenciación sobre estos datos viene en cómo se crea la nueva partida. Para generar el mundo cada vez que se inicia una partida nueva se hará uso de diferentes algoritmos aleatorios con mezcla de elementos fijos. Dichos algoritmos a su vez acceden a una base de datos online en la cual se tiene almacenada toda la información que queremos usar para la generación de los elementos de juegos tales como objetos, encuentros, lugares etc.

Una vez que los datos empleados por el programa para la partida han sido generados se almacenarán dentro de clases de C# y los usaremos para construir el mundo. A partir de este momento Unity gestionará todo.

Si se inicia una partida nueva se volverá a generar otro mundo semi-aleatorizado accediendo a la base de datos.

Existe además un programa externo al juego a modo de herramienta de desarrollador que se podrá emplear para consultar la base de datos, modificarla y añadir nuevas entradas (operaciones CRUD). De esta forma no será necesario abrir Unity o editar código si queremos editar la base de datos. El usuario podrá disponer de una interfaz diseñada con java swing mediante la cual modificar todo lo que desee.

1.2 - Ejemplos comerciales

En esta sección se mostrarán algunos ejemplos comerciales ya existentes similares a lo que se pretende recrear. Es importante destacar que estos juegos solo se usan para intentar ubicar mejor la idea, en general el proyecto cambia muchas cosas y añade otras que no se ven en estos juegos.

1.2.1 - Caves of Qud

Caves of Qud

Caves of Qud es un juego de tipo roguelike desarrollado por Freehold Games. Se compone de un mundo abierto generado **semi-proceduralmente** donde nuestro personaje se mueve en una vista plana en 2D y dispone de sistema de interacción e inventario con sus alrededores.

El objetivo de Caves of Qud se basa en sobrevivir todo el tiempo posible mientras se explora el inhóspito mundo lleno de peligros. Tiene una dificultad alta y es uno de los pilares sobre los que se basa. Además dispone de un sistema complejo de política con diferentes reglas, clanes, sucesos históricos y otros componentes que también son generados proceduralmente.

En el momento de redactar este documento Caves of Qud aún se encuentra en desarrollo, se tiene previsto su lanzamiento definitivo a lo largo del 2024. Fecha aún por determinar.

1.2.2 - Dungeons of Dredmor

Dungeons of Dredmor

Dungeons of Dredmor es un juego roguelike que publicado en 2011 por Gaslamp Games en el cual el jugador toma el rol de un explorador que va descendiendo niveles de creciente dificultad dentro de una mazmorra generada aleatoriamente usando diferentes biomas y enemigos.

El objetivo del juego es llegar hasta el final de la mazmorra y derrotar a un jefe final de gran poder. Mientras se va explorando la mazmorra el jugador se vuelve más poderoso gracias a los objetos que encuentra y los monstruos que logra derrotar.

Uno de los aspectos que más podemos remarcar de este ejemplo es **el movimiento** del jugador en un **tablero separado por casillas**.

Nuestro proyecto usará un movimiento muy similar a este pero la cámara no estará tan cerca del personaje.

1.2.3 - Pixel Dungeon

Pixel Dungeon

Pixel Dungeon es un juego roguelike lanzado tanto para PC como para Android y está hecho con Java. Fue lanzado en el 2012 por Watabou y Retronic Games, dos años más tarde en 2014 lo hicieron **de código abierto**.

Puede consultarse toda la información del juego en el siguiente enlace:

https://watabou.itch.io/pixel-dungeon

Este juego se centra principalmente en la supervivencia del personaje, dispone de una interfaz más limpia que los anteriores ejemplos para disponer de un mejor control (orientado a móviles) y claridad a la hora de presentar los datos. Esto es algo de vital importancia para nuestro proyecto y uno de los puntos que queremos lograr con más prioridad.

1.3 - Explicación de términos

1.3.1 - Roguelike

Definición de **Wikipedia**:

> *Los videojuegos de mazmorras (también llamados roguelike, o videojuegos de calabozos en América latina) son un subgénero de los videojuegos de rol que se caracterizan por una aventura a través de laberintos, a través de niveles generados por procedimientos al azar, videojuegos basados en turnos, gráficos basados en fichas y la muerte permanente del personaje del jugador. La mayoría de los videojuegos de mazmorras se basan en una narrativa de fantasía alta, que refleja su influencia de los juegos de rol de mesa como Dungeons & Dragons.*

Definición de **Devuego**:

> *Subgénero de los RPG que se caracteriza porque los niveles son generados aleatoriamente, la muerte del personaje es irreversible (permadeath) y el movimiento se realiza por turnos. Inicialmente se caracterizaban también por tener gráficos muy simples (en ASCII) y controlarse mediante comandos de texto.*
>
> *El nombre proviene del videojuego Rogue (Roguelike, como Rogue), desarrollado en 1980, que fue pionero de este estilo.*

Dentro de este marco se moverá nuestro juego, si bien también cabe remarcar que en lo que a clasificación de videojuegos, juegos de mesa o ficción en general es complicado definir los límites del producto para asignar correctamente un género u otro. Incluso dentro de un género existen muchos subgéneros o variaciones dadas que se diferencian unos de otros en cómo se aplica una mecánica exactamente o el tipo de cámara usados.

Por ejemplo, nuestro juego es **roguelike** por que cuenta con **muerte permanente** pero vamos a tener un pequeño sistema de guardado que permite de forma puntual guardar la partida para continuar luego. En cambio si el personaje muere, dicha partida deja de ser accesible. Esto lo mueve en un terreno complejo en cuanto a la definición de si el juego es realmente roguelike o **roguelite**. Los roguelite son un género totalmente diferente de los roguelike (aún que su nombre se parezca) donde existe persistencia entre diferentes partidas guardadas o un sistema complejo de guardado de partidas.

Existen muchos otros términos interesantes referentes a juegos de nuestro estilo. Por ejemplo, si nuestra cámara fuese en primera persona, el juego sería un **dungeon crawler**.

1.3.2 - Muerte permanente y guardado de partidas

1.3.2.1 - Savegames

Pese a que nuestro juego contará con un sistema de muerte permanente, también tenemos un sistema de guardado ligero de forma situacional. Mientras nuestro personaje explora el mundo podrá encontrar unos **edificios concretos donde guardar la partida.**

Estos edificios son de un solo uso y una vez guardada la partida en ellos no se podrán volver a usar para dicho fin. Esto nos permite guardar el progreso y cerrar el juego para volver más adelante.

Si en algún momento el personaje **muere** durante la partida el punto de guardado **se perderá** quedando inaccesible. Se podrá consultar la información del guardado con datos de la partida tales como los turnos que ha jugado, sitios explorados etc. Pero no se podrá acceder a la partida para seguir jugando.

La intención detrás de esta decisión radica en permitir jugar partidas extensas tanto en móvil como en PC pero sin necesidad de hacerlas todas en una sola sesión pero a la vez se respeta la mecánica de muerte permanente.

1.3.2.2 - Permadeath

Como ya se ha mencionado en el punto anterior, nuestro juego contará con un sistema de muerte permanente que bloquea el acceso al archivo de guardado una vez que el jugador ha sido eliminado por cualquier motivo.

Existirán algunos objetos valiosos dentro del juego que te permiten, por ejemplo, evitar la muerte en una ocasión. Dando una oportunidad extra de seguir el recorrido. Todo esto se verá más adelante con el diseño de las mecánicas definitivas del proyecto.

En general y combinando el sistema de puntos de guardado y muerte permanente podemos concluir que el juego sigue dentro de la categoría de **roguelike**.

1.4 - Multiplataforma (Windows y Android)

Uno de los principales objetivos a la hora de desarrollar el juego es que pueda ser ejecutado tanto en Windows como Android.

Este es el motivo principal para elegir **Unity** como motor de desarrollo. Unity permite compilar el proyecto fácilmente para ambas plataformas así como también lo permite, por ejemplo, Android Studio. Unity nos aporta un entorno de desarrollo complejo y versátil donde no tenemos que preocuparnos de sobre que sistema va a lanzar el juego final.

En una primera instancia se ideó Java directamente para este proyecto y usar una librería gráfica como libGDX para el renderizado. Finalmente se ha decidido Unity para poder afrontar el reto de tener el juego funcionando simultáneamente con el mismo código tanto en Windows como Android.

1.4.1 - Posibles problemas con la base de datos

Debido a que nuestro juego y todos los códigos subyacentes deben poder ejecutarse desde Android nos vemos limitados a la hora de definir qué base de datos vamos a usar. Por ejemplo, una base de datos PostgreSQL o MySQL como pueden ser las que nos encontramos gestionadas por XAMPP o SQLite no nos valdrían dado que son bases de datos centradas en sistemas Windows/UNIX no orientadas a las distribuciones que usa Android.

Por ello se ha optado por usar una base de datos online y conectar directamente a ella para recuperar los datos. En los primeros compases del proyecto uno de los objetivos a lograr es definir qué base de datos se usará de forma definitiva y algunas pruebas con dicha base de datos tanto desde Windows como desde Android.

En caso de que la base de datos de excesivos problemas (por imposibilidad de usar bases de datos online, gratuitas y óptimas) se **priorizará el lanzamiento del juego en Windows** dejando el despliegue de Android en un segundo plano.

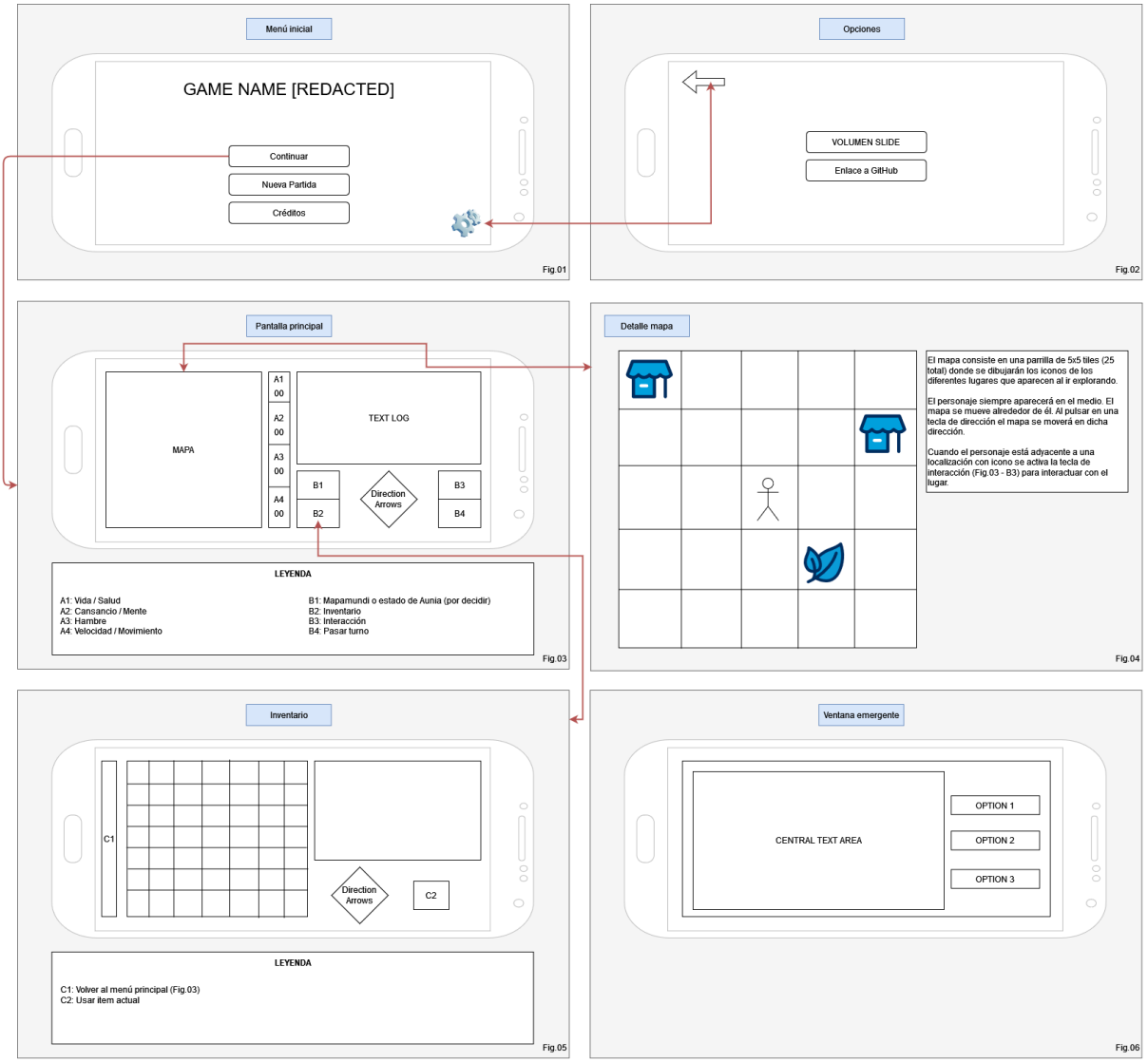
Esto no quiere decir que no se vaya a realizar el despliegue en Android si no que, posiblemente y en dicho caso, **se usarán dos arquitecturas diferentes para Windows/Android.** Por ejemplo en Windows usaremos una base de datos propiamente dichas (SQL gestionado por XAMPP en local) que es fácilmente replicable de forma online y un sistema de archivos en Android.

Demostrando con esto la versatilidad de la aplicación así como el funcionamiento de la base de datos en, al menos, una de las dos plataformas.

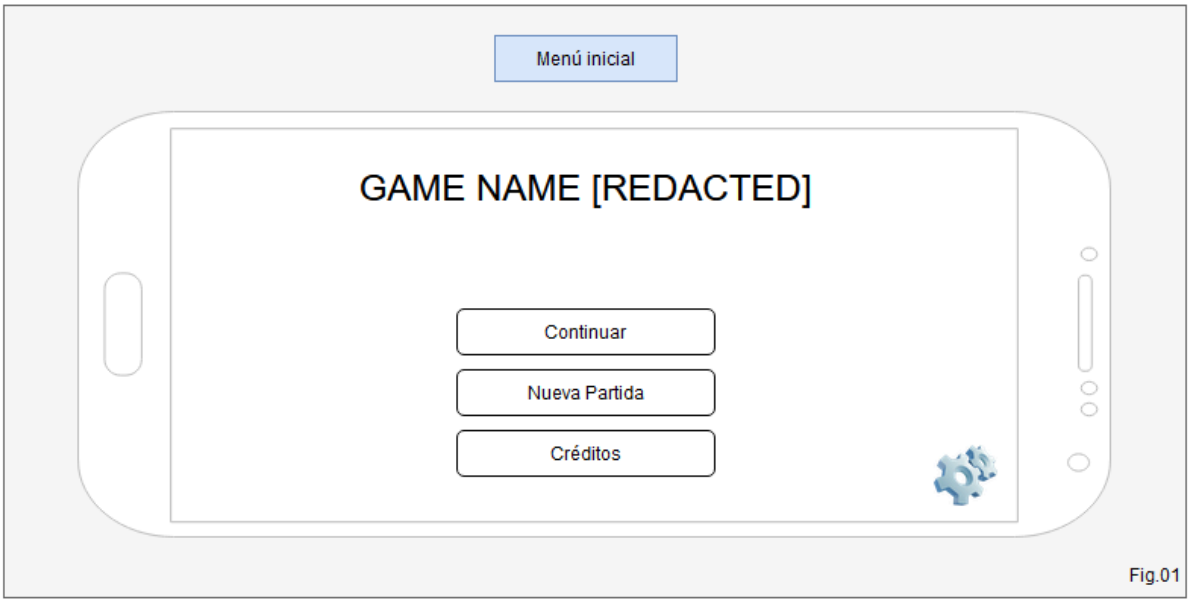
2 - Diseño de interfaz

2.1 - Revisión del diseño (prototipo)

Usando la herramienta **app.diagrams.net** de Google se ha diseñado un primer prototipo informativo sobre el aspecto que tendrán los diferentes menús del juego. Es importante reseñar que estos modelos son informativos y se verán ligeramente modificados en función de los requisitos funcionales/no funcionales que se planteen más adelante. Debido al corto periodo de desarrollo del que se va a disponer es muy posible que la versión final del proyecto difieran en algunas cosas de lo mostrado en el prototipo como, por ejemplo, en la cantidad o funcionalidad de los botones del menú opciones.



2.2 - Menú inicial



Esta será la pantalla que veremos al abrir la aplicación tanto desde Android como desde Windows.

El menú inicial presentará un diseño sencillo con una imagen de fondo que se irá moviendo lentamente, el nombre del juego en grande y varios botones. La distribución y cantidad de botones aún queda por definir en función de las mecánicas finales que se decidan implementar. El esquema que vemos en la foto superior nos enseña ligeramente como se colocarán dichos botones.

Los botones posibles son los siguientes (pendiente de revisión):

- Nuevo juego
- Cargar partida
- Opciones
- Salir del juego
- Créditos

La distribución de los botones y submenús puede variar debido a diferentes factores. Por ejemplo, inicialmente al tratarse de un roguelike no se iba a disponer del botón “Cargar partida” pero finalmente durante el periodo de análisis inicial se ha decidido crear un sistema ligero de guardado por lo cual necesitamos dicho botón.

Otros botones, como el de créditos, puede estar dentro de, por ejemplo, el menú de opciones así como la cantidad de opciones de dicho menú. Cuando se proceda al diseño definitivo y a su siguiente implementación en Android nos veremos enfrentados a una más que posible modificación de estos componentes.

2.3 - Opciones del menú inicial

Fig.02

Este submenú será accesible desde el menú inicial y una versión más ligera puede que sea también accesible mientras se está ingame. Las opciones posibles que dispondrá este menú son las siguientes:

- Ajuste de volumen
- Ajuste de resolución
- Modos de pantalla
- Idioma
- Enlaces de interés (github del alumno, youtube, redes)
- Créditos
- Otros ajustes

Más adelante cuando nos encontremos en desarrollo dentro de Unity se estudiarán proyectos ya comerciales o manual de Unity así como documentación para poder contrastar y comparar qué opciones son las demandadas por los usuarios a nivel de compatibilidad (resolución, pantalla) o accesibilidad (idioma, volumen).

Estas opciones pueden variar dependiendo de si nos encontramos en Windows o en Android y para cada uno de los casos será necesario un estudio preliminar diferente. En ambos escenarios el estudio será afrontado de igual manera, consultando qué es lo que necesitan las aplicaciones ya creadas para disponer de una base real de lo que demanda el mercado en temas de opciones/accesibilidad. Una vez identificadas dichas necesidades se intentará en la medida de lo posible cubrir dichas opciones lo máximo posible. Esto es **algo secundario** que se dejará para el final del proyecto. Las opciones básicas para hacer nuestro juego adaptable (responsive) si estarán desde un inicio pero las opciones para que el usuario lo configure se implementarán más adelante.

2.4 - Pantalla principal de juego

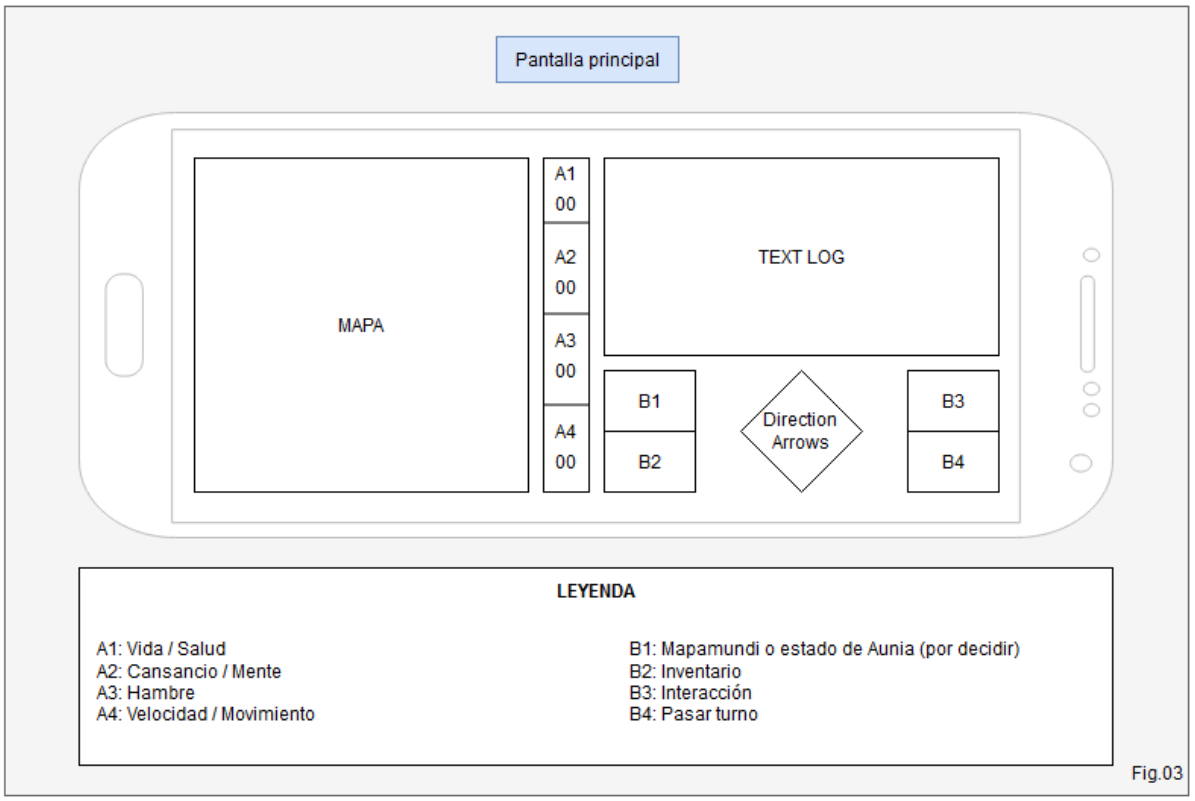


Fig.03

Esta será la pantalla en la que se realizará principalmente la mayor parte del uso de la aplicación. Consta de una interfaz semi-dinamica (mejor explicada en el punto 3) dentro de la cual tenemos todos los elementos para la interacción del jugador con el entorno virtual.

La interfaz se divide en partes visuales para mostrar datos y partes interactuables para modificar parámetros de juego.

En las partes visuales tenemos tres secciones:

- Mapa
- Área de texto (registro)
- Indicadores de estado

Estas tres secciones no son modificables por el usuario de forma directa pero se verán influenciadas por sus acciones. Por ejemplo en el apartado de indicadores de estado disponemos de 4 indicadores, Vida, Cansancio, Hambre y Acciones de movimiento. Las acciones de movimiento se verán directamente reducidas cada vez que el usuario use las teclas de movimiento dado que esto confirma que queremos desplazar al personaje, restando un punto de movimiento al realizar dicha acción.

En las partes interactuables del menú principal de juego tenemos:

- Teclas de dirección
- Mapamundi / Estado
- Inventario
- Tecla para interactuar
- Terminar turno

Estos botones pueden ser pulsados por el jugador y es la forma de controlar al personaje e informarnos, mayormente, sobre lo que sucede en el juego.

2.4.1 - Ventanas emergentes

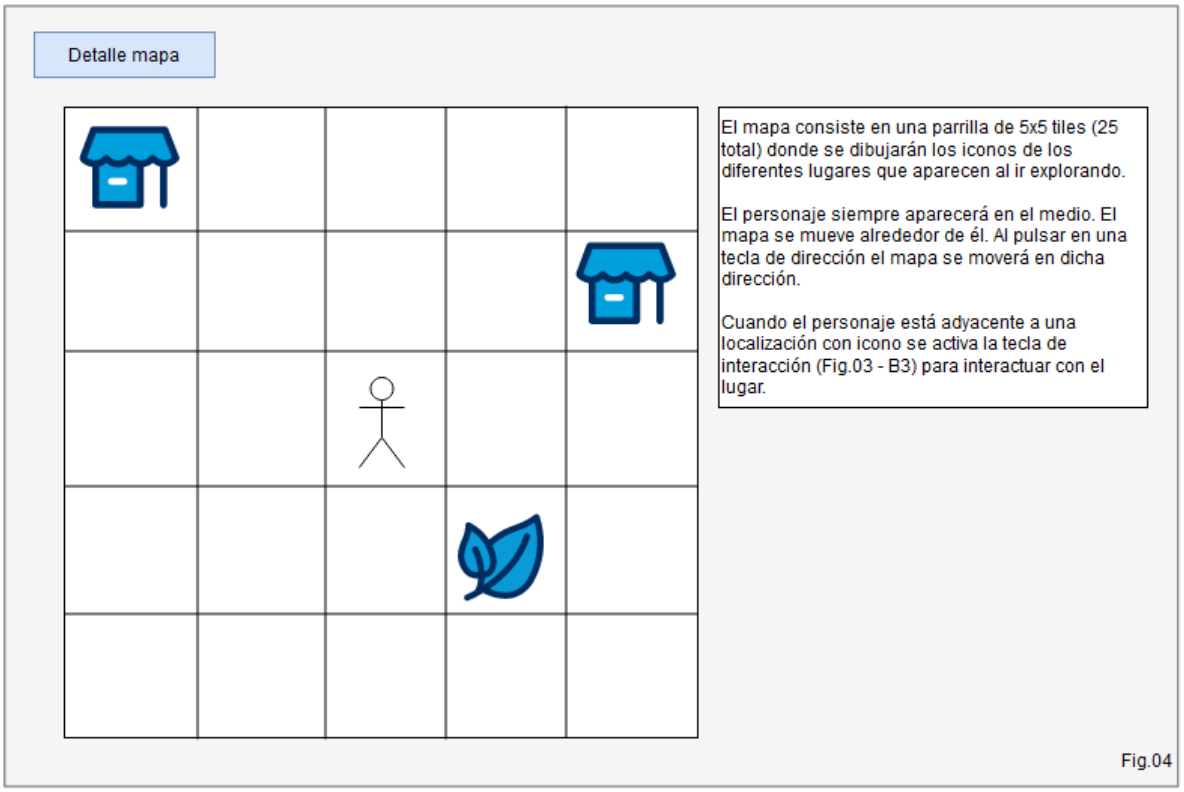
En la pantalla principal tenemos un área para el **registro de texto** (text log) donde se mostrarán mensajes breves de información al jugador como por ejemplo “Tu personaje se encuentra cansado” o “Has descubierto X lugar” según te acercas en el mapa. Pero, generalmente, cuando se tiene que mostrar información más detallada como cuando interactuamos con un lugar se abrirá una ventana emergente.

Fig.06

Esta ventana se superpondrá a la interfaz de juego bloqueando el acceso a los botones vistos anteriormente y nos muestra, por ejemplo, una decisión a tomar por el jugador.

Hasta que no se haya terminado el flujo de uso de esta ventana no se nos devolverá a la anterior.

2.4.2 - Detalle del mapa de juego



Una de las secciones más importantes de la pantalla principal de juego es el mapa. Este mapa no es más que una **representación visual de una matriz** que se moverá en nuestro código (sección 4 de este documento).

En dicha matriz es donde realmente sucede el control del personaje, interacción, posición de lugares etc. Nuestro personaje será una especie de “puntero” dentro de esa matriz que indicará si podemos hacer alguna acción, si llegamos a los límites del mapa, si debe colisionar con algo etc.

Cuando el personaje se encuentra adyacente a una posición de la matriz donde hay algo (entiéndase por adyacente arriba, abajo, derecha, izquierda, no ortogonalmente) el botón de interactuar de la pantalla principal estará activo y se podrá interactuar con dicha localización.

Una vez interactuado se abrirá una ventana emergente (punto 2.4.1) con la información relevante de la localización visitada así como de sus opciones.

Toda esta información es generada de manera semi-procedural al iniciar una partida nueva tomando datos de nuestra base de datos (punto 3.1).

2.4.3 - Inventario

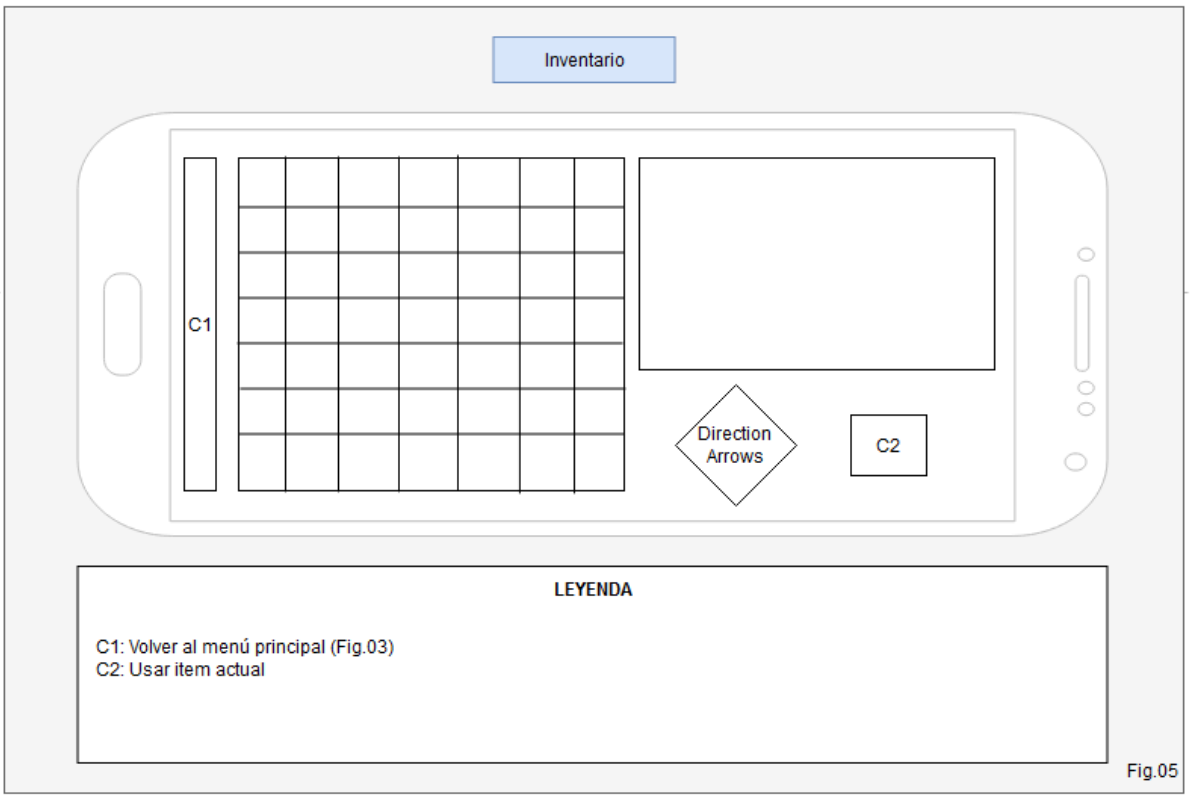


Fig.05

Desde la pantalla principal de juego se puede acceder también a la pestaña de inventario. Esta opción estará habilitada desde el inicio del juego y siempre que el flujo de juego se encuentre activo menos cuando una ventana emergente bloquee la entrada de nuevas acciones por parte del usuario.

La ventana de inventario se compone de una celda a modo de lista visual donde se mostrarán los iconos de los objetos que nuestro personaje lleva encima. Disponemos de unas flechas para movernos por el inventario, una sección de texto donde ver definición del objeto y una tecla de “usar”.

La tecla de usar tiene dos estados, puede estar activada o desactivada. Esto viene dado porque existen dos tipos de objetos, principalmente. Objetos consumibles (comida, vendajes, bebidas etc) y objetos no consumibles (llaves, piezas de ropa, monedas). Los objetos consumibles habilitan la tecla de “usar” para consumirlos y al hacerlo aplicarán un efecto concreto (curación, descanso etc) y desaparecerá una unidad de ese objeto de nuestro inventario. Muchos objetos, por ello, también son acumulables y veremos qué cantidad nos quedan.

Los objetos no consumibles como por ejemplo una llave son un tipo de bien que nuestro personaje lleva en el inventario y se pueden inspeccionar para ver su descripción. La lógica del luego será la encargada de quitarnos estos objetos. Por ejemplo al abrir una puerta nos quitará la llave del inventario si es necesario.

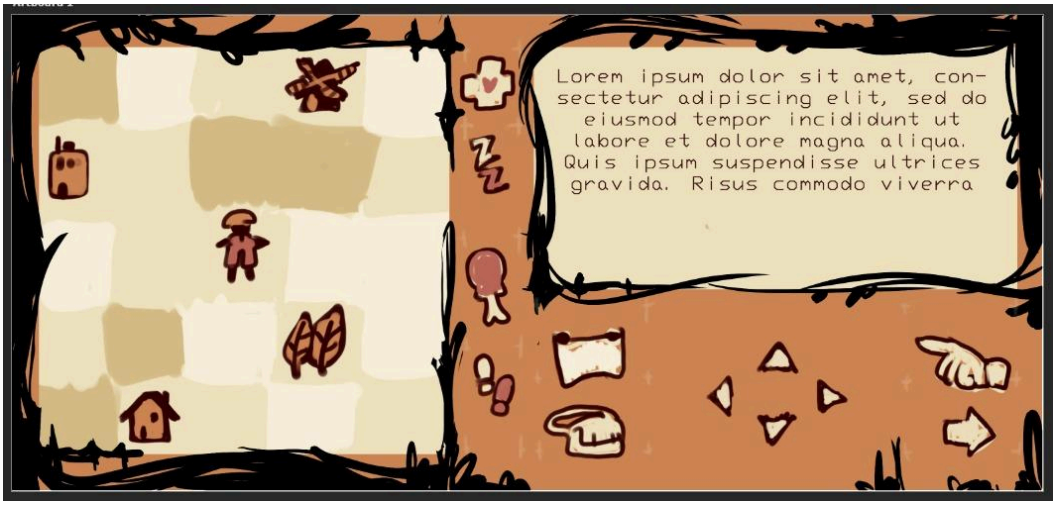
2.5 - Otras pantallas

Es posible que durante la realización del proyecto sea necesario alterar, modificar, eliminar o añadir componentes gráficos tanto parcial como de forma completa siendo así que quizás un menú o submenú definido no exista en la versión final o que exista uno que aquí no ha sido definido. Esto es algo que se contempla dado el corto periodo de desarrollo que se tiene para poder realizar el proyecto y que algunas dificultades o nuevos requisitos pueden surgir cuando se esté desempeñando el trabajo.

2.6 - Bocetos de la interfaz

Los bocetos de la interfaz han sido realizados por **Irene Lloret Miguel** en colaboración conmigo. En el caso del alumno ha facilitado las directrices para la creación (los esquemas vistos en el punto 2) y ella se ha encargado del arte. Cabe la posibilidad de consultar con ella para recursos gráficos.

Primero se muestran en grandes dos de los posibles aspectos de la interfaz para apreciar el esquema de colores o los iconos.



A continuación se muestran todos los bocetos realizados con diferentes esquemas y/o iconografía.

Debido, una vez más, al corto periodo de desarrollo que enfrenta el proyecto **no hay garantía en que la interfaz gráfica pueda tener un aspecto tan pulido** como se pretende o da a entender en la visualización de estos bocetos.

Se priorizará el funcionamiento de la aplicación así como de todos los scripts que la componen además de que sea posible iniciar, jugar y finalizar una partida completamente.

Posiblemente los recursos gráficos usados vendrán de fuentes gratuitas o que permitan licencia no comercial / educativa, se define en el siguiente punto.

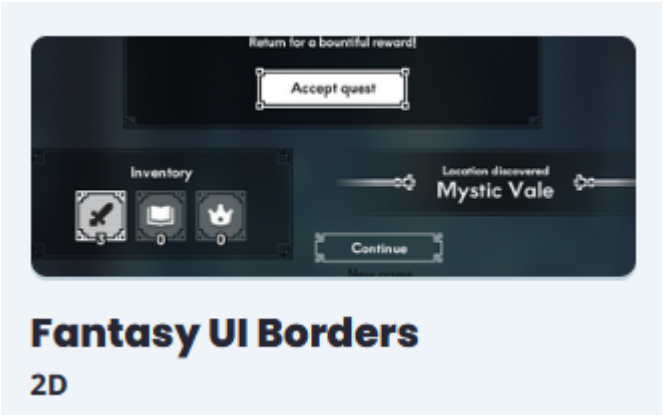
2.7 - Recursos gráficos gratuitos

Existen multitud de cuentas y/o webs de recursos gratuitos tanto con licencia 100% gratis tanto para uso comercial / no comercial como para uso educativo.

En la medida de lo posible se intentará disponer de estos recursos para la creación del juego y así evitar dedicar tiempo a esta tarea. A continuación se citan algunos ejemplos contemplados para el proyecto pero se usarán muchos más, todos serán debidamente acreditados en el documento de fuentes final.

2.7.1 - Kenney free assets

Casi todos los recursos que se usarán en el proyecto provienen de la web kenney.nl cuyo autor se dedica a la creación de recursos gráficos **100% libres para todo uso y pueden ser modificados** por otros usuarios.

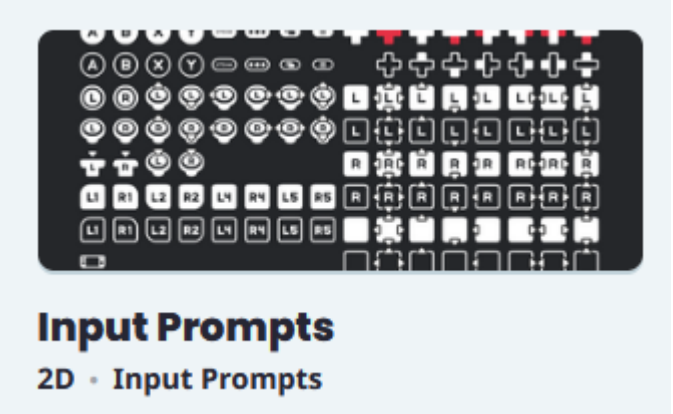


Un ejemplo de los assets que se pueden usar es el siguiente paquete de gráficos para interfaz que cuenta con diferente diseño de botones o layouts para crear las secciones interactuables.

En caso de que no dé tiempo a diseñar una interfaz propia, se usará este recurso.

Otro paquete que posiblemente se usará será el de Input Prompts con diferentes botones que nos permiten ahorrar tiempo a la hora de diseñar iconografía específica de los diferentes sistemas operativos.

Este paquete nos trae diferentes teclas equivalentes al control en PC, con mando o móvil.

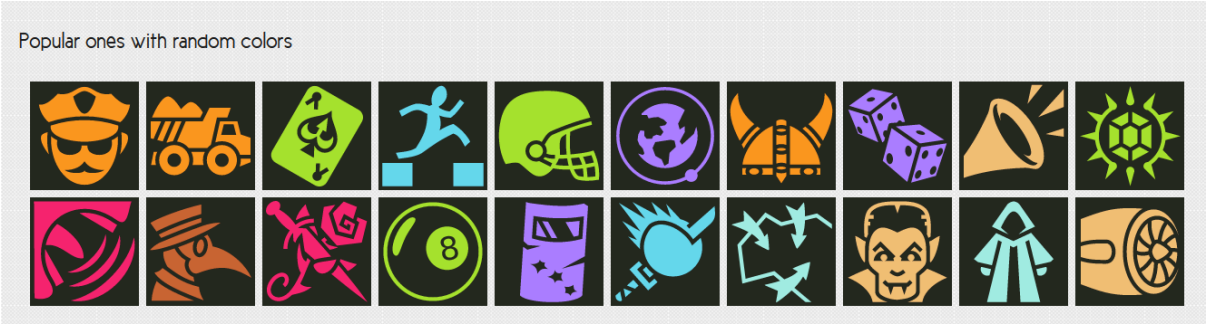


Otro ejemplo es el paquete **1-bit pack** que dispone tiles de un estilo muy similar al que se pretende replicar en nuestro juego.

Como podemos ver al pie de página todos los recursos gráficos de Kenney son totalmente libres para el uso que necesitamos. Es muy posible que dichos recursos sean modificados o usados de base para la creación de recursos propios pero son un gran punto de partida para poder diseñar lo que el proyecto necesita sin tener que dedicarle una gran cantidad de tiempo a este paso que, normalmente, en el desarrollo de un videojuego lleva todo un equipo creativo detrás.

2.7.2 - Game-Icons

La web de game-icons es un repositorio 100% gratuito con diferentes iconos sencillos en 2D que pueden ser usados tanto comercial como no comercialmente. Se pueden también modificar y usar de base para otros trabajos. Posiblemente se usará esta web para, por ejemplo, iconos de los objetos.



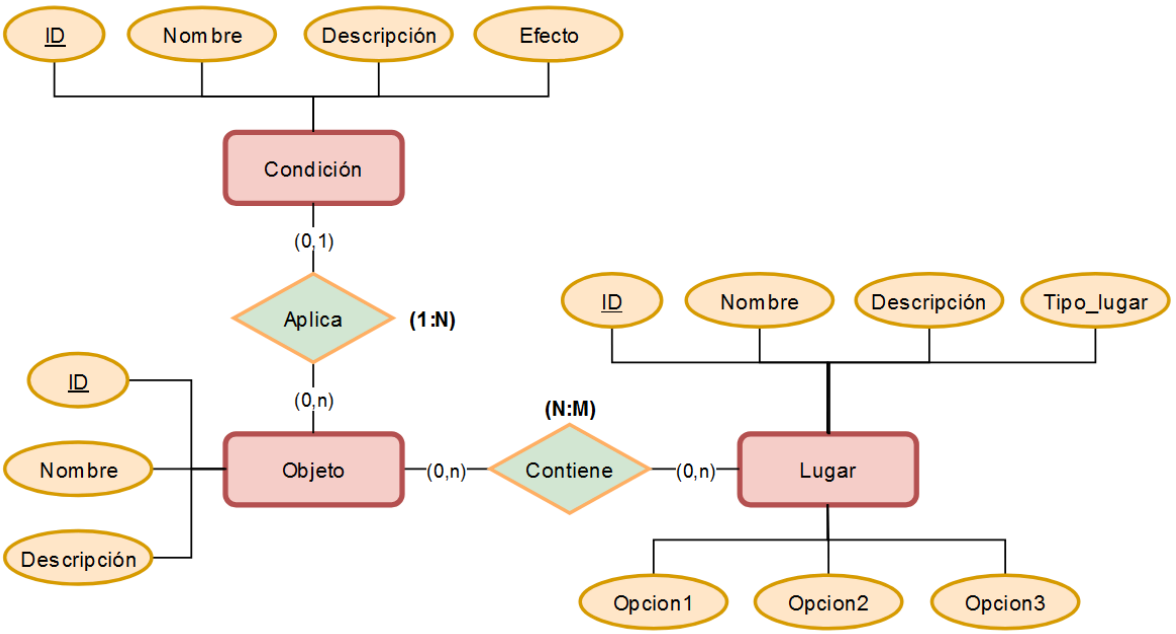
3 - Base de datos

3.1 - Creación inicial

Para la creación inicial de nuestro juego vamos a disponer de una base de datos con toda la información de los estados, objetos y lugares que van a poblar el juego. La idea es que nuestro juego se conectará con una base de datos online y volcará toda la información en el sistema generando así los objetos y luego aleatorizado el mapa.

3.2 - Modelo E/R

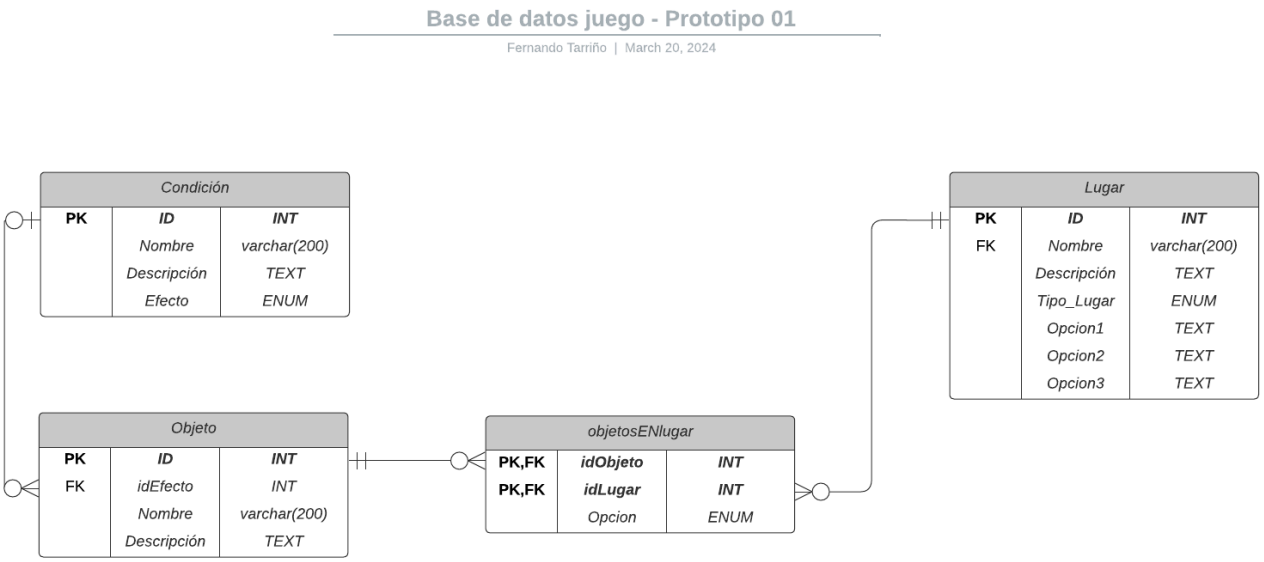
Es importante destacar que esta base de datos es **provisional** y aún no está terminada. La base de datos definitiva se verá modificada. Esto es debido a que, antes de terminar la base de datos y crearla correctamente, debemos probar **que la conexión con Unity** así como la transformación de **datos en objetos** se hace de forma correcta tanto desde Windows como desde Android. Una vez que las pruebas han terminado y se comprueba que todo funciona correctamente con una base de datos sencilla se pasará al diseño de la base de datos definitiva.



Podemos ver que se ha generado una relación de muchos a muchos entre objeto y lugar que debemos solucionar, esto será tratado en la siguiente sección.

3.3 - Modelo relacional

En el modelo relacional vemos cómo podemos solucionar el problema que nos ha surgido en el esquema E/R con la tabla de muchos a muchos.



Se ha creado una tabla intermedia llamada “objetosENlugar” que almacenará qué objeto pertenece a qué decisión.

Si bien **esta tabla no se creará en el prototipo** porque vamos a aprovechar esta ocasión para aprender a tratar tipos nulos dentro de C# que es el lenguaje de programación que usa Unity. Con ello veremos cómo traer estos registros y transformarlos en objetos de C#.

+ Opciones

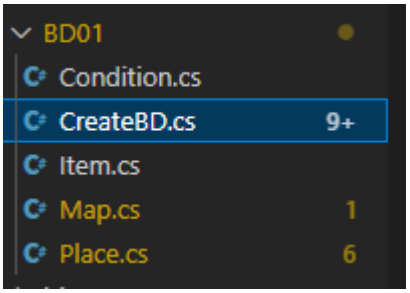
	idPlace	Name	Description	PlaceType	Option1	Object1	Option2	Object2	Option3	Object3
Editar Copiar Borrar	1	Place1	Description1	empty	Place1_option1	1	Place1_option2	2	NULL	NULL
Editar Copiar Borrar	2	Place2	Description2	empty	Place2_option1	NULL	3	NULL	NULL	NULL
Editar Copiar Borrar	3	Place3	Description3	forest	Place3_option1	4	Place3_option2	5	Place3_option3	NULL
Editar Copiar Borrar	4	Place4	Description4	empty	Place4_option1	NULL	Place4_option2	NULL	6	NULL
Editar Copiar Borrar	5	Place5	Description5	forest	Place5_option1	NULL	NULL	NULL	NULL	NULL
Editar Copiar Borrar	6	Place6	Description6	forest	Place6_option1	7	Place6_option2	8	Place6_option3	9
Editar Copiar Borrar	7	Place7	Description7	empty	Place7_option1	NULL	Place7_option2	NULL	10	NULL
Editar Copiar Borrar	8	Place8	Description8	empty	Place8_option1	11	Place8_option2	12	Place8_option3	NULL
Editar Copiar Borrar	9	Place9	Description9	forest	Place9_option1	NULL	NULL	NULL	NULL	NULL

En la base de datos definitiva del proyecto si se realizan correctamente estos ajustes, creación de tablas intermedias etc.

Es importante recordar que **está no es la base de datos definitiva** si no una base de datos temporal creada para comprobar que todo funciona correctamente.

3.4 - Conexión con C#

Antes de probar la conexión con Unity se ha realizado la conexión con C# usando un ide externo (VSC) y se montará la lógica básica tras el programa en el código. Esto quiere decir que en lugar de dibujar en pantalla con Unity usaremos salidas de consola y texto plano.

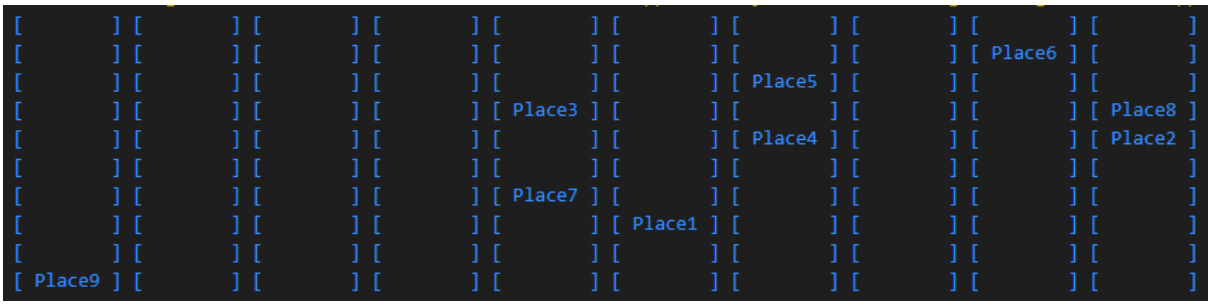


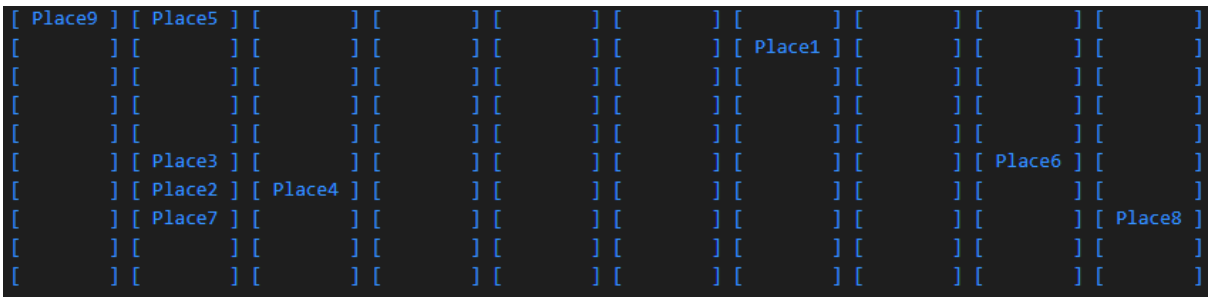
Se han creado los objetos equivalentes a las tablas (Condition, Item y Place) y mediante la clase CreateBD.cs nos conectaremos a la base de datos e iremos leyendo su contenido para posteriormente volcar dicho contenido en las clases correspondientes.

El objetivo final de esta clase (CreateBD) es devolver una lista de objetos Place donde se encuentre toda la información que necesitamos, así pues dentro de dicha lista tendremos la relación con los objetos que hay en esos lugares y, dentro de esos objetos, las condiciones que aplican.

Como podemos ver esto es un proceso **sencillo y simple** en comparación con el **objetivo final**. Podemos definir a este prototipo como **una versión ligera del proyecto** donde comprobaremos que todo funciona. Por ello se ha optado por no solucionar los problemas de la tabla intermedia en el punto anterior para evitar dedicarle más tiempo.

El objeto **Map** guardará una matriz de objetos Place entre los cuales se generan posiciones aleatorias para los recuperados. A continuación dos ejemplos dibujados por consola.





Se puede apreciar que la distribución es totalmente diferente pero el contenido de cada objeto Place será el mismo dado que ha sido recuperado de la base de datos.

3.5 - Elección de la base de datos

Este es uno de los retos más grandes a los que se enfrenta el proyecto. La dificultad viene dada por que nuestro programa debe funcionar tanto en Windows como Android.

En caso de que el programa fuese solo en Windows podemos usar XAMPP para gestionar una base de datos SQL de forma local sin más problema. Pero debemos poder conectarnos desde Android también. Es por ello que se ha decidido buscar una base de datos **online** que pueda ser consultada en todo momento.

Existen muchas opciones entre las que se están valorando:

- Planetscale
- Cockroachlabs
- YugabyteDB
- Fauna
- Render
- Supabase
- Hetzner
- Freesqldatabase

Todos estos servicios disponen de diferentes opciones **gratuitas** para montar nuestra base de datos SQL o PostgreSQL de forma online y conectar con ella si bien tienen bastantes limitaciones como el tamaño de la base de datos o que si no se usan en un tiempo determinado, el cual suele rondar la semana, se borran.

Por ello se está también valorando la opción de **configurar un VPS** gratuito e instalar dentro la base de datos, configurar la conexión y acceder a dicho servicio. Algunos de los servicios considerados son:

- AmazonRDS
- AzureSQL
- ElephantSQL
- OracleCloud

La información está siendo consultada en diversas fuentes como por ejemplo foros de GitHUB, youtube, investigación en Google etc.

Si bien la cantidad de tiempo que hay que dedicar a la configuración del VPS es bastante grande y puede retrasar otras secciones del proyecto tiene una importancia vital para el buen funcionamiento del juego.

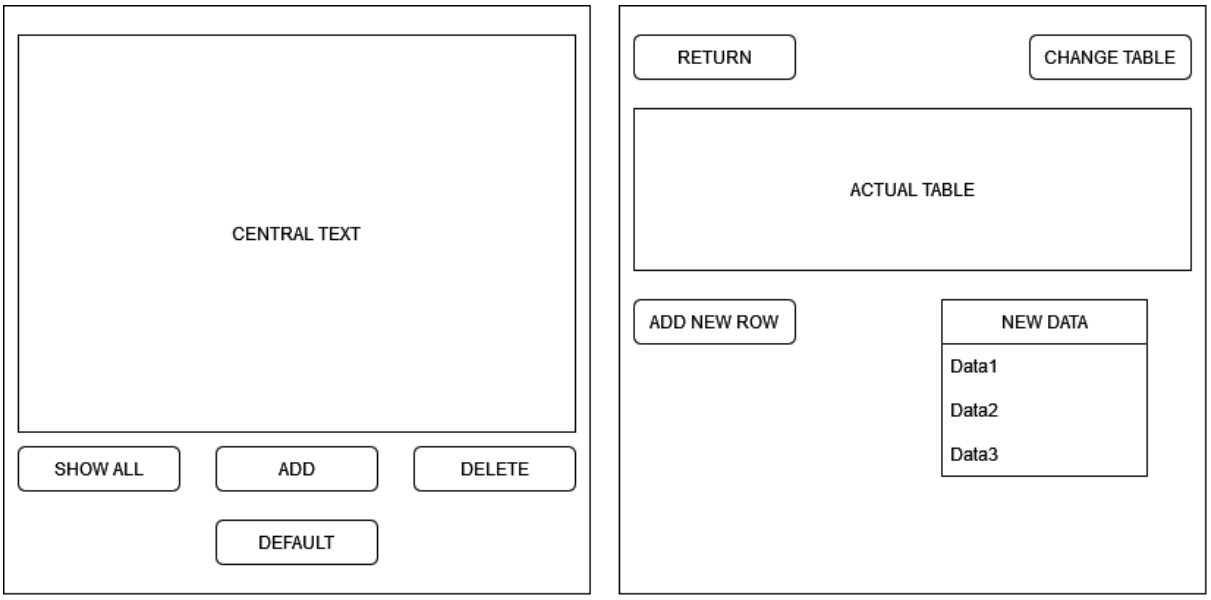
Esta sección **aún está en construcción** y es una de las decisiones que quedan pendientes de realizar.

3.6 - Programa independiente edición BD

Dispondremos de un programa ejecutable completamente independiente del juego de Unity para editar y consultar la base de datos. Dicho programa estará hecho en **java** y para el procesado gráfico y de interfaz usaremos **java swing**.

El propósito de este programa es simular la etapa en la que el equipo de diseño de mundo, equipo que no tiene que conocer absolutamente nada de programación, creará las localizaciones, objetos, contenido, texto etc.

Al iniciarlo nos pedirá los datos de la base a la que queremos conectarnos y podremos visualizar su contenido así como reiniciar a estado de fabrica (cargando un script), añadir datos o borrar datos.



El esquema mostrado es altamente mejorable y solo trata de reflejar una versión preliminar de lo que será la herramienta.

El objetivo por el cual se crea esta herramienta es, principalmente, la modularidad de nuestro juego. Una vez que todo el sistema funcione correctamente gracias a esta herramienta podremos ampliar todo lo que queramos el juego (añadiendo cientos de lugares o decenas de nuevos objetos por ejemplo) sin necesidad de usar Unity para ello. Siendo esta una barrera que evitamos y permitiendo así que usuarios ajenos al código puedan participar en la creación del mundo.

Por ejemplo, un jugador que disfrute del juego puede usar la herramienta para crear sus propios escenarios, subirlos a la base de datos y que dichos escenarios aparezcan en el juego gracias a la generación inicial que vimos en el punto 3.1 (creación inicial).

4 - Mapa de juego

4.1 - Objetos

Como hemos visto en la sección 3.4 (conexión con C#), el objetivo del programa al crear una nueva partida será conectarse a la base de datos y bajar información para transformar las columnas en objetos. Por ejemplo, el objeto **Place** tiene todas las equivalencias a las columnas de la base de datos. Cabe resaltar que este constructor no pide obligatoriamente los Item dado que, recordemos, pueden ser Null. Al rescatar la información de la base de datos los iremos asignando con los métodos get en caso de ser necesarios. Además el método toString() también controla esto. En las pruebas es importante dado que así sabemos si todo funciona correctamente.

```
public class Place
{

    6 references
    public string Name { get; }
    3 references
    string Description { get; }
    3 references
    string Type { get; }
    6 references
    public string Option1 { get; set; }
    3 references
    public string Option2 { get; set; }
    3 references
    public string Option3 { get; set; }
    6 references
    public Item Item1 { get; set; }
    3 references
    public Item Item2 { get; set; }
    3 references
    public Item Item3 { get; set; }

    1 reference
    public Place(string n, string d, string t)
    {
        Name = n;
        Description = d;
        Type = t;
    }
```

Todos estos objetos tendrán que ser adaptados en un futuro cuando trabajemos con Unity pero en el prototipo se pretende conectar, construir y comprobar funcionamiento.

4.2 - Matriz

```
11 references
Place[,] map;
8 references
int height;
8 references
int width;

1 reference
public Map(int height, int width)
{
    this.height = height;
    this.width = width;
    map = new Place[width, height];
}
```

Tanto ahora como en Unity nuestro personaje no es más que un “puntero” que se mueve dentro de una matriz de objetos de tipo Place. Esto nos indica dónde se encuentra y que hay a su alrededor.

El objeto Map genera dicha matriz con unos argumentos específicos para su altura y anchura.

Una vez generado rellena la matriz y permite diferentes acciones sobre ella.

4.3 - Programa de prueba

Nuestro programa de pruebas se encargará de instanciar un objeto CreateBD y llamar a su método create() que devuelve una lista de Place para posteriormente generar un mapa de 10x10 y rellenarlo.

En el código a continuación podemos ver el flujo del código. En el apartado 3.4 (conexión con C#) ya vimos el resultado al imprimir por pantalla el objeto.

```
0 references
static void Main(string[] args)
{
    List<Place> places = new List<Place>();
    CreateBD cbd = new CreateBD();
    places = cbd.create();

    Map map = new Map(10, 10);
    map.fillMap(places);
    Console.WriteLine(map);

}
```

Todo este flujo de código se adaptará posteriormente a Unity pero actualmente lo que se pretende es comprobar el correcto funcionamiento de la base de datos así como la conexión con C# y la transformación en objetos de las tablas. Es por ello que todo lo mostrado en este prototipo tiene fallos, información temporal y, mayormente, errores de estructura.

Dichos problemas se solucionarán desde un principio en la versión real de Unity.

5 - Historia

La historia y trasfondo del juego han sido escritos de forma superficial y rápida pero se quiere dedicar un tiempo prudencial al cuidado de este aspecto para poder dotar de mayor interés al proyecto.

A nivel narrativo el jugador no conocerá muchos detalles de la trama si no que estos se irán desvelando según desarrolla la aventura. Nuestro personaje protagonista es una chica llamada **Aunia** que se enfrenta a un escenario semi-desierto donde no encontrará otras entidades vivas.

El aspecto general del mundo sobre el que se mueve es postapocalíptico y el jugador no tendrá información inicial de donde está o por que se encuentran en ese estado los lugares que visita. El objetivo principal es que Aunia sobreviva a las inclemencias que se le presentan como el cansancio, encontrar comida y otros eventos extraños.

Además de ello hay un objetivo final que el jugador debe descubrir mientras explora el mundo que podemos resumir, de forma sencilla y explicativa, en que debe llegar a un lugar concreto antes de que Aunia no pueda continuar.

Todos los detalles de la trama se podrán entender por la descripción de los lugares. Aunia nos contará como si estuviese hablando consigo misma mediante el área de texto o las descripciones de los sitios que visita en las ventanas emergentes recuerdos del entorno o pequeñas piezas de conocimiento para que sea el propio jugador el que construya la historia al completo.

Podemos definir que la historia en realidad empieza por el final, todo ha sucedido y lo único que debemos hacer ahora es lograr que el personaje sobreviva lo suficiente para realizar la última tarea mientras que, a su vez, nosotros descubrimos el por que todo se encuentra en ese estado y tomamos diferentes decisiones que pueden tener un impacto directo en el mundo o sobre Aunia.

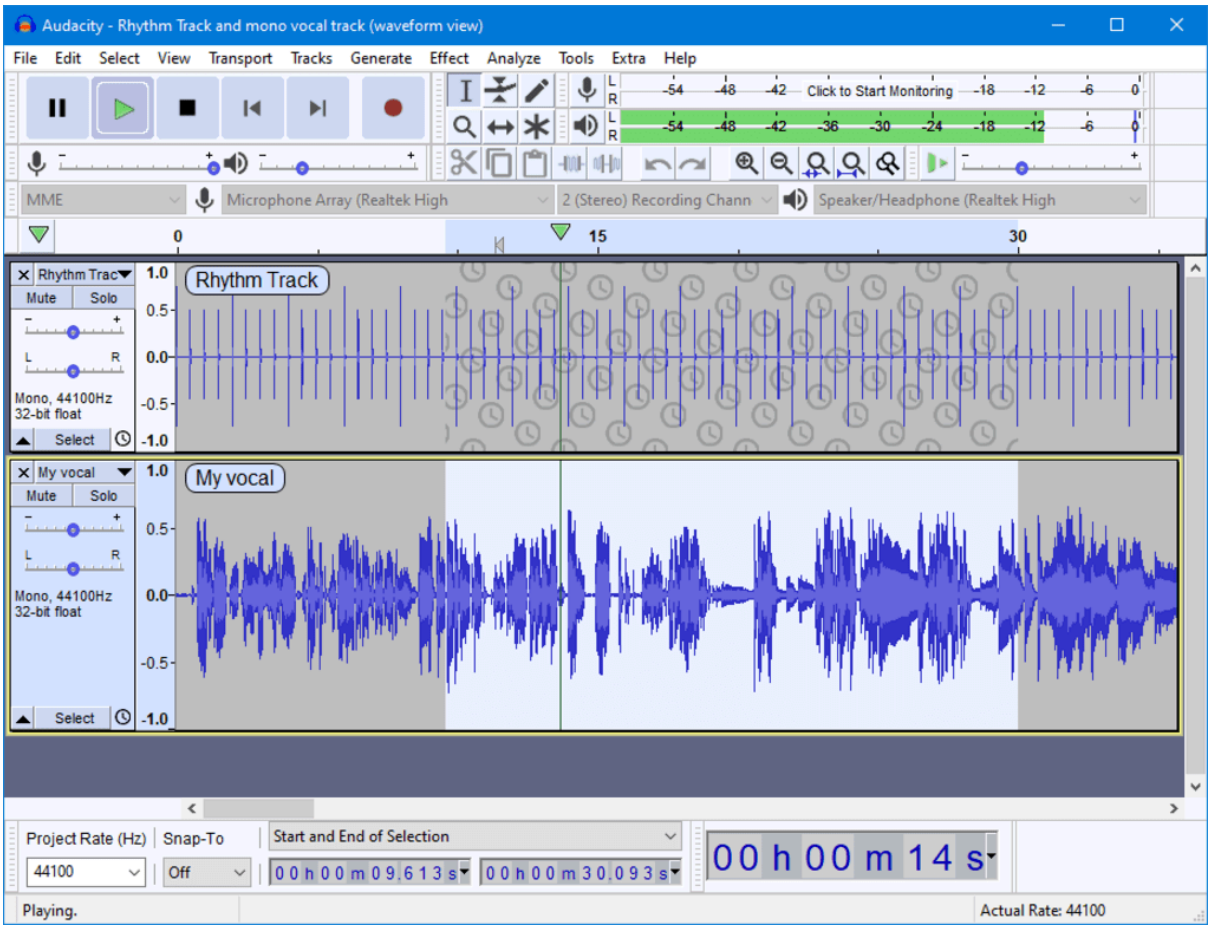
6 - Sonido

Todos los efectos de sonido así como sonido ambiental serán obtenidos de **bibliotecas y repositorios gratuitos** tanto para fines **comerciales** como **no comerciales**.

Los derechos referentes a todo lo que engloba el mundo musical son bastante delicados por ello se optará por disponer de licencias 100% libres.

Además de ello cabe destacar que se usarán diferentes programas de edición tanto de audio como de imágenes o vídeos. Siempre se usarán programas **libres y/o open source** para ello evitando el uso de herramientas de pago.

Por ejemplo para la edición de audio se emplea el programa Audacity o para edición de imágenes Gimp.



Todas las webs, repositorios y programas usados para sonido se encuentran en el documento de referencias.

7 - Referencias y fuentes

Todas las fuentes y referencias consultadas tanto en el documento como en la creación del proyecto, el archivo Análisis iniciales y trabajo relacionado se encuentran debidamente acreditadas en un documento de acceso público llamado “**FuentesIniciales.pdf**” que puede ser consultado en el siguiente enlace:

[Enlace al PDF online]

One last time before Edereta devours the world

Fuentes del proyecto

Imagenes	
pixabay001.jpg	mila-del-monte en Pixabay
Bocetos de interfaz	Irene Lloret Miguel (Personal externo)
Kenney assets	kenney.nl
pixabay002.jpg	icheinfach en Pixabay

Tipografía	
SourceCodPro	Google Fonts
Old Newspaper Types	Manfred Klein en Dafont

Personal externo	
Irene Lloret Miguel	Linktr.ee

Juegos de ejemplos		
Caves of Qud	Freehold Games	Steam
Dungeons of Dredmor	Gaslamp Games	Steam
Pixel Dungeon	Watabou, Retronic Games	Itchi.io

Enlaces de referencias / Programas	
Roguelike	Artículo de Wikipedia
Roguelike	Artículo de Devuego
Audacity	Web oficial del producto
Freesound	Web oficial
Pixabay	Web oficial
Game-Icons	Web oficial

Proyecto fin de grado, Fernando Tarriño del Pozo. DAM-2023-2024 - Madrid