Roguelike:

Género de videojuegos orientado a que el jugador se adentre en una mazmorra en la que se crea de manera procedimental cada nivel.

Suelen estar ambientados en mazmorras, pues es sencillo adaptar esta ambientación a la generación aleatoria del entorno.

El nombre del género deriva del nombre de su precursor: Rogue (1980).

Combate por turnos:

Derivado del género TBS (Turn-based Strategy), el combate por turnos viene de juegos de mesa tales como el ajedrez.

Gracias a la inclusión del ordenador y de la posibilidad de que la máquina calculara y arbitrara el juego, surgieron este tipo de juegos. Ejemplos del combate por turnos pueden ser el final fantasy, Bravely Default, Fire Emblem, Suikoden, etc.

Pasamos a qué necesita este juego para funcionar. Pues bien:

El juego necesita de una generación aleatoria de mazmorras. Gracias a esto, el jugador tiene asegurada una partida completamente distinta cada vez que empieza a jugar.

Cada nivel aumenta la dificultad de los enemigos y el tamaño de las mazmorras, independientemente de la generación aleatoria de los mismos.

El combate. Es clave que este juega posea una función de combate, ya que el objetivo principal es derrotar a los jefes finales de cada piso y adentrarse más en la mazmorra hasta llegar al último nivel.

El combate debe ser por turnos ya que actualmente se ofrecen muchos juegos con mecánicas de combate en tiempo real, pero casi ninguno o ninguno con combate por turnos.

La tienda. Dada la naturaleza del juego, el jugador se irá encontrado con monedas y otros objetos que puede utilizar.

La tienda ofrece al jugador la oportunidad de gastar ese dinero en mejoras para su equipo y así hacer más sencillo el avance entre los niveles. Además, también deberá de permitir al jugador vender dichos objetos para así conseguir más dinero y realizar estrategias en base a sus objetos para una partida más eficiente y dinámica.

Por último, es clave que el jugador tenga a su disposición un inventario en todo momento. De esta manera podrá gestionar sus recursos de manera cómoda y sencilla, a la vez que dándole más sensación de control sobre la partida al jugador en todo momento.

La metodología ágil.

En este proyecto se ha utilizado la denominada Metodología Ágil. Esta metodología es muy común entre los proyectos de videojuegos ya que permite testeos inmediatos y constantes, a la vez que una base jugable desde el principio.

La Metodología Ágil se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

En líneas generales, se hace una lista PRIORIZADA con las cosas que hay que hacer para el proyecto. Se planifica un tiempo estimado para la duración de la iteración del ciclo de vida actual y se procede a ir haciendo las cosas de la lista hasta que no quede ninguna o se haya acabado la iteración.

Cada elemento de la lista debe ser acabado correctamente y sin errores para poder proceder al siguiente. Posteriormente se hacen pruebas con los nuevos módulos implementados y el ciclo vuelve a comenzar.

Esta captura de un txt es un ejemplo de las listas que estaba utilizando personalmente a la hora de ir haciendo el proyecto.

Podéis observar como está ordenada por prioridades, con el tiempo estimado de cada tarea y justo debajo anotaba lo importante sobre ese día de trabajo para el siguiente volver a priorizar los elementos restantes y así poder presentar el juego con sus características más importantes.

El ciclo de vida del juego, en base a esta metodología ya mencionada se basa en repetir indefinidamente estas etapas:

1. Analizar lo que necesita el juego.
2. Posteriormente diseñarlo.
3. Proceder a desarrollarlo.
4. Integrarlo con el resto de módulos.
5. Hacer las pruebas pertinentes para asegurarse de que todo funciona correctamente.

Si algo no funcionaba bien o necesitaba añadirle más cosas, se volvía a la fase de analizar y se repetía el ciclo.

Ahora vamos a hablar un poco sobre Unity, el motor de juego que he utilizado para desarrollar este juego.

Unity es, como ya he dicho, un motor de videojuego multiplataforma. Unity puede ser utilizado junto a Blender, 3DS Max, ZBrush, Cinema 4D, Photoshop y demás programas de diseño gráfico. Los cambios realizados a los objetos creados con estos productos se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.

El motor gráfico utiliza OpenGL, Direct3D y demás. Tiene soporte para mapeado de relieve, reflejos, paralaje, oclusión ambiental, sombras dinámicas, efectos de post-procesado y renders de textura.

En el apartado de scripting, todo deriva de Mono. Mono es la implementación de código abierto de .NET Framework. Se puede trabajar en Javascript, C# o Boo (Con sintáxis inspirada en Python), aunque Javascript y Unityscript están descontinuados desde hace un año.

Como pros de este motor de juego podemos destacar.

* Su funcionalidad multiplataforma. Es muy sencillo hacer los juegos multiplataforma desde Unity con no muchas cosas a tener en cuenta durante el desarrollo.
* Es sencillo de utilizar, hasta cierto punto. Es necesario hacerse a las peculiaridades del motor, tales como las físicas del mismo.

El orden de ejecución de los métodos y componentes de un objeto puede parecer muy arbitrario a primera vista, y es necesario utilizar muchas horas para entender cómo funciona realmente todo.  
Como ejemplo podría poner el tiempo de reacción de las físicas. Mientras que en un entorno de desarrollo tradicional no es muy complejo desarrollar varios módulos a la vez, en Unity hay que tener en cuenta el tiempo de reacción de las físicas y de carga del motor, como también la potencia que la máquina objetivo del juego pueda manejar. Es aquí cuando tienes que hacer que muchos métodos se ejecuten en segundo plano 0.001 segundos desfasados para que le de tiempo al resto de componentes a cargar y poder colisionar e interactuar con los demás.

* El mercado de Assets es una muy buena funcionalidad de Unity que, aunque yo no haya usado demasiado, he podido observar el potencial que tiene.

Tienes a tu disposición decenas de miles de assets y funcionalidades para el propio programa creadas por la comunidad a un click de distancia. Muchas siendo de pago, evidentemente.

* C#. Como he mencionado antes, en Unity se puede programar con C#. Y eso es un gran punto a favor. Como todos hemos ido viendo a lo largo del curso C# es un lenguaje de programación muy potente y también muy sencillo de utilizar. Gracias a esto y a la implementación de Mono, Unity tiene las funcionalidades de .NET Framework para implementar en cualquier proyecto.
* Se puede trabajar en 2D y en 3D. Tal como indica el nombre completo de Unity; Unity3D, está pensado para hacer proyectos con renderizado 3D, pero eso no impide el uso del 2D como podéis observar en este proyecto. La manera que Unity trata el desarrollo 2D es de la misma manera que el 3D, pero eliminando el eje Z para los sprites.

Ahora pasemos a los contras que he encontrado con este motor.

* La documentación. Es cierto que Unity posee una amplia documentación de todas funcionalidades, pero también es cierto que mucha de ella está deprecada o simplemente ha sido borrada de la API. Esto dificulta mucho el hecho de buscar la manera actualizada de hacer las cosas, pues sólo te puedes guiar por comentarios de la gente (los cuales también dependen de la fecha en que hayan sido creados).
* Mala gestión de memoria. Esto es algo que me di cuenta mientras estaba desarrollando, y es que Unity sufre en términos de memoria a la hora de Debuggear sobre todo.

Unity posee una funcionalidad que te permite ver en tiempo real qué componentes del motor están usando más recursos mientras ejecutas el juego, y gracias a esto se puede observar como casi siempre el 50/60% del uso es el mismo editor de Unity. Esto implica que cuando estás intentando trabajar sobre un elemento en concreto, se consuman tantos recursos que sea completamente imposible pasar la traza y el programa se fuerce el cierre a sí mismo.

* Los bugs. Los bugs han sido un gran problema en el desarrollo de este proyecto.

Unity tiende a sacar actualizaciones cada no mucho tiempo, lo cual es muy bueno. Pero también tiene a cargarse elementos anteriores que funcionaban perfectamente con cada nueva actualización. Un ejemplo muy directo es la paleta de creación de mapas, la cual cuando se actualizó en abril dejo de funcionarme completamente.

Tras investigar sobre el sector, se entiende como un sacrilegio actualizar el motor de juego durante el desarrollo de un juego, pues nunca se sabe qué se puede cargar que ya funcione y mande todo el proyecto a la basura.

El diagrama de clases. No voy a hablar mucho del diagrama de clases ya que he intentado hacerlo de manera condensada y omitiendo muchas clases que no eran interesantes y aún así se ha hecho muy complicado de leer y no creo que vaya a aportar mucho a esta exposición. Por ello mismo voy a comentar los elementos generales y pasaremos al siguiente punto.

El juego está basado en grandes bloques, los cuales contienen bastantes scripts. Están el bloque de guardado, el de personajes, el de objetos, el del combate, el de la tienda, el del inventario, el de la generación aleatoria de la mazmorra y el de los menús.

La gran parte de scripts están relacionados con casi todo pues es necesaria la interacción en tiempo real en todo momento.

De la manera en que Unity trata a las clases es que de serie todas extiendes de MonoBehavoir, y en ellas escribes script asignado al GameObject que tenga esa clase. Los GameObjects son la base de Unity y todo gira en torno a ellos, por lo que si quieres comunicarte con otra clase, primero debes encontrar el GameObject que tiene pegado ese script, y después recuperar el componente del script. Esto deriva en que la gran mayoría de elementos de las clases son públicos para que, o bien otro GameObject pueda interactuar con ellos, o para que puedan ser accesibles desde el propio editor de Unity.

Dificultades y cosas que mejorar.

El hecho de trabajar con una tecnología completamente nueva ha sido clave a la hora de estimar los tiempos de compleción. Todo te toma más tiempo del que estimas, prácticamente todo.

Aunque tengas el esquema en la cabeza de cómo programarías todo, es necesario investigar exhaustivamente la forma en la que eso se hace en este motor de juego, dar con el mejor método para ello y posteriormente pelearte con sus peculiaridades.

Especialmente el hecho de que, al ser un videojuego, programas en tiempo real con muchos componentes actuando individualmente cada uno a su manera.

Desarrollar un videojuego individualmente. Es cierto que algunas de las cosas que voy a mencionar ahora también se dan en programas tradicionales, pero también es cierto que son mucho más determinantes en un videojuego pues afectan directamente a cómo se juega el mismo.

Cuando haces un videojuego por tu cuenta no sólo tienes que diseñarlo y programarlo, también debes tomar cargo de los siguientes elementos:

* Sonido. El sonido en los videojuegos es clave, al igual que en las películas. Es cierto que se pueden tomar librerías de música y efectos de sonido libres, pero eso casi nunca es suficiente ya que estos deben ser personalizados al entorno del juego para una inmersión más rica.
* El arte. El arte de un juego es lo primero que le llega al jugador, por lo que no es ninguna locura que sea uno de los aspectos más importantes de uno. Se aplica lo mismo que he dicho del sonido para este elemento y para casi todos los que voy a mencionar.
* Las animaciones. Por muy buen arte que tenga un juego, si las animaciones no van acordes al mismo este no va a pasar de tener escenografía bonita y va a sacar de la inmersión al jugador con una velocidad increíble.
* El diseño de niveles. El diseño de niveles es la base para que el jugador no se aburra jugando junto a las mecánicas del juego. Es preciso que haya especialistas diseñando dónde debe ir cada arbusto, cada farola, cada enemigo para así ofrecer una experiencia óptima.
* El guión. No todos los juegos precisan de un guión muy profundo ni linear, el juego que estoy presentando es un claro ejemplo de ello. Pero también es cierto que se requiere un mínimo de trasfondo para darle sentido a todo lo que ocurre en el mismo.
* Balance. El balance del juego impacta directamente con la curva de dificultad que tiene el mismo y a cómo el jugador va a afrontar cada escenario. Si el juego está muy desbalanceado (Como lo está The Binding of the Final Legacy), puede hacer muy difícil o incluso imposible avanzar en el mismo si no se dan unas condiciones óptimas y muy poco realistas.
* Estos son sólo unos pocos elementos clave a tener en cuenta a la hora de desarrollar un videojuego, pero hay muchos más. (Diseño de personajes, mecánicas del juego, cinemáticas, etc.)

El tiempo. El tiempo sin duda ha sido el mayor problema de este proyecto. Si bien es cierto que he empezado a trabajar en serio un poco tarde debido a varias razones, también es cierto que todo ha tomado mucho más tiempo del planeado por todo lo que he mencionado antes.

Mi consejo para la próxima vez que haya que hacer un proyecto de este estilo no sería simplemente empezar antes, pues en mi caso poco tiempo podría haber ganado de esta manera. Sino estudiar muy bien cómo funciona la plataforma en la que se va a trabajar y estimar los tiempos de cada módulo con más exactitud. Y además de esto dejar un margen grande para hacer investigaciones durante el desarrollo de este, pues siempre va a ser necesario en muchas ocasiones, sobre todo cuando se trabaja con tecnologías nuevas.

Aquí podéis ver un ejemplo de la cantidad de horas que le he echado al proyecto, sobre todo en el último mes. Como dejé las prácticas, puede centrarme completamente en el desarrollo del juego, al cual le dedicaba entre 10 y 15 horas diarias casi sin falta y sin días de descanso hasta que mi cuerpo no daba para más. A pesar de esto, al juego le falta todavía mucho camino por delante como voy a comentaros ahora.

El futuro del juego.

El juego todavía no está terminado ni por asomo, por lo que me gustaría comentaros los cambios que implementaré en un futuro, si no este verano porque me ponga a trabajar con nuevas tecnologías algún otro.

* Añadir más enemigos junto a sus correspondientes sprites. En el prototipo que os voy a enseñar a continuación hay 3 enemigos y un jefe final, todos con el mismo Sprite y animaciones pues no me ha sido posible diseñarlas ni encontrar a nadie que las pueda hacer.
* Implementación de sonidos. El juego ya tiene música para la exploración de la mazmorra y para el combate, pero ningún sonido ha sido implementado porque el simple hecho de buscar assets acordes al juego gratis es una tarea de semanas.
* Expansión de ítems. Lo que dice el nombre, implementar más ítems con mecánicas únicas fuera de las típicas de aumentar los atributos. Por ejemplo, el objeto de Vju’Umes.
* Correción de balance. En el estado actual del juego está todo muy desbalanceado, o matas a los enemigos muy fácil porque has conseguido un ítem muy fuerte, o todo lo contrario.
* Crear el minimapa. El minimapa del prototipo es un placeholder para poder jugar mientras diseño el minimapa real.
* La venta y el uso de ítems en combate son las dos grandes funcionalidades que no me han dado tiempo a implementar, pero también hay más que me gustaría ir añadiendo, como por ejemplo un sistema de habilidades más complejo o un encuentro con enemigos en la mazmorra más dinámico.

Como conclusión decir que he disfrutado muchísimo con este proyecto a pesar de las dificultades y el estrés que este ha provocado y me gustaría que esto os haya impulsado, aunque sea un poco, en hacer vuestros propios juegos y disfrutar de esta gran industria.

Preguntas. Presentación del juego.