UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Desarrollo de un videojuego roguelike en Unity

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Solís Flores, Javier Andrés

Tutor/a: Alberola Oltra, Juan Miguel

Cotutor/a: Sánchez Anguix, Víctor

CURSO ACADÉMICO: 2022/2023

Resumen

En el mercado de videojuegos existe un subgénero que cada vez ha ido ganando más popularidad: el roguelike. Sin embargo en los últimos años se han hecho tantas variaciones de esta fórmula que muchos elementos que caracterizaban al género se han ido dejando de lado, y como consecuencia hoy en día pocos títulos nuevos satisfacen a ese sector. El objetivo de este proyecto es el desarrollo de un roguelike con una jugabilidad similar a la clásica pero con cambios enfocados a mejorar la experiencia de juego y, a su vez, aumentar la accesibilidad al añadir métodos para ajustar la dificultad en base al rendimiento del jugador. Este es un trabajo colaborativo, dividido entre la parte gráfica y la programación y diseño del juego. En este documento se explica la parte de programación, las decisiones que se han tomado en cuanto a la organización del proyecto y el montaje de algunos niveles para mostrar la jugabilidad y el mundo del juego.

Palabras clave: Videojuego; Roguelike; Procedural; Programación; Unity

Abstract

In the videogame market there is a subgenre that has been gaining popularity over time: roguelike. However in recent years there have been so many variations of the formula that many characteristic elements have been left aside and as a result there are few games nowadays that satisfy that sector. The objective of this project is to develop a roguelike video game with a gameplay close to the original titles but with changes focused on improving game experience and at the same time create more accessibility by adding difficulty adjusting methods based on the player's performance. This is a collaborative work divided in the graphical and the programming and designing parts. In this document the programming part will be explained along with the decisions made regarding organization of the project and the making of some levels to show gameplay and the game's world.

Keywords: Videogame; Roguelike; Procedural; Programming; Unity

Índice general

Índice de figuras

1. Introducción

Existen una gran variedad de tipos de videojuegos. Desde los más antiguos y básicos como “Tetris” hasta los más modernos y complejos como “Elden Ring”. Lo cierto es que con el paso del tiempo los géneros que se han ido cimentando han sabido reinventarse y seguir figurando hoy en día: es el caso de los juegos de “plataformas”, “RPG”, y otros. Incluso con las ramificaciones que han ido surgiendo, han podido convivir con títulos fieles a sus orígenes.

En el año 1980 se desarrolló el videojuego “Rogue”. Entre sus varias características se encuentran la exploración, la administración de recursos, el movimiento basado en casillas, la recolección de objetos, la obligación de terminar la partida en un único intento y el uso de generación procesal o procedural para crear los niveles. Sobre este juego, se puede decir que “se le considera el ancestro de los juegos Roguelike, los cuales son unos de los principales géneros de juegos que usan contenido generado proceduralmente (PCG) en videojuegos modernos para crear una experiencia ‘sin fin’” [1, p. 21].

Para este trabajo se considerará como roguelike un juego que tenga muerte permanente, movimiento por casillas, turnos y un entorno generado proceduralmente.

Los Roguelike son conocidos, además de por su mundo aleatorio, por su tremenda dificultad, impredecibilidad, muerte permanente del personaje y la gran cantidad de métodos para causar dicha muerte [2]. A pesar de que sus componentes se encuentran bien definidos, en los últimos años con el surgimiento de títulos como “Spelunky”, “The Binding of Isaac” o “Crypt of the Necrodancer” la fórmula ha cambiado tanto que muchos elementos que caracterizaban al género se han ido dejando de lado y como consecuencia hoy en día pocos títulos nuevos satisfacen a ese sector.

La saga que mejor representa al género clásico en la actualidad es *Mystery Dungeon*, de Chunsoft. Ha sabido adaptar varios conceptos antiguos a los tiempos modernos con buenos resultados. Sin embargo, su último lanzamiento original sin contar remakes fue *Etrian Mystery Dungeon 2* en el año 2017, y actualmente se desconoce si hay otro título en desarrollo. Esto significa que actualmente no hay mucha oferta en cuanto a esta clase de juegos.

Cabe destacar que uno de los aspectos más criticados de “Mystery Dungeon” es la dificultad. Algunos títulos como *Shiren the Wanderer* siguen los parámetros de dificultad clásicos pudiendo resultar frustrantes para el público general. Otros, como *Pokémon Mystery Dungeon: Rescue Team*, resultan demasiado sencillos para los más experimentados. La teoría respecto al reto dice que un videojuego no debe ser ni tan fácil que resulte aburrido ni tan difícil como para frustrar demasiado al jugador [3], y aunque hay gente que precisamente busca un gran reto, cabe la posibilidad de que se puedan cubrir las necesidades tanto de novatos como de veteranos.

Este es un trabajo colaborativo dividido entre la parte gráfica en 3D y la programación, que es en lo que se centra esta propuesta. A lo largo del documento se explicarán las decisiones que se han tomado en cuanto a la organización del proyecto y qué elementos se implementaron para la jugabilidad y el mundo del juego.

1.1. Objetivos

Hacer un videojuego desde cero requiere de muchos meses de planeación y trabajo. Debido al tiempo limitado para la entrega del TFG, se ha decidido crear una pequeña demo que muestre en lo que podría convertirse en caso de ser continuado en un futuro.

Hay varios puntos de calidad de vida que se pueden mejorar respecto a los roguelikes clásicos para brindar una mejor experiencia al jugador, como la personalización de los controles, el diseño y disposición de ciertos menús, el desbalance de objetos y movimientos, etc. Sin embargo hay un punto al que no se le ha prestado la atención merecida: la dificultad.

El género se ha caracterizado siempre por su elevada dificultad, pero con el pasar del tiempo esto ha podido alejar a varios potenciales jugadores. Sin embargo, el enfoque dado por la saga *Mystery Dungeon* parece insuficiente o incluso contraproducente, pues para acceder a la mayor dificultad hay que completar primero la historia principal. Hasta llegar a ese punto, el resto de las mazmorras pueden resultar demasiado sencillas para los más habilidosos, siendo ellos quienes se alejan en este caso.

Si bien se suele preferir ser cauteloso y enfocarse en un solo grupo para no quedar en un punto muerto, la solución propuesta en este proyecto tiene potencial para dejar a ambas partes satisfechas. Se trata de un sistema de clasificación del jugador: dependiendo del desempeño en cada sala, la dificultad subirá o bajará ya sea por la cantidad de objetos, enemigos o demás que se generarán en la siguiente habitación. Al final de cada mazmorra se le mostrará al jugador su clasificación, por ejemplo con un clásico sistema donde la peor nota es la F y la mejor la S. Se premiará el buen desempeño con distintos recursos, incluyendo aquellos que no afecten a la jugabilidad como logros y medallas. A futuro se puede llegar a añadir un ranking online para medir las mejores marcas con otras personas. Por supuesto todo esto son suposiciones, pero resulta lo suficientemente interesante como para ser puesto a prueba.

Para resumir, el objetivo principal es crear un prototipo de un videojuego que contenga algunos elementos básicos de un roguelike (más adelante se explicarán más a detalle): un entorno generado proceduralmente, muerte permanente y movimiento basado en casillas y por turnos. También se tendrá un personaje que puede navegar por el mapa, enemigos que lo sigan y un sistema de combate básico en el que al menos haya un tipo de ataque tanto para el jugador como para los enemigos. El sistema de dificultad dinámica queda relegado como objetivo secundario junto a la inclusión de un personaje que acompañe al protagonista.

1.2. Requisitos

Como bien se ha dicho el desarrollo de un videojuego puede llevar varios meses e incluso años, sobre todo si se trata de una única persona. Por eso se tendrá en cuenta que se trata de un prototipo en el que se muestra el funcionamiento de las mecánicas básicas.

La definición de roguelike no es absoluta, y hay muchas posibles variaciones. Según la definición vista en [4], los puntos clave son que sea un juego por turnos, que use casillas, que haya muerte permanente (que al perder se deba volver al inicio del juego), entornos procedurales, resultados de combate aleatorios (refiriéndose a que el daño causado no es fijo), el uso de un inventario y que sea de un jugador. Para el prototipo se dejarán de lado el daño variable y el inventario por temas de tiempo ya que en un prototipo se puede simplemente usar daños fijos para la demostración y crear ítems que pueden recogerse pero cuyo uso aún no ha sido desarrollado. El último punto no será tomado en cuenta pues en este juego aunque el jugador seguirá controlando a un solo personaje a la vez, el mismo irá acompañado por aliados para añadir elementos estratégicos.

La demo incluirá dos mazmorras que no necesariamente seguirán ese orden en el producto final; es decir, tan solo se crearán para mostrar los movimientos y las acciones básicas del juego. La primera mazmorra será el tutorial, donde se le muestra al jugador poco a poco la variedad de acciones a realizar en los niveles, y cuáles son sus objetivos. La segunda será un ejemplo de cómo será un nivel real, con una dificultad mayor.

Como el centro del proyecto es la jugabilidad, no se tocarán puntos de la historia. Los únicos personajes aliados presentados son el protagonista y su primer compañero. Esto es para mostrar que el jugador controlará a personajes que no comparten el mismo set de movimientos. Además de estos se creará al menos un tipo de enemigo que se comportará de forma autónoma.

En cuanto a la jugabilidad en sí, hay dos instancias de juego: la zona neutral y las mazmorras. La zona neutral normalmente será el pueblo desde el que se puede organizar al equipo, el inventario, hablar con NPCs, comprar objetos, guardar y demás cosas. Le da al jugador un momento de tranquilidad: no hay que atender a ningún contador, preocuparse por enemigos ni gastar objetos. El movimiento es libre; es decir, no se restringe a 8 direcciones. También es desde donde se accede a las mazmorras.

Para la demo únicamente se accederá a las mazmorras, y la zona neutral será reemplazada por menús. En las mazmorras es donde se implementará el sistema de movimiento del equipo y de los enemigos, los ataques, los objetos y el sistema de clasificación.

1.3. Metodología

Tras plantear la idea del proyecto, el siguiente paso fue especificar su alcance. Después de eso se hizo un calendario general en el que se tuvieron en cuenta las tareas primordiales: idear la jugabilidad, implementarla y probarla. La mayor parte del tiempo se dedicó a la implementación.

Al tratarse de un proyecto individual no se puede aplicar plenamente la metodología ágil SCRUM. Sin embargo, sí se han llegado a aplicar algunos de sus principios:

- En todo momento lo más importante es el diseño, pues al tener las cosas claras desde un inicio la implementación resulta mucho más sencilla y eficiente.

- En primera instancia se realiza una reunión con el tutor para planificar los siguientes pasos. Tras eso, se escriben las “historias de usuario” en Trello y se dividen en tareas, que deberán ser completadas antes de la siguiente reunión con el tutor.

- Las primeras tareas son sobre diseño. Se apunta todo lo necesario para que una tarea esté completa, y solo después de tenerlo claro empieza la implementación.

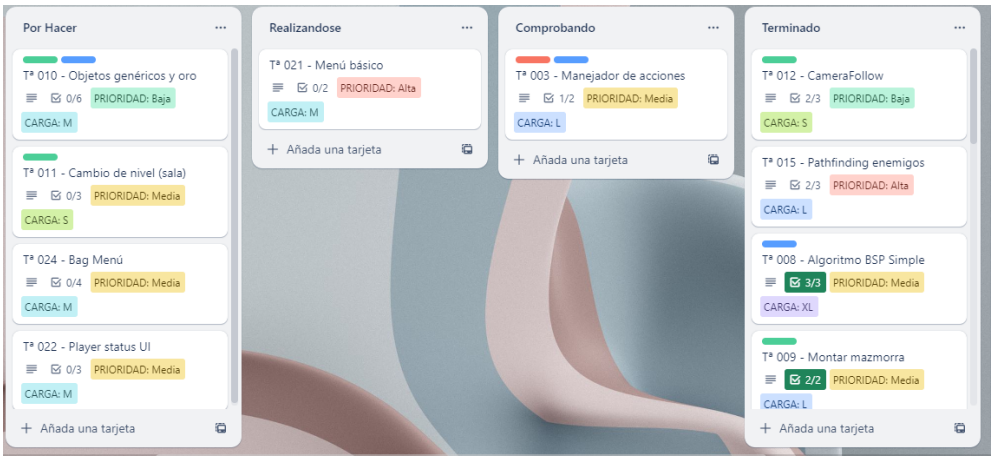

Figura 1: tablero de Trello mostrando las distintas fases de cada tarea

Para organizar las tareas se recurre al tablero de Trello. Cuando se crea una tarea esta se añade a la sección de “Por Hacer”. Cuando se decide llevarla a cabo se pasa a “Realizándose”, donde no pueden haber más de dos tareas a la vez. Una vez parece haberse terminado se pasa a “Comprobando” en caso de que se necesiten más pruebas, y finalmente se pasa a “Terminado”. Tras esto se añaden los cambios al repositorio de git para actualizarlo.

1.4. Etapas

Tras enunciar los objetivos, la metodología y comentar los problemas encontrados a lo largo del proyecto, se presentará el estado del arte donde se explicará qué es la generación procedural con sus ventajas y desventajas. Seguidamente se hace un repaso a los elementos que caracterizan a un juego roguelike en la actualidad junto a algunos ejemplos. Se hará lo mismo con los roguelikes clásicos, centrándose en la saga *Mystery Dungeon* y cómo adaptó conceptos antiguos a la modernidad. Para cerrar este apartado se repasan las tecnologías usadas para el desarrollo del proyecto.

En cuanto a la propuesta, se muestra cómo se desarrolló el proyecto, comenzando con las fases de diseño seguido de la fase de implementación.

Por último se exponen las conclusiones, se explica si se ha alcanzado el objetivo principal del proyecto y se proponen cambios a futuro en caso de seguir con el desarrollo.

1.5. Problemas

Durante el desarrollo se encontraron varios problemas. A continuación se detallan en profundidad y se explica qué se hizo para solventarlos.

- Primera implementación del tablero: para el primer intento de implementación se consultó un tutorial de la documentación de Unity titulado "*2D Roguelike*". Sin embargo, se trataba de un tutorial muy antiguo; la fecha de creación indica que es de 2015. Se ignoraba completamente el concepto de "tablero lógico", por lo que para realizar las comprobaciones de dónde se encuentran otros elementos en el mapa se utilizaban constantemente *raycast*. Los *raycast* son una función de Unity que proyecta un rayo en la dirección indicada y devuelve información sobre el primer objeto que toque. Resultan muy útiles, pero realizar tantas llamadas a este elemento podría resultar contraproducente.
  El principal motivo para su descarte fue, sin embargo, que esa metodología resultaba muy poco organizada. Emplear un tablero lógico como el que se usó en otros proyectos de la carrera como en "polis y cacos" del proyecto de "algorítmica y matemática para videojuegos" de segundo año o el proyecto de "othello" de "inteligencia artificial" de tercer año parecía tener mucho más sentido.

- Dibujar tiles gráficamente: una vez creado el tablero lógico se deben dibujar los tiles de forma gráfica. Se estimó la dificultad de esta tarea de forma errónea, pues acabó consumiendo más tiempo del esperado.
  El principal problema fue el desconocimiento de los gráficos necesarios para los bordes. Al principio parecía que con un gráfico para representar el suelo, uno para representar un muro y ocho para representar las ocho direcciones (horizontales, verticales, diagonales) a las que puede apuntar un borde (lo que se encuentra entre una casilla de suelo y una de muro) sería suficiente. Sin embargo, no se tuvo en cuenta que faltaban representaciones para esquinas interiores, bordes de una única casilla de tamaño, bordes que unen bordes unitarios a bordes normales…
  Tras consultar otros juegos como ejemplo, se descubrió que a los nueve tiles originales se le debían sumar al menos otros veinte.

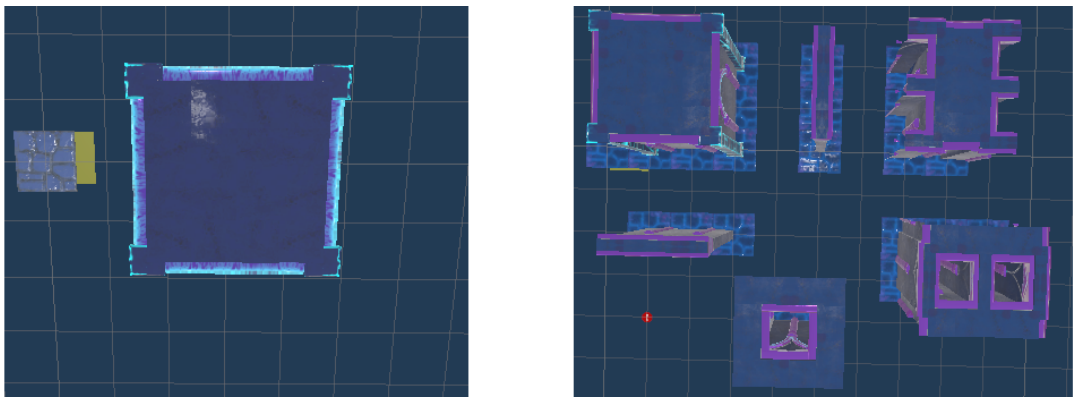

Figura 2: gráficos estimados al principio (izquierda) y gráficos necesarios (derecha)

También se modificó la función para optimizarla más al tener no tener en cuenta tiles que ya hubieran sido cambiados. Se logró reducir el tiempo de generación en casi un segundo.

- Sistema de turno: al ser el primer acercamiento a una jugabilidad como esta, no estaba del todo claro cuál era el camino a seguir. Se plantearon varias formas de implementación hasta que se dio con una que parecía organizada e intuitiva.
  Se emplearon las “Action” de Unity. Se trata de una forma de indicar a distintos scripts de que se ha realizado algún cambio sin que estos deban referenciar al script original. Este sistema se aprovechó no solo para el turno, sino que para muchas otras instancias como por ejemplo indicar a los elementos de interfaz cuándo deben actualizarse.
  En cuanto a la lógica de cómo funciona un turno, se tomó inspiración de su funcionamiento en *Pokémon Mystery Dungeon: Red Rescue Team* gracias a la investigación realizada en [5] y algo de ingeniería inversa.

2. Estado del arte

En este capítulo se revisará el estado actual de los puntos más relevantes para el proyecto, desde la definición de generación procedural hasta el género del juego a desarrollar. Se nombran también las tecnologías empleadas y la justificación de su elección.

2.1. Generación procedural

La generación procedural se compone de una serie de algoritmos usados para crear o colocar varios tipos de elementos de forma automática y siempre distinta. En la actualidad tienen distintos usos, siendo uno de los más atractivos su utilidad a la hora de crear elementos gráficos [1]. En el caso de un roguelike, su uso principal es para crear el entorno de juego, es decir, la forma y diseño de las mazmorras.

Es un acercamiento al diseño de niveles que requiere de mucha planeación antes de su implementación, pues de no haber una preparación correcta el resultado puede ser repetitivo o aburrido para el jugador. Además, de forma inherente se limitan las capacidades creativas en cuanto al aspecto del entorno.

Sin embargo, su uso reduce altamente el coste de diseño y también permite que el jugador tenga una experiencia única cada vez que vuelve a jugar [6]. Esto conlleva a que se ahorre tiempo y dinero en el desarrollo del proyecto, además de que se acentúa el factor de rejugabilidad.

Por supuesto, no todos los elementos se colocan de forma totalmente aleatoria; se trata de una aleatoriedad controlada. La generación procedural en general no se trata de caos, sino que los resultados deben tener una intención y no ser algo que pase porque no se tiene control sobre el proceso [7]. A la hora de implementar los algoritmos, es el desarrollador el que debe decidir el tamaño y hasta cierto punto forma de los niveles, así como qué tipo de objetos aparecerán, su frecuencia, eventos que pueden o no activarse, etc.

2.2. Rogue-likes modernos

Al igual que otros géneros de videojuegos, el roguelike ha ido evolucionando en varias direcciones. Tanta ha sido la ramificación que, a pesar de realizarse eventos como la International Roguelike Development Conference (IRDC) de Berlin en 2008 donde se discutió qué elementos hacen que un roguelike sea considerado como tal, no puede afirmarse que un juego no pertenezca a este género sólo por carecer de alguna de esas características [8].

A continuación se discuten los puntos que por lo general se siguen manteniendo, desde la ambientación de las historias hasta elementos clave de jugabilidad, y finalmente se nombran varios títulos que ejemplifican lo expuesto.

2.2.1. Mazmorras y exploración

Rogue se vio influenciado por obras de la década de 1970 como *Colossal Cave Adventure,* un juego de aventura con elementos ficticios con extensas descripciones, y *Dungeons and Dragons*, un juego de rol de mesa que embarca a los jugadores en aventuras fantásticas a través de mazmorras llenas de monstruos y tesoros (Andre Alves Garzia, 2020).

Debido a esto la ambientación típica es un entorno de fantasía al que se le suele denominar mazmorra. Puede tratarse de una gran habitación o varias más pequeñas que hay que atravesar hasta llegar a la salida que lleve hacia el siguiente escenario. Se trata, pues, de la forma en la que se organizan los niveles. Por norma general hay varias mazmorras con temáticas distintas, y una vez se completa una ya no se puede volver a visitar.

A pesar de que el jugador puede elegir buscar directamente la salida para acabar el juego lo más rápido posible, la gracia de las mazmorras es explorarlas. Incluso así, muchas veces la exploración será casi obligatoria para adquirir objetos y subir de nivel al personaje, para de esta forma volverse más fuerte y poder acceder a niveles más complicados.

Cada juego la maneja a su manera, algunos dan la posibilidad de poder encontrarse con los mejores objetos del juego apenas en los primeros niveles y otros prefieren poner un límite de la calidad del ítem en relación a la dificultad del nivel.

Otro recurso suele ser el oro/dinero, que servirá para comprar todo tipo de cosas en las tiendas, haciendo que la recolección de buenos objetos no sea puramente cosa del azar.

2.2.2. Administración de recursos y sanciones

Los objetos juegan un papel crucial para poder superar los distintos obstáculos, por lo que saber usarlos es primordial.

Cuanto más se explore una mazmorra más recursos se utilizarán, pero también es más probable encontrar objetos valiosos. Esto crea un balance en el que el jugador es quien decide si toma los caminos más complicados a cambio de obtener los mejores ítems.

El jugador tiene un espacio limitado en el inventario, es decir, no puede llevar todos los objetos que quiera a la vez. Debe planear de antemano cuáles llevará y cuándo usará cada uno, pues pueden haber recursos que sean muy limitados o incluso únicos y malgastarlos sería catastrófico.

De todas formas ser muy celoso con los objetos puede ser contraproducente, pues el castigo por perder es la muerte permanente que, por supuesto, incluye la pérdida de todo lo obtenido durante la partida. Este punto es tan característico para el género como la generación procedural.

En estos juegos, al perder una partida no vuelves al último punto de guardado o se te resta una única vida del total; al perder una sola vez debes regresar al inicio del juego. Es decir, perderás todo el progreso que hayas conseguido; armas, objetos, niveles, etc. Comienzas nuevamente desde el primer nivel y sin tener nada.

La penalización es muy dura y obliga a completar el juego por completo de una sola vez, aunque en realidad se espera que el jugador pierda varias veces en el camino para ir mejorando su entendimiento del juego y sus habilidades.

2.2.3. Subida de nivel y combate

Al inspirarse en juegos RPG de la época, se incluye el sistema de niveles para los personajes. En la mayoría de los juegos de rol, las habilidades de los personajes aumentan: se vuelven más poderosos, adquieren mejores armas, técnicas y demás. La idea es simular el crecimiento de los personajes, y que así de cierta forma aprendan de sus experiencias [9].

Estas ideas se reflejan en el juego cuando un personaje vence enemigos. Su barra de experiencia sube hasta alcanzar la cantidad necesaria para subir de nivel, y con cada nivel se aumentan sus estadísticas de combate e incluso se pueden aprender habilidades nuevas. A medida que se va subiendo de nivel, es necesaria más experiencia para seguir avanzando a comparación del nivel anterior. Es necesario subir de nivel pues cuanto más avanza el juego los enemigos irán siendo cada vez más poderosos.

En cuanto al combate, cada juego lo maneja a su forma: puede ser desde un combate por turnos al estilo de los RPG clásicos hasta un Shoot em' Up donde hay que enfrentarse a muchos enemigos a la vez de forma frenética. Sin embargo, las estadísticas suelen ser las mismas:

- Puntos de vida: cuando este valor llegue a cero, el personaje "morirá". Mientras no se agoten, se pueden restaurar mediante objetos o habilidades. Pueden haber objetos especiales que permitan al personaje "revivir" una vez, es decir, llenar parcial o totalmente la barra de vida cuando esta se vacíe.

- Puntos de ataque y defensa: son los valores que se usan para determinar cuánta vida pierde un personaje en el combate. Suelen haber fórmulas que ponen un valor de daño fijo a cada ataque, y estas estadísticas aumentan o disminuyen ese total de daño. Dependiendo del juego, pueden haber también puntos de ataque y defensa especial, haciendo una distinción entre ataques físicos (realizados con armas o habilidades de lucha cuerpo a cuerpo) y especial (hechizos y otros ataques no frontales).

- Velocidad: según la forma de jugar, esta característica puede referirse a la velocidad de movimiento o a la prioridad al atacar. Es decir, en un combate por turnos, atacará antes quien tenga mayor velocidad o quien use algún movimiento con prioridad.

2.2.4. Ejemplos

The Binding of Isaac [10].

El personaje principal, Isaac, vive con su madre tranquilamente en la colina. Sin embargo, un día su madre escucha una "voz celestial" que le dice que su hijo ha sido corrompido por el pecado. Ella le quita todos sus objetos, sus juguetes y hasta su ropa. Cuando esto no es suficiente para esa voz, la madre se dispone a matarlo. Asustado, Isaac se esconde tras una trampilla que lleva al sótano, donde comenzará una gran travesía llena de todo tipo de monstruos.

Figura 3: Menú principal de The Binding of Isaac (2011)

Tal y como indica la sinopsis la ambientación del juego se inspira en la religión cristiana y, por lo tanto, se pueden encontrar objetos como "La Biblia" o "Clavo", y personajes desbloqueables como "Judas", "Eva", "Sansón", entre otros. También se van dejando pequeñas referencias a la verdadera historia del juego, la cual es bastante oscura, en distintas partes del mismo. Como consecuencia, la estética puede resultar algo grotesca e incluso aterradora hasta cierto punto.

Figura 4: uno de los primeros niveles en The Binding of Isaac, 2011.

El juego tiene un control libre, es decir, no sigue una cuadrícula y ocurre en tiempo real. El jugador puede lanzar “lágrimas” a modo de proyectil para acabar con los enemigos. Las mejoras aumentan la cantidad de proyectiles, su forma y su fuerza. La aleatoriedad reside en los enemigos y trampas de cada sala así como la cantidad y organización de estas mismas. Sin embargo, la forma de las salas es siempre rectangular y ocupa toda la pantalla.

Spelunky [11]

El jugador controla a un aventurero que se adentra en una cueva en la que descubrirá cientos de tesoros y monstruos de todo tipo a vencer.

A diferencia del ejemplo anterior, la historia no es un punto importante. Sin embargo, se puede notar una clara inspiración en franquicias como *Indiana Jones* para el aspecto del protagonista y su arma básica, el látigo, así como guiños a civilizaciones antiguas como la Azteca en la estética de algunos tesoros.

Figura 5: primer nivel de Spelunky

En cuanto a la jugabilidad se puede considerar un “plataformas 2D”, pues el jugador se mueve, esquiva y derrota a los enemigos a tiempo real mientras recorre un escenario lleno de plataformas de distintos tipos. Son estas plataformas las que se colocan de forma procedural, así como los objetos y enemigos que el jugador pueda encontrarse. Nunca se sale del modo de juego principal, pues las tiendas se encuentran en los propios niveles.

Vampire Survivors [12]

La descripción del juego dice que en el año 2021, en Italia, vivía una persona maligna con el nombre de Bisconte Draculó. Su magia malvada creó un mundo lleno de hambruna y sufrimiento, y ahora está en manos de la familia Belpaese acabar con su reino de terror y volver a traer la comida a la mesa.

En este juego el jugador únicamente debe moverse por el escenario, que es únicamente un amplio terreno. El personaje realizará ataques de forma automática. Al subir de nivel se aprenden nuevas habilidades que se ejecutan también sin la necesidad de que el jugador los active, y cada vez son más poderosos. Por su parte, irán saliendo enemigos más fuertes y, sobre todo, en mayor cantidad. El objetivo es sobrevivir el mayor tiempo posible, siendo que a la media hora de partida saldrá un enemigo invencible que llevará a una muerte segura por parte del jugador para intentarlo nuevamente. Con los recursos obtenidos se pueden desbloquear personajes, y si se consigue cierto avance en el juego, se podrá enfrentar a ese enemigo siendo este ahora vulnerable.

Figura 6: gameplay de Vampire Survivors (2022) a los 20 minutos de partida

El juego se apoya mucho en el caos como factor llamativo, y como su jugabilidad consiste en únicamente mover al personaje, resulta muy sencillo engancharse para intentar llegar cada vez más lejos.

2.3. Rogue-like clásicos

Como se comentó anteriormente, hay muchos aspectos que se dejaron de tener en cuenta en los juegos más modernos. A continuación se explicarán los elementos clásicos que ya no suelen emplearse, y finalmente se expondrán ejemplos de juegos más actuales que siguen implementando las fórmulas clásicas.

2.3.1. Movimiento basado en casillas y uso de tilesets

Los tilemaps son escenarios compuestos por tilesets. Los tilesets son una serie de elementos gráficos que se organizan en forma de cuadrícula. Para su uso, el escenario del juego se divide en casillas y cada elemento del tileset corresponde a una.

Se usaron en un principio para ahorrar espacio y memoria en juegos antiguos. En lugar de tener escenarios gigantes para el nivel entero, los gráficos serían ensamblados juntando piezas más pequeñas (Andre Alves Garzia, 2020).

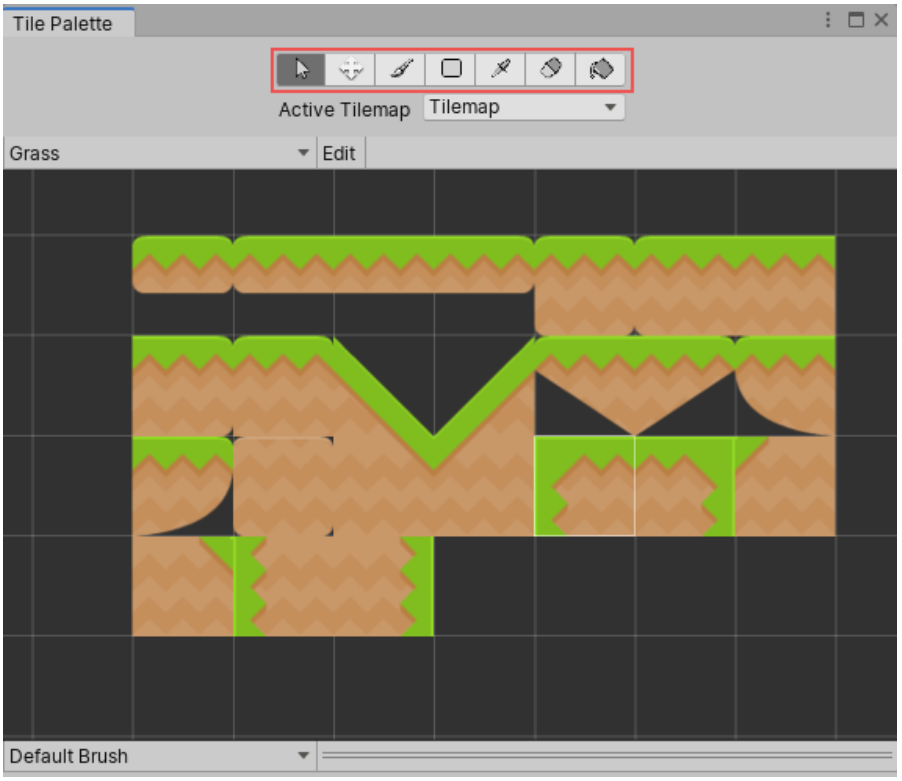

Figura 7: tilemap de ejemplo en la documentación de tilemaps 2D en Unity

En roguelikes clásicos se emplean tilemaps porque el entorno de juego es casi como un tablero, y los personajes son fichas. Además, al generar escenarios de forma procedural, es mucho más sencillo crear elementos coherentes si todos los gráficos ocupan el mismo espacio, en este caso una o varias casillas enteras.

Los personajes solo se pueden mover en 8 direcciones (de forma vertical, horizontal y diagonal) y el tiempo de juego se basa en el movimiento: las acciones del juego no se ejecutan hasta que el jugador ha decidido su siguiente jugada. Una vez que el jugador decide una acción, todos los personajes de la sala hacen lo propio siguiendo un orden: hasta que un personaje no ha completado su movimiento, el resto no actuará. Esto hace que el jugador deba pensar bien sus movimientos antes de actuar, teniendo en cuenta no solo sus acciones sino también las de sus enemigos.

2.3.2. Mazmorras

Las mazmorras se componen por pisos, que a su vez se componen de salas o habitaciones. Al pasar a un nuevo piso el jugador es colocado en una sala al azar. Las salas son los espacios por los que pasa el jugador, buscando la salida del piso que usualmente se representa como unas escaleras. Las habitaciones están interconectadas por pasillos o bien son únicas pero con una gran extensión. Cuando se llega al final del piso se pasa al siguiente, y se repite el proceso hasta salir de la mazmorra.

Las primeras mazmorras son más cortas y tienen pisos más pequeños para que el jugador se vaya acostumbrando a las mecánicas. Algunas pueden tener incluso sus propias mecánicas, como iluminación limitada, distintos tipos de trampas e incluso pequeños puzzles secundarios.

En *Rogue* la sala entera se veía por pantalla pues los gráficos eran simplemente texto ASCII. Con el pasar del tiempo, la salas enteras dejaron de caber por pantalla al utilizar gráficos más complejos, por lo que el jugador tiene a disposición un minimapa que le indica las habitaciones por las que ha pasado y qué hay en ellas. De esta forma se puede recorrer el piso sin el riesgo de volver a pasar por una zona previamente explorada sin darse cuenta, evitando así que el jugador se pierda.

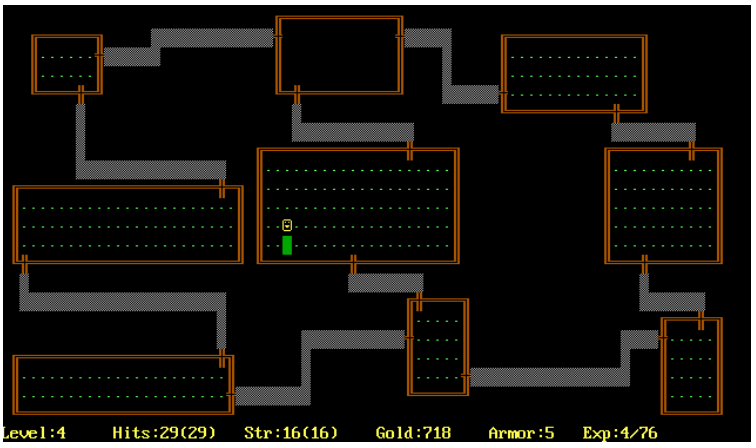

Figura 8: entorno de juego de Rogue (1980)

2.3.3 Ejemplos

La saga *Mystery Dungeon*, desarrollada por *Chunsoft,* existe desde la década de 1990 y ha sabido adaptar la fórmula clásica a tiempos actuales. La saga principal es *Shiren the Wanderer* pero la empresa se ha caracterizado por realizar crossovers con otras franquicias de videojuegos, siendo los más exitosos los títulos de *Pokémon Mystery Dungeon*. A continuación se explicarán las características de los videojuegos S*hiren the Wanderer: The Tower of Fortune and the Dice of Fate* y *Pokémon Mystery Dungeon: Rescue Team DX*, ambos remakes de juegos más antiguos de la compañía.

Shiren the Wanderer: The Tower of Fortune and the Dice of Fate [13]

Según la sinopsis Reeva, el dios del destino, domina el pasado, presente y futuro. Se dice que habita en la *Torre de la fortuna* desde donde domina a toda la humanidad. El protagonista, Shiren, se dirige a conquistar la misteriosa torre junto a sus compañeros.

El apartado de la historia queda mejor cubierto por los personajes secundarios puesto que Shiren es un protagonista mudo, es decir, no tiene diálogos. De esta forma el jugador se ve mejor representado por Shiren y sigue pudiendo disfrutar de la historia del mundo en el que se adentra.

La ambientación es completamente fantástica, pero bebe bastante de elementos clásicos japoneses. Por ejemplo, el propio Shiren es un *Rōnin*, un espadachín del Japón feudal que no servía a ningún señor. Debido a esto la mayoría de gráficos del juego tienden a emular elementos y localizaciones naturales, como bosques o cuevas.

Figura 9: mazmorra de Shiren the Wanderer: The Tower of Fortune and the Dice of Fate (2020)

La cámara, que en *Rogue* permanecía estática debido a limitaciones técnicas, ahora sigue al jugador para dar una mayor inmersión. Por ello se debe usar el minimapa, que se puede ver en la figura 9 en la parte superior izquierda. Los puntos rojos indican enemigos y los azules objetos.

También se añaden compañeros de equipo que cuentan con acciones distintas al protagonista, como ataques con distintos patrones, para añadir estrategia a la hora de plantear una exploración. El control sobre estos es limitado pues se mueven de forma automática y el jugador solo puede darles indicaciones como “ataca a los enemigos”, “escapa”, etc.

Aunque hay puntos de guardado, si pierdes en este título volverás al nivel 1 y se te quitarán todos los recursos que llevases encima. Es importante guardar los objetos más valiosos en caso de adentrarse a una exploración peligrosa.

Pokémon Mystery Dungeon: Rescue Team DX [14]

En este juego el jugador se adentra en el mundo de los Pokémon, unas criaturas con poderes singulares, y se convierte en uno. Junto a otros compañeros deberá salvar ese mundo y descubrir cómo llegó allí en un primer lugar.

La historia, aunque no muy compleja, involucra al jugador de forma más directa que el título anterior. En esta saga el sentimiento de compañerismo es algo principal, y sus mecánicas apoyan a enfatizar ese hecho: siempre se tendrán como mínimo a dos personajes en el equipo, con un máximo de tres integrantes usualmente y hasta ocho en circunstancias especiales. Además el jugador puede elegir en qué Pokémon convertirse al empezar la partida y quién será su primer compañero, por lo que hay más personalización para cada uno.

Se mantienen muchos cambios efectuados en títulos anteriores, como la cámara anteriormente mencionada o el minimapa.

Al igual que la franquicia principal de *Pokémon*, en este juego se pueden capturar cientos de monstruos distintos para usarlos en la aventura. Todos tienen estadísticas, tipos y habilidades distintas, por lo que el jugador puede armarse el equipo como mejor le parezca.

Ser derrotado en un territorio no es tan grave como en *Shiren the Wanderer*, pues solo se perderán la mitad de los objetos que se lleve encima y el nivel de los personajes quedará intacto. Además, en los territorios más largos hay puntos de guardado desde los que se podrá volver a empezar en caso de ser derrotado más adelante.

2.4. Tecnologías

En este apartado se discuten las tecnologías empleadas en el desarrollo del proyecto. Esto es, qué se ha usado para la planeación, diseño y creación del juego, así como otras herramientas para su organización y control.

2.4.1. Motor de videojuegos

Según expone Luis Jesús Arce:

> Un Motor de Videojuegos (en inglés Game Engine) es una aplicación de software que ofrece todas las herramientas necesarias para el diseño y desarrollo completo de un videojuego, disponiendo de un motor de renderizado para gráficos 2D y 3D, detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y mucho más. [15, p. 9].

Así pues, elegir un motor adecuado será el primer paso para desarrollar un videojuego. Se han analizado las ventajas y desventajas de tres de los motores más populares a día de hoy: Godot, Unreal y Unity.

- Godot: algunos de los puntos a favor que enumera Maithiuli Dhule [16] son sus herramientas gráficas, el soporte de varios lenguajes de programación, su interfaz agradable a la vista así como sencilla de entender, el poco espacio que usa en el PC y el hecho de que sea de código abierto, que entre otras cosas conlleva a que no exista ningún coste por su uso ni por la venta de juegos hechos con este motor.

  Como puntos en contra encontramos que, si bien hay una activa y creciente comunidad, sigue habiendo muchísima menos documentación a comparación de otras opciones más populares. Además Godot es mucho más potente para el desarrollo 2D, por lo que no es la mejor opción para el desarrollo de este proyecto.

- Unreal Engine: la investigación realizada en [17] señala que Unreal cuenta con el motor de físicas actualmente, así como un poder gráfico superior a la competencia, con capacidad de reflejar texturas a tiempo real, un gran sistema de raycasting y transparencias, entre otros. Para uso personal es completamente gratuito. Se hace uso de C++ y *Blueprint Visual Scripting* para programar.

  Sus desventajas son que a partir de que el juego genere $3,000 USD de ingresos Epic Games, los desarrolladores de Unreal, se quedarán el 5% de lo generado. Al ser un programa tan potente, cualquier proyecto pesará más de lo normal. Está más bien enfocado en grandes producciones a niveles AAA, es decir, juegos hechos por

grandes compañías. Debido a esto se ha considerado que tampoco es la mejor opción para el proyecto.

- Unity: según la investigación de Jesús Isaac Lerma Lizarraga en 2017, Unity es un motor multiplataforma para 2D y 3D. Es compatible con casi todas las plataformas de diseño 2D y 3D como Blender, Maya, Adobe Photoshop, etc. Es usado tanto por desarrolladores independientes como por algunas compañías. Tiene una versión gratuita que cuenta con todas las características necesarias para el desarrollo. Además es uno de los motores con mayor documentación tanto oficial como por su comunidad. Su lenguaje de programación es C#, por lo que resulta en principio menos accesible que otras opciones.

  Se empieza a pagar a partir de que el usuario o empresa que lo use haya facturado $100,000 USD el año anterior, por lo que a efectos de este proyecto, el desarrollo en Unity es gratuito. Por último, se trata de una herramienta que ya ha sido estudiada en la carrera, concretamente en el proyecto de la asignatura *Proyecto de aplicaciones multimedia y videojuegos* de segundo año.

  Teniendo todo lo anterior en cuenta, Unity es el motor escogido para este trabajo.

2.4.2. Herramientas para la organización

Se emplearán ciertas herramientas que ayuden a agilizar y organizar los procesos de forma ordenada. A continuación se enumeran las principales.

- Visual Studio Community 2019: se usará para escribir el código del juego. Gracias a la licencia proporcionada por la UPV se podrá emplear sin ningún coste. Es fácilmente integrable a Unity, por lo que sus funciones de autocompletado y detección de errores se adaptan perfectamente a dicho motor.

- Trello: se trata de una herramienta online que permite organizar la planificación del proyecto mediante el uso de tarjetas que representan las distintas tareas a realizar. Es una forma de digitalizar las ideas y mantener cierto orden. Esta ha sido la herramienta organizativa por excelencia en los proyectos realizados a lo largo del curso.

- GitHub Desktop: Git es un sistema de control de versiones que sirve para llevar un registro de los cambios que se realizan en el proyecto. Se debe manejar a través de líneas de comandos. Es muy útil para trabajar en equipo, pues cada uno puede ocuparse de su parte y más tarde pueden juntarse todas. En este caso se ha optado por usar GitHub Desktop, que es una aplicación gratuita que sirve el mismo propósito pero puede manejarse de forma gráfica, resultando así más sencilla de manejar.

2.4.3. Otros

- Mixamo: se trata de una página web en la que se pueden subir modelos de personajes bípedos para que mediante algoritmos internos se haga un rigging automático. Esto se refiere a que crea una especie de esqueleto para que se pueda animar fácilmente. Además, la propia página proporciona varias animaciones pre hechas que pueden ser insertadas en un nuevo modelo sin muchos problemas.
  Es una herramienta totalmente gratuita que permite aliviar la carga de trabajo del animador y está pensada para ser usada en videojuegos.

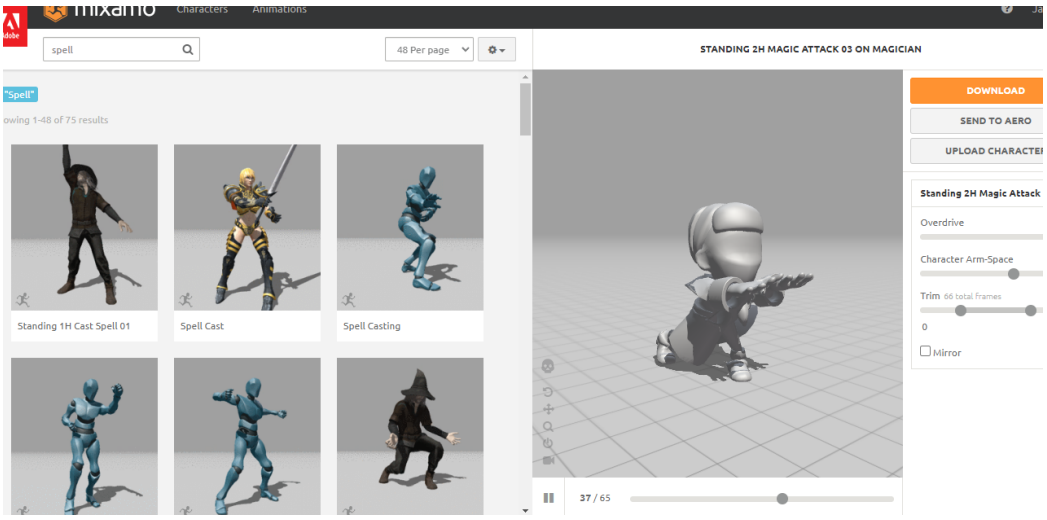

Figura 10: web de Mixamo con el modelo del personaje de la maga al que se le ha cargado la animación de lanzar un ataque mágico

- Google Scholar: es un buscador en línea que se especializa en trabajos académicos y otro tipo de documentos oficiales. Se ha empleado para sacar fuentes para explicar varios conceptos que se han ido presentando en la introducción del proyecto.

3. Propuesta

Como se mencionó durante la introducción, se propone el desarrollo de un videojuego roguelike clásico inspirado principalmente en la saga *Mystery Dungeon*. El trabajo se enfocará en la creación del prototipo enfatizando el estudio de algoritmos de generación procedural y de búsqueda, y la implementación de algunos cambios como el sistema de dificultad dinámica.

3.1. Diseño

Lo primero siempre es diseñar y planear todo lo necesario para que el juego funcione. Los puntos principales fueron los algoritmos de generación procedural, los objetos y personajes, el combate y los menús.

3.1.1. Algoritmos de generación procedural

El mapa se crea de forma procedural para que ningún nivel sea igual. Hay algoritmos que ayudan a conseguir esto. En una misma mazmorra se usará sólo un tipo de algoritmo para mantener una cohesión. Por ejemplo, si el algoritmo crea salas estructuradas, todo el territorio deberá ser así.

Tras investigar distintos algoritmos para la creación del mapa, hubo tres que parecían poder encajar con el juego. Se mostrarán también su diagrama de flujo y una implementación realizada en Unity de cada uno.

3.1.1.1. Random Walk Algorithm (RWA)

Según se explica en [18], este algoritmo consiste en situar un agente, que es un objeto lógico que actuará por su cuenta siguiendo unos comandos dados, en cualquier lado del mapa. Acto seguido se moverá a una casilla adyacente de forma aleatoria. Desde la nueva casilla se repite el proceso, y así hasta las veces que se le haya indicado. Denominaremos a todo el espacio que el agente haya recorrido como “isla”. Al acabar se añade la isla al espacio de juego, y después se asigna una nueva dirección hacia la cual el agente se moverá en línea recta. Al llegar a la nueva posición se considerará la nueva casilla como el nuevo punto de inicio, desde el cual se repite todo el proceso para crear una nueva isla. Esto se repite tantas veces como se quiera. Hay que tener en cuenta que es posible que dos islas se conecten, creando así una única isla de mayor tamaño.

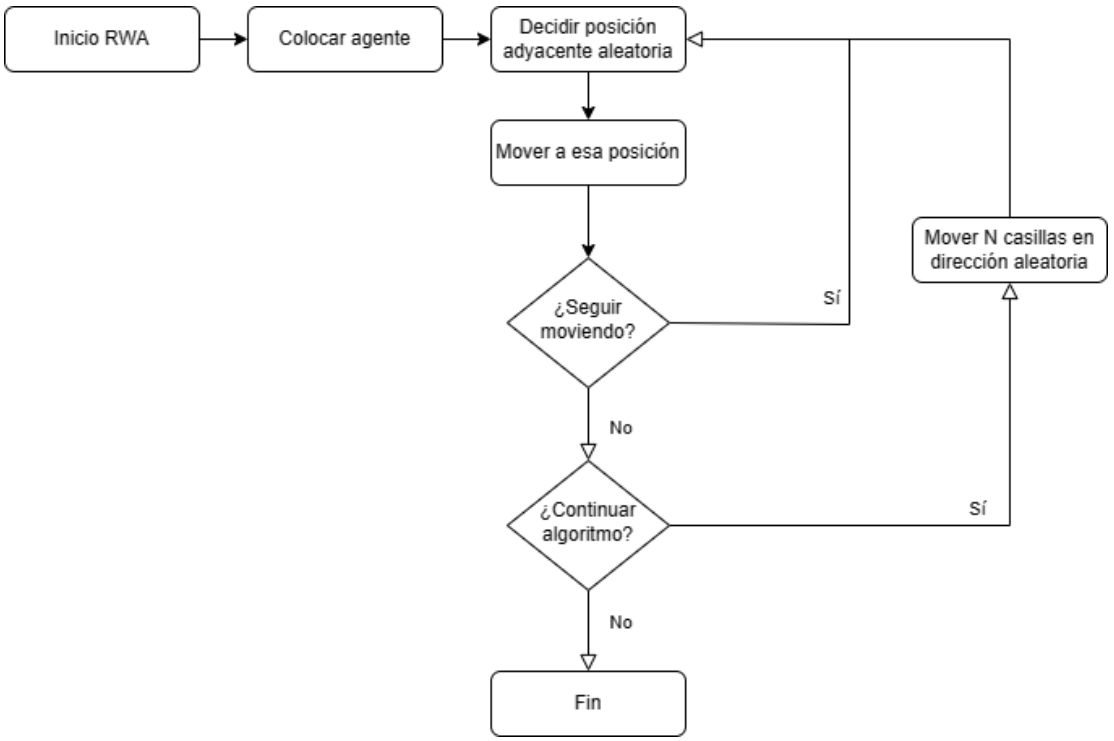

Figura 11: diagrama de flujo del algoritmo RWA

Las variables que el desarrollador puede controlar son el número de iteraciones totales, el número de islas y la cantidad de pasos que dará el agente. El cambio de valores en alguna de estas afectará enormemente al resultado.

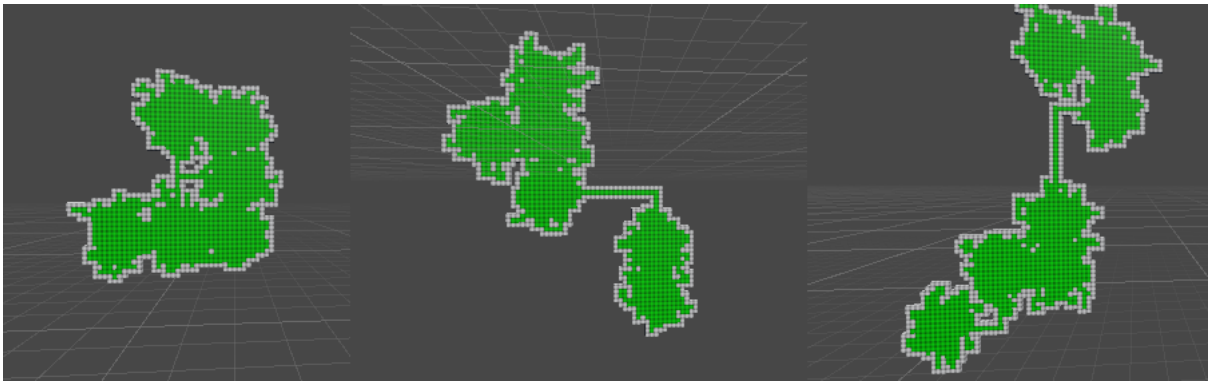

Figura 12: resultados del uso de RWA sin cambiar las variables

Este método crea niveles muy caóticos pues no hay formas predefinidas y es más complicado definir la extensión del tablero. Se ha decidido, por tanto, que finalmente no se utilizará este algoritmo en el juego o, por lo menos, en la demo; debido a la ambientación del juego se prefieren obtener pisos más ordenados, como si se tratase de torres. Aún así el producto final podría incluirlo para secciones de puzzle específicas.

3.1.1.2. Binary Space Partitioning (BSP)

En este algoritmo se trata el espacio como celdas individuales. Desde el inicio se sabe la extensión máxima del terreno, pues tomará forma de rectángulo. La única celda existente al inicio abarca todo el espacio total.

En cada iteración todas las celdas disponibles se partirán por la mitad en un punto aleatorio de las mismas, y cada parte resultante será una celda independiente.El proceso continúa hasta que cada celda llegue a su umbral mínimo de tamaño. Tras eso se conectan con sus celdas adyacentes con filas y columnas que actúan como pasillos.

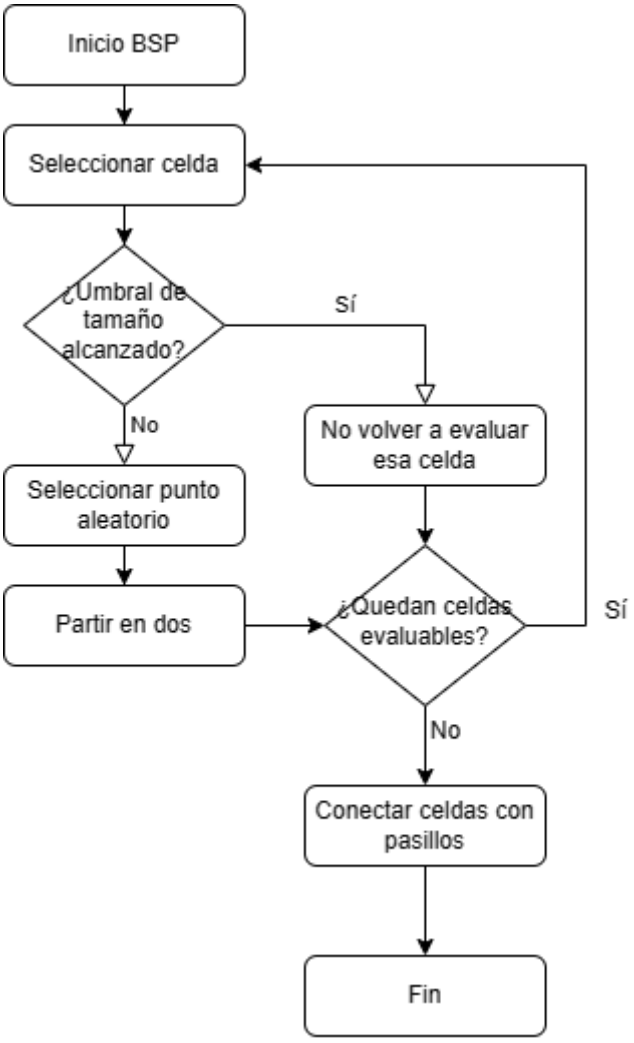

Figura 13: diagrama de flujo de BSP

El desarrollador puede controlar el umbral de cada celda y el tamaño total del terreno. Cambiando un poco los valores se pueden conseguir pocas habitaciones de gran extensión o muchas de espacio reducido.

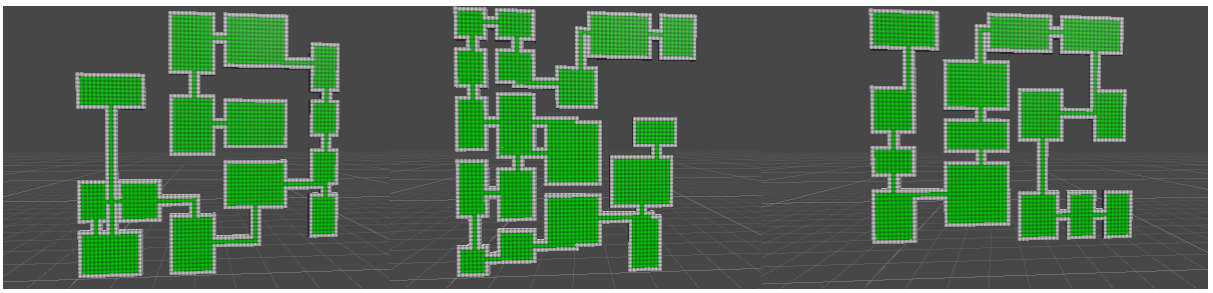

Figura 14: resultados del uso de BSP sin cambiar las variables

Este método crea niveles bien estructurados por lo que el jugador puede saber cómo se distribuirá el nivel aunque no sepa exactamente qué formas y tamaños tienen las habitaciones. Es un algoritmo útil para niveles más avanzados pues se acaban creando muchas habitaciones en no mucho espacio.

3.1.1.3. BSP simplificado

Se trata de un algoritmo basado en BSP pero simplificado de tal forma que el desarrollador puede controlar de mejor forma la cantidad de habitaciones totales. Este algoritmo es usado en la saga *Pokémon Mystery Dungeon* según la información recopilada en [19].

En primera instancia se marca el tamaño total del espacio de juego. También se deben definir la cantidad de cortes verticales y horizontales, y un umbral de tamaño mínimo. Se parte todo el espacio de forma uniforme según los cortes indicados. Por ejemplo, si se indican dos cortes verticales y uno horizontal, el resultado es que el espacio se reparte en un espacio de 3x2. Después se crea una sala dentro de cada celda, con una extensión aleatoria pero respetando el tamaño mínimo indicado. Para finalizar se unen las salas adyacentes con pasillos en puntos aleatorios.

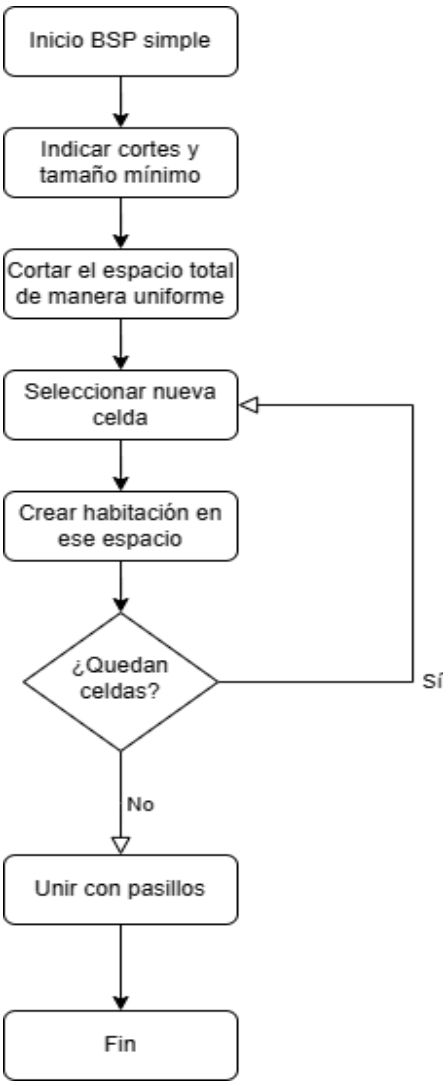

Figura 15: diagrama de flujo de BSP simple

El desarrollador puede controlar el espacio total, la cantidad de habitaciones y el tamaño mínimo de cada una.

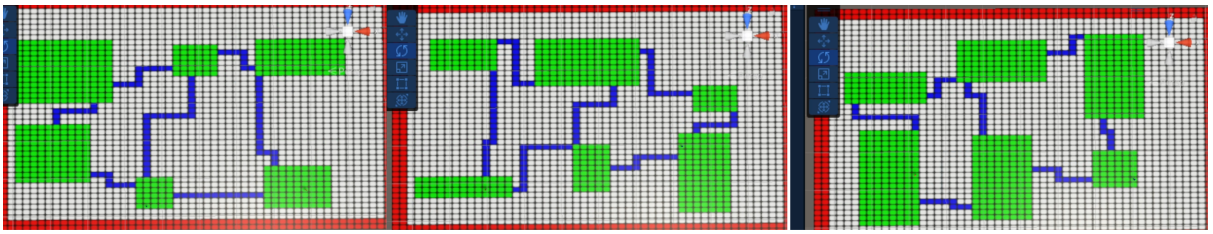

Figura 16: resultados del uso de BSP simple sin cambiar las variables

Como resultado se obtienen terrenos más estructurados, más moldeables a gusto del desarrollador. Es el algoritmo más adecuado para los primeros niveles pues tiene patrones más reconocibles.

3.1.2. Elementos de mazmorra

Son todos aquellos aspectos que se encuentran en una mazmorra. En el juego completo se haría esta distinción con la parte del juego fuera de los territorios, pero para la extensión de la demo esto es esencialmente todo aquel elemento que existe durante el gameplay.

3.1.2.1. Personajes

Refiriéndonos a los personajes, cada uno deberá contar con las siguientes características:

- Tipo: indica si es un enemigo o un aliado.

- Tipo de movimiento: indica cómo se moverá por la mazmorra. Esto va cambiando continuamente. Por ejemplo, al principio puede tener la indicación “explorar por ahí”, pero al ver a un enemigo puede pasar a “buscar enemigo”.

- Objeto equipado: cada personaje puede llevar hasta un objeto encima. De esta forma si un enemigo lleva un ítem este lo soltará al ser derrotado, pero también podrá consumirlo.

- Lista de movimientos: hace referencia a todos sus ataques: ataques que afecten a un área, ataques que afecten en línea recta, habilidades de curación, etc.

- Estadísticas: las estadísticas que se usarán en el combate como ataque, ataque especial, defensa… Se incluye también la cantidad de experiencia que se obtendría al derrotarlo.

- Casilla: referencia a la casilla en la que el personaje se encuentra en este momento.

- Habitación: sala en la que se encuentra la casilla. Es útil para realizar ciertas comprobaciones, como ataques que afecten a toda la habitación.

3.1.2.2. Objetos

Desde algunos que restauren vida hasta otros que causen daño, todos los objetos comparten ciertas características en común:

- Valor: ya sea para dañar, curar o incluso el tiempo durante el que actuará, todo ello se basa en esta variable.

- Cantidad máxima: es la cantidad del mismo objeto que se pueden acumular en un solo compartimento. Esto quiere decir que por cada objeto de “cantidad máxima” igual a uno, cada unidad usará un espacio del inventario. En cambio, si por ejemplo vale 99, se podrán llevar de 1 a 99 utilizando un solo espacio del inventario.

- Precio de venta: la cantidad de monedas que cuesta comprarlo en una tienda.

- Tipo: pueden ser de un solo uso, para hacer daño, para aturdir enemigos, etc.

3.1.2.3. Casillas

Las casillas también tienen ciertos atributos que es necesario tener en cuenta.

- Tipo: puede ser una casilla de suelo donde se pueda caminar, un muro que bloquee el paso o incluso una casilla trampa.

- Posición lógica: es la posición que ocuparía si fuera estrictamente un tablero. Por ejemplo, (1, 1).

- Posición real: es la posición que se ocupa en el mundo del juego. Por ejemplo, si los gráficos de los tiles fueran rectangulares en lugar de cuadrados por cuestión de perspectiva, la posición (1, 1) podría ser realmente (2, 1.4).

- Objeto actual: hace referencia al objeto que se encuentre ocupando la casilla, ya sea un ítem o un personaje.

- Vecinos: una lista con todos sus tiles adyacentes ya sea de forma horizontal, vertical o diagonal. Será necesario conocer a sus vecinos para implementar ciertos algoritmos.

- Habitación: sala en la que se encuentra la casilla. Es útil para realizar ciertas comprobaciones, como ataques que afecten a toda la habitación.

3.1.2.4. Habilidades

Son los ataques que puede realizar un personaje. Todos deben tener estas variables definidas:

- Nombre: nombre de la habilidad. Varios personajes pueden compartir habilidades, y cada una se identificará por su nombre.

- Tipo: qué clase de habilidad es: hace daño, cura, sube o baja estadísticas del personaje, etc.

- Alcance: a qué posiciones afectará: justo a la casilla de enfrente, hasta N casillas de distancia, a toda la habitación, al propio personaje…

- Valor: se refiere a qué tan fuerte es su efecto: cuánto daño hace, cuánta vida recupera, cuánto sube una estadística, etc. Este es el valor base, pero puede resultar mayor según las estadísticas del personaje que la utilice y del receptor.

- Coste: cuánto cuesta utilizar la habilidad. Normalmente se usan “puntos de habilidad”. En *Pokémon Mystery Dungeon* las propias habilidades tienen sus propios puntos de habilidad, por lo que gastar una habilidad al completo no impide usar cualquiera de las otras. Sin embargo, en este proyecto cada personaje tendrá su propio contador de puntos de habilidad o “maná”, lo cual es un acercamiento más próximo a los RPG clásicos. De esta forma se deberá administrar bien para no malgastar dichos puntos.

3.1.2.5. Otros

Hay otro tipo de elementos en las mazmorras, como las escaleras que son el objeto que permite pasar al siguiente nivel o las trampas que dañan al personaje que pase por encima.

También hay que tener en cuenta a la propia mazmorra como un “objeto”, pues hay algunos puntos que deben tener definidos:

- Algoritmo: qué algoritmo se usa para su creación, así como las variables requeridas para este mismo.

- Tamaño: qué extensión tendrá la mazmorra, es decir, cuántos pisos deberán ser superados para completarla.

- Tileset: los elementos gráficos que conformarán el mapa del juego.

- Enemigos: qué tipos de enemigos aparecerán en este territorio, así como su nivel y frecuencia de aparición.

- Objetos: qué tipos de ítems se podrán encontrar mediante la exploración.

3.1.3. Sistema de turnos

El turno no se lleva a cabo hasta que el jugador decida una acción. Las acciones de los personajes se dividen en tres tipos: moverse, usar una habilidad y usar un objeto. Una vez el jugador decide su acción, el resto de los personajes también lo hace de forma automática y se lleva a cabo el turno. Dependiendo de la acción del jugador, los turnos seguirán un orden u otro:

- Si el jugador decidió moverse o quedarse en el sitio:

Ataque a distancia > Movimiento jugador > Movimiento personajes > Habilidades / objetos

- Si el jugador decidió usar una habilidad o un objeto:

Ataque a distancia > Movimiento jugador > Habilidades / objetos > Movimiento personajes

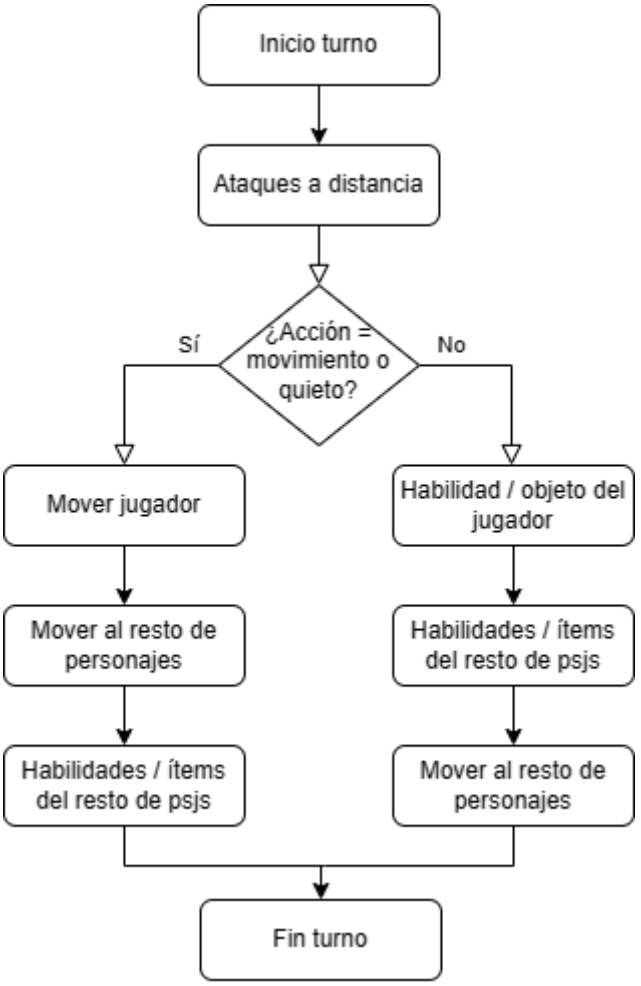

Figura 17: diagrama de funcionamiento de un turno

3.1.4. Menús del juego

Es la forma que tiene el usuario de seleccionar ciertas acciones y opciones respecto a la partida. Los menús son:

- Menú principal: también referido como pantalla de título, es el intermediario entre abrir el juego y empezar la partida.

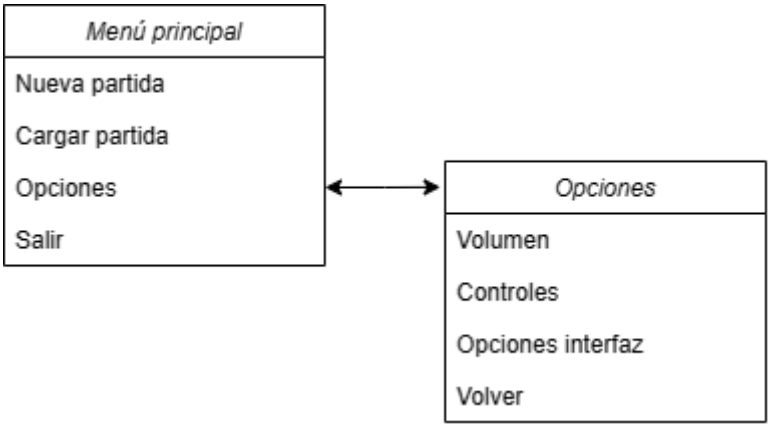

Figura 18: diagrama del menú de la pantalla principal del juego

- Menú de juego: desde aquí se podrán elegir acciones dentro de la mazmorra. Estas son:
    - Habilidades: menú donde se encuentran todas las habilidades (o ataques) disponibles, cada una con su propio nombre y gasto de puntos de habilidad.

    - Bolsa: es el inventario, es decir, donde se encuentran todos los ítems que el jugador lleva consigo en ese momento.

    - Equipo: desde esta ventana se puede consultar información del equipo, como su nivel, estadísticas, ataques, estado actual, comportamiento actual y objetos equipados.

    - Otros: permite acceder al mismo menú de opciones que se mostraba en la pantalla de título, así como abandonar la partida desde ahí.

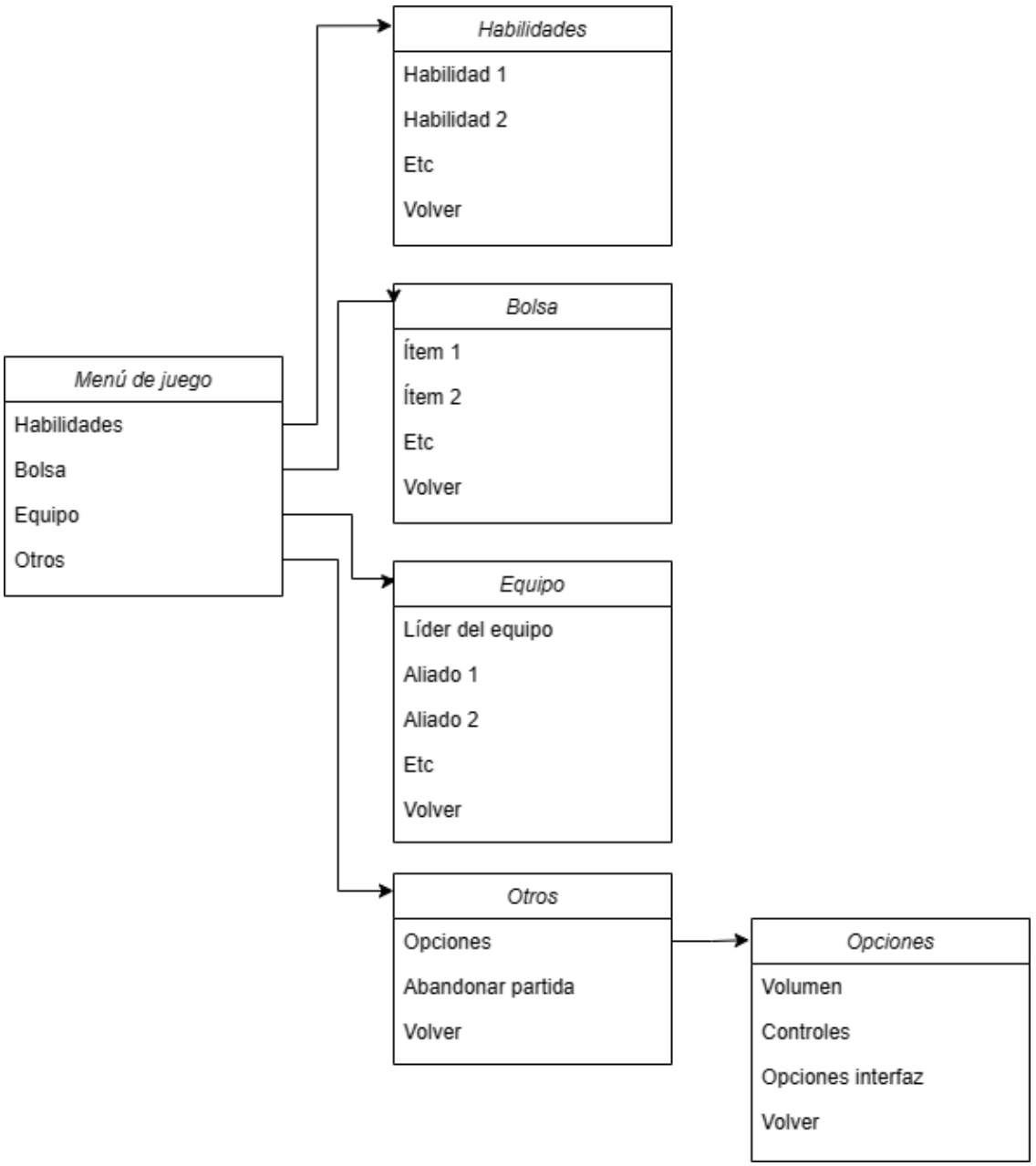

Figura 19: diagrama del menú de juego

3.2. Implementación

En este apartado se discute cómo se han implementado los elementos previamente diseñados en Unity. Se explicará la función de cada script y por qué se han organizado de esta forma.

3.2.1. Clases principales

Las clases que representan a los objetos principales se basaron en los diseños del apartado anterior, implementando así “BoardTile” para las casillas, “BoardManager” para manejar lo que suceda en el juego y “BoardObject” que representa a cualquier cosa que pueda ocupar una casilla. De esta última heredan las clases “BoardCharacter” y “BoardItem”.

3.2.1.1. Tablero

El entorno de juego consiste en un tablero virtual. Los algoritmos de generación procedural están en “LevelCreationAlgorithms” y un script llamado “BoardCreator” se encarga de llamar al anterior pasándole los valores especificados para un territorio en concreto. El resultado se guarda en una lista de BoardTiles que se le pasa luego a BoardManager, que la usará para llevar a cabo los turnos.

```
int roomX = x1 + xOffset;
int roomY = y1 + yOffset;
for (int x = roomX; x < roomX + roomLength; x++)
{
    for (int y = roomY; y < roomY + roomHeight; y++)
    {
        boardList[x, y].SetTileType(TileType.Floor, rooms.Count);
        roomTiles.Add(boardList[x, y]);
    }
}
```

Figura 20: función para crear una única sala usando BSP simple

Durante el proceso de creación del piso se guarda un HashSet, un tipo de lista que se asegura de que haya una única instancia de cada ítem en la lista, que contiene referencias a los tiles que conforman el piso por el que se puede caminar.

”BoardPainter” se encarga de mostrar el gráfico adecuado para cada tile, y usa también la lista de tiles de suelo para adecuar cada tile según su dirección en la función “PaintBorders”. Sucede que los tiles de los bordes, los que están entre el muro y el suelo, usan un asset distinto dependiendo de su dirección: ya sea un lateral derecho, una esquina inferior izquierda, un borde único… Para ese caso se sigue la siguiente lógica:

Identificar bordes > Bordes laterales > Esquinas interiores > Esquinas exteriores > Esquinas unitarias > Uniones entre bordes unitarios y no unitarios

En cada paso se descartan los bordes que ya hayan sido definidos. Como resultado, se realizan muchos menos cálculos que revisando el tablero entero y se tardan menos segundos en cargar todo.

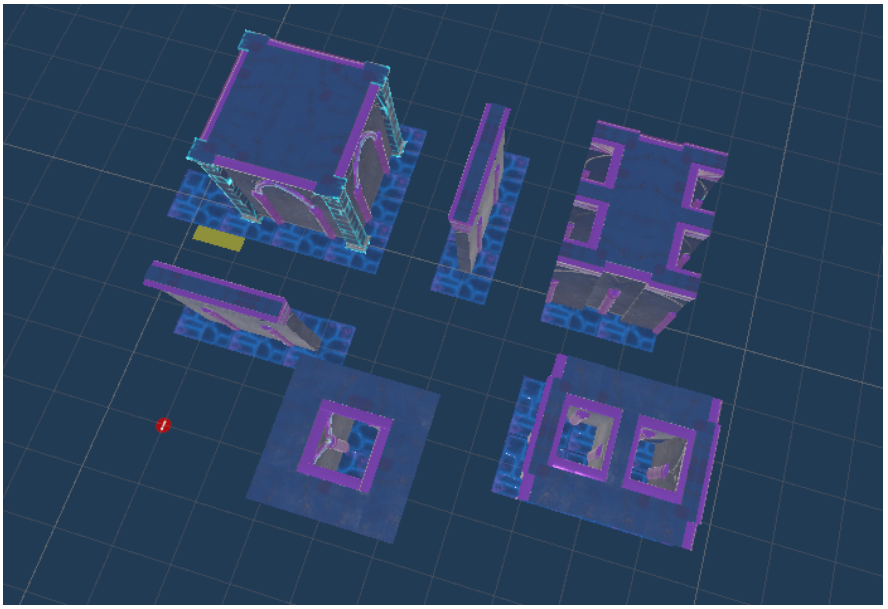

Figura 21: tileset de la primera mazmorra

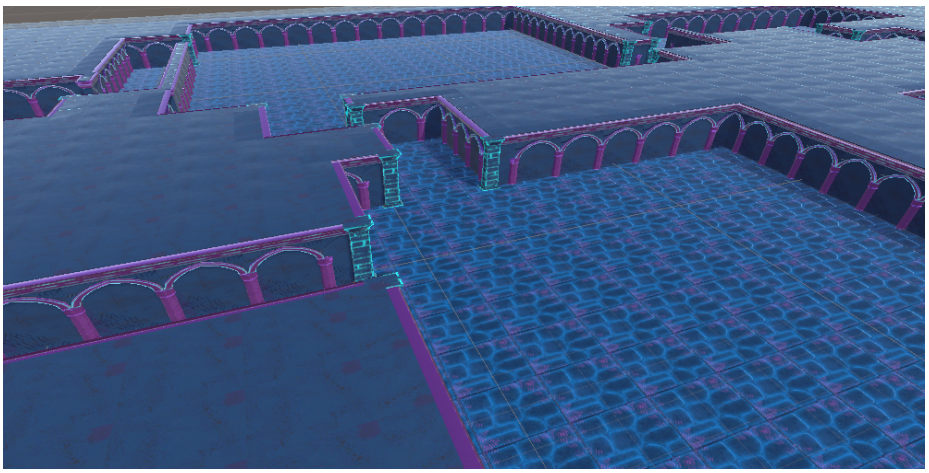

Figura 22: ejemplo de generación del tablero en el primer nivel

3.2.1.2. Personajes y objetos

Los personajes llevan el script “BoardCharacter”. Este script se encarga de realizar acciones referentes al personaje como elegir su siguiente acción, activar una habilidad, consumir un ítem o moverse. También se manejan sus animaciones.

Además de un script genérico para todos los personajes, aliados o enemigos, existe “LeaderController” que controla las acciones del jugador. Cuando el jugador desee por ejemplo mover a su personaje, “LeaderController” le indicará a “BoardManager” qué debe hacer. Se llama de esta forma porque se está controlando al líder actual del equipo, pero este puede cambiar en medio de la exploración si el jugador lo desea.

Cuando empiece un nuevo nivel, “EnemyManager” se encargará de controlar el estado de los enemigos: cargará una cantidad aleatoria en posiciones aleatorias, decidirá qué enemigos cargar basándose en la probabilidad de que salga cada uno (es más probable que salga un enemigo débil a uno fuerte) y también sacará más enemigos cuando pasen unos turnos desde que el último fue derrotado.

En cuanto a los ítems, hay que hacer una importante distinción: por una parte están los ítems en su estado dentro del tablero. En esta forma usan el script “BoardItem”, y se mostrarán en el mapa con un modelo 3D. Cuando un personaje interactúe con ello, se añadirá al inventario.

Dentro del inventario pasarán a encontrarse dentro del menú “bolsa”. Desde aquí podrán ser consumidos, y se utiliza el script “MenuManager” para ello. Cada ítem tiene también un script “ItemObject” que indica sus características como su valor y su tipo. Se ha decidido que sea un “ScriptableObject”, una clase especial de Unity pensada para almacenar datos.

```
Mensaje de Unity | 0 referencias
private void OnEnable()
{
    BoardManager.notifyCharactersToDecideAction += SelectNextAction;
    BoardManager.notifyCharactersToMove += MoveCharacterObject;
    BoardManager.onTurnEnded += ResetOnTurnEnded;
}

Mensaje de Unity | 0 referencias
private void OnDisable()
{
    BoardManager.notifyCharactersToDecideAction -= SelectNextAction;
    BoardManager.notifyCharactersToMove -= MoveCharacterObject;
    BoardManager.onTurnEnded -= ResetOnTurnEnded;
}
```

Figura 23: código en el que se muestra que BoardCharacter se subscribe a BoardManager, por lo que está atento a algunas acciones que ese script efectúa

3.2.2. IA de los personajes

Normalmente la inteligencia artificial se reserva para los enemigos. Sin embargo, mientras el jugador controla al líder del equipo sus compañeros son manejados de forma automática. A continuación se discuten los parámetros usados para crear una IA competente y cómo se implementó el *pathfinding*.

3.2.2.1. Toma de decisiones

Como se comentó con anterioridad, los personajes tienen tres posibles acciones: moverse, usar una habilidad o usar un objeto. Los aliados se comportan con una instrucción dada por el usuario como “ve por ahí”, “ve a por los enemigos”, “mantente cerca” y así. Por ello se explicará la toma de decisiones de los enemigos.

Normalmente un enemigo decide caminar por ahí, simplemente explorando el territorio. Sin embargo en cuanto un enemigo vea a un aliado irá a combatirlo. En cuanto el enemigo tenga a un aliado a rango, si lo tiene justo delante lo atacará. Si lo tiene a rango (por ejemplo, con un ataque que afecte a toda la habitación) habrá una pequeña probabilidad de que use ese ataque ya que si lo usase siempre sería injusto para el jugador. Si no, se dedicará a seguirlo hasta que alguno sea derrotado o lo pierda de vista.

Al principio del turno cada enemigo que pueda hacerlo tendrá una pequeña probabilidad de realizar un ataque a distancia. En caso de no hacerlo, se decidirá la acción tras la decisión del jugador. Si tras eso el jugador o un aliado suyo está a una casilla de distancia del enemigo, este atacará inevitablemente. Si no, este se moverá.

Se usa un sistema de “peso” para elegir la siguiente acción. Así por ejemplo en un turno dado puede ser que el movimiento tenga un peso de 4 y atacar tenga una de 1. En ese caso, hay una probabilidad de 4 entre 5 de que el enemigo se mueva.

Para una futura ampliación del juego se puede trabajar una toma de decisiones más refinada que tenga en cuenta factores como las debilidades de cada personaje, a cuántos puede dañar en un mismo turno e incluso evitar el “fuego aliado” (es decir, atacar a otro enemigo sin querer).

3.2.2.2. Pathfinding

Pathfinding se refiere a algoritmos para llegar de un punto a otro normalmente de la forma más rápida posible. En este caso se exploraron tres algoritmos que consisten en buscar el punto deseado y una vez este es alcanzado se guarda el camino que condujo a este. En cada caso se evalúan los vecinos de cada casilla para decidir el siguiente movimiento [20].

- *Breath First Search (BFS)*: la idea es ir ampliando un área de búsqueda llamado "frontera" en forma de anillo que se expande hasta que se encuentre la casilla deseada. Es muy simple de implementar y da buenos resultados en cuanto a la búsqueda, pero hace muchas evaluaciones innecesarias en este contexto por lo que puede conllevar problemas de rendimiento.

- *Greedy Best-First Search*: en este caso se evaluarán primero las casillas que resulten más próximas al destino. Esto se consigue añadiendo una función *heurística*, que en este caso evalúa la distancia de una coordenada a la otra. Resulta muchísimo más eficiente que la anterior para esta tarea pues se enfoca en llegar a un único punto. Sin embargo, puede dar resultados inexactos si hay obstáculos. Además como se muestra en la figura 24 el camino incluye varios movimientos en zigzag que no parecen muy naturales.

- *A**: es una mezcla de *Greedy Best-First Search* y el algoritmo de Dijkstra, el cual no se ha implementado pero que consiste en evaluar la distancia entre el punto actual y el punto inicial, no el final. En este caso se evaluará la distancia al punto inicial y al final a la vez. El resultado es un camino mucho más natural como se muestra en la figura 24 a cambio de evaluar algunas casillas más que en el anterior algoritmo.

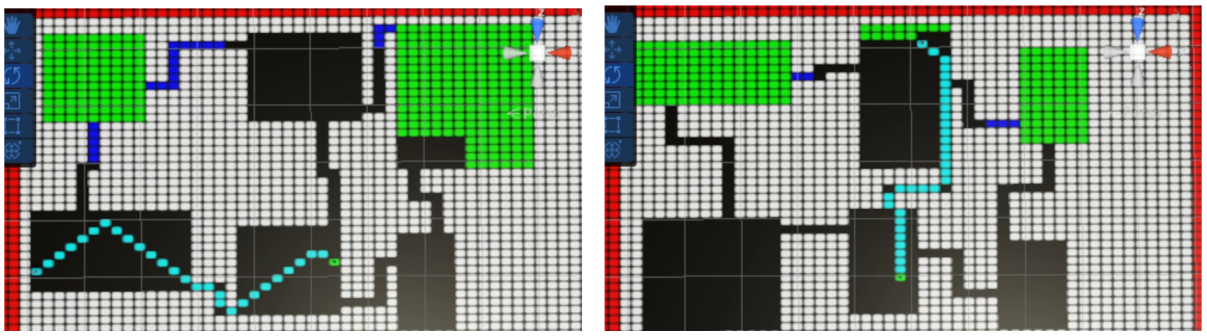

Figura 24: Greedy Best-First Search (izquierda) y A (derecha) en dos escenas distintas. En color blanco y rojo se representan muros, en verde y azul el suelo caminable, en negro las casillas evaluadas por el algoritmo y en celeste el camino resultante.*

3.2.3. Dificultad dinámica

Se pretende implementar la dificultad dinámica mediante un sistema de ranking, alcanzando de esta forma dos objetivos: por una parte el juego se ajusta ligeramente al jugador para reducir la frustración de los menos habilidosos y aumentar el reto para los expertos, y por otra se incentiva al usuario a rejugar los niveles hasta alcanzar las mejores puntuaciones.

Esta evaluación se realiza en tiempo real y por cada piso de la mazmorra. Desde que se entra a un territorio hasta que se sale, estos puntos son los que se tendrán en cuenta:

- Tiempo: el tiempo que tarda un jugador en completar un territorio. Por lo general se prefiere incentivar la exploración en estos juegos, por lo que este apartado se reserva para algunas misiones especiales.

- Enemigos derrotados y daño recibido: la cantidad de enemigos derrotados se tendrá en cuenta para subir el ranking del jugador, más no para bajarlo. Es decir, si decide evitarlos a todos y seguir adelante, no se le penalizará. Sin embargo, si decide evitarlos pero recibe mucho daño, entonces sí será un motivo para bajar su puntuación.

- Objetos encontrados: se valorará positivamente la cantidad de objetos encontrados en relación a los objetos disponibles totales. Es decir, recoger un objeto en una sala en la que ese era el único objeto se valora más que recoger dos objetos cuando había hasta cuatro en la sala.

- Uso de objetos: se podrá penalizar al jugador en caso de usar objetos demasiado poderosos en relación al territorio en el que se encuentra. Por ejemplo, si se usa un objeto para revivir en el nivel tutorial, se considerará una penalización.

- Nivel del grupo: en los juegos en los que existe la subida de niveles existe un término llamado *grinding*. Esto se refiere a combatir enemigos una y otra vez para aumentar el nivel del personaje. Es posible que el grupo tenga mucho más nivel que el que estaba pensado para un territorio en concreto, por lo que en estos casos en lugar de empezar la mazmorra con un nivel de dificultad medio se usará directamente uno más alto.
  También puede pasar al revés y que el grupo tenga menos nivel del recomendado. En ese caso no se cambiará nada, pues alguien que haga eso probablemente sea muy habilidoso así que no se le brindará ninguna ayuda especial.

La puntuación afectará a la dificultad del territorio de diversas formas, entre ellas:

- Enemigos: la cantidad de enemigos variará, así como su nivel y, en casos extremos, su comportamiento. Por ejemplo si a alguien le está yendo muy mal entonces los enemigos dejarán de perseguir al grupo tan seguidamente, o incluso puede verse afectada su puntería (por lo que algunos ataques podrían fallar).
  Si el territorio es lo suficientemente grande, tendrá una gran variedad de enemigos que pueden aparecer. En ese caso se puede tender más a los enemigos más débiles o a los más fuertes según el caso.

- Objetos: el porcentaje de aparición de objetos puede verse afectado, así como la calidad de estos. Si alguien está recibiendo demasiado daño, tal vez en la próxima sala sea más probable que haya objetos de curación.

- Trampas: si a un jugador le está yendo demasiado bien, pueden colocarse trampas de vez en cuando para complicarle un tanto la vida. Esto solo ocurriría si lleva varios pisos sacando las mejores puntuaciones.

Algo que nunca se verá afectado negativamente son las cosas relacionadas al protagonista y su grupo; quitarle control al jugador de sus únicas herramientas para explorar el mundo puede resultar frustrante, incluso si las diferencias son sutiles.

3.2.4. Game HUD

La web GamerDic [21] define el HUD como “un conjunto de iconos, números, mapas, etc. que durante el juego nos dan información sobre el estado de nuestra partida y/o nuestro personaje, como por ejemplo vida restante, ubicación, munición, objetos en uso, etc”.

Este proyecto no cubre los gráficos de las interfaces, por lo que su apariencia es únicamente orientativa.

- Minimapa: es información crucial para el jugador. Gracias a este puede saber qué habitaciones le quedan por recorrer y qué elementos hay en cada una. El mapa se va revelando a medida que se van explorando las salas. La primera idea de implementación fue usar una segunda cámara que únicamente renderiza una capa, “MinimapLayer”, e ignora el resto. Se pondría un tile extra encima de cada casilla que representase qué era: el jugador se muestra como una casilla de color amarilla, los enemgos como casillas rojas y las escaleras como casillas celestes. Las casillas verdes son habitaciones y las azules pasadizos. Las casillas blancas y rojas representan el espacio inutilizado y los límites de la mazmorra. El resultado es el mostrado en la figura 25.

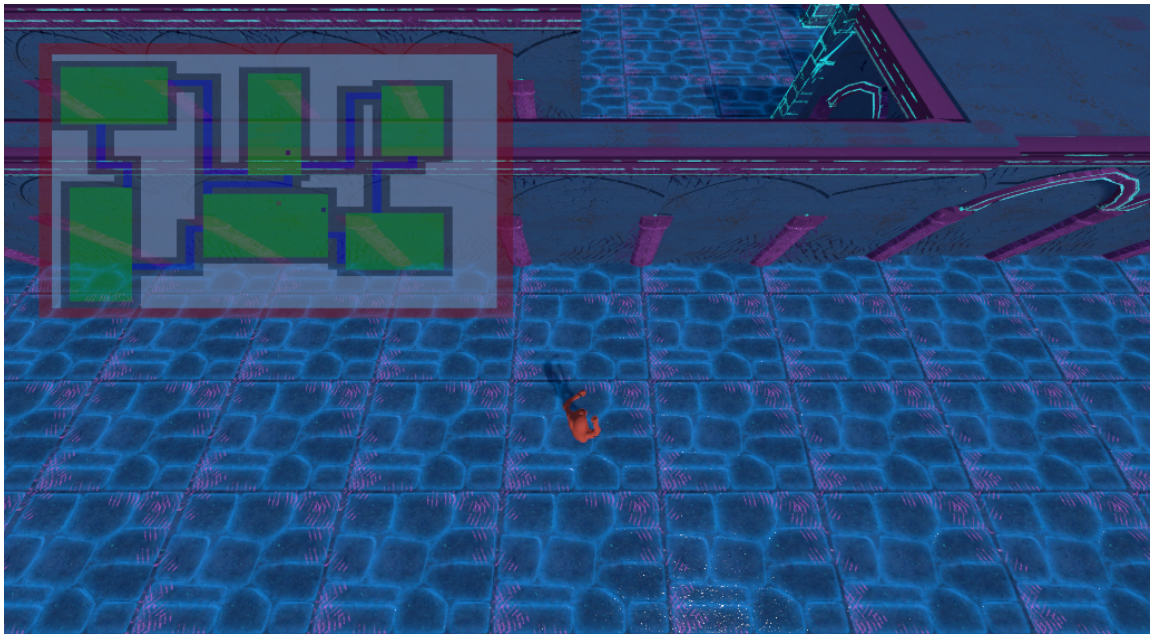

Figura 25: captura de pantalla del primer nivel del juego. El minimapa antiguo se encuentra en la parte superior izquierda de la pantalla

Sin embargo esto haría que el jugador tuviese demasiada información desde un inicio. Se implementó una lógica que muestra únicamente, aparte de los tiles de los personajes, los tiles de las habitaciones que el jugador ya ha visitado. Basta con entrar a una nueva habitación para desbloquearla en el minimapa. Cuando el jugador recorre un pasillo, este se va desbloqueando una casilla a la vez.

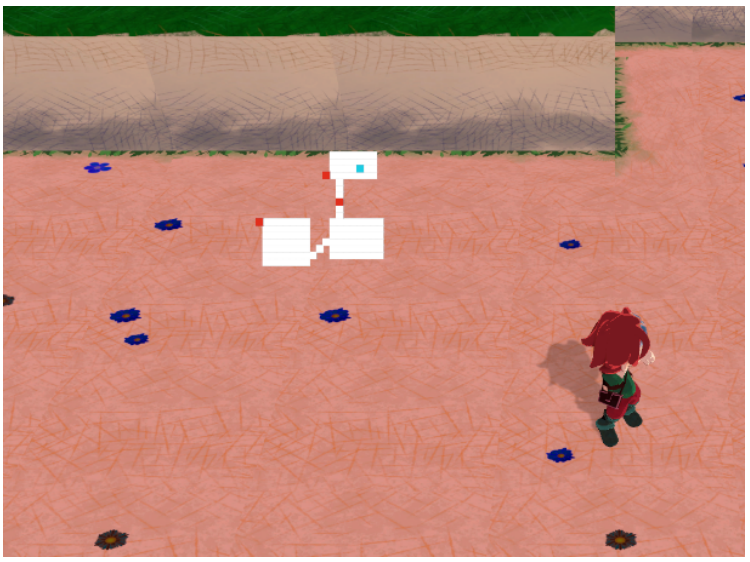

Figura 26: captura de la segunda mazmorra del juego donde se muestra el nuevo minimapa. Se enseñan únicamente los iconos de personajes y de suelo ya explorado

Por último hay que indicar que ocupa un espacio importante en la pantalla. Por ello es semitransparente y se puede mostrar u ocultar pulsando un botón.

- Información del personaje: tal y como el minimapa, se puede elegir que no se muestre por pantalla continuamente. Sin embargo, siempre que se abra el menú estará disponible. Muestra al líder junto a su nivel actual, su barra de vida y sus puntos de habilidad restantes. Se muestran con barras que se van vaciando, pero también con números para que queden claros los valores exactos. El icono del personaje muestra su estado actual (normal, paralizado, dormido, etc).

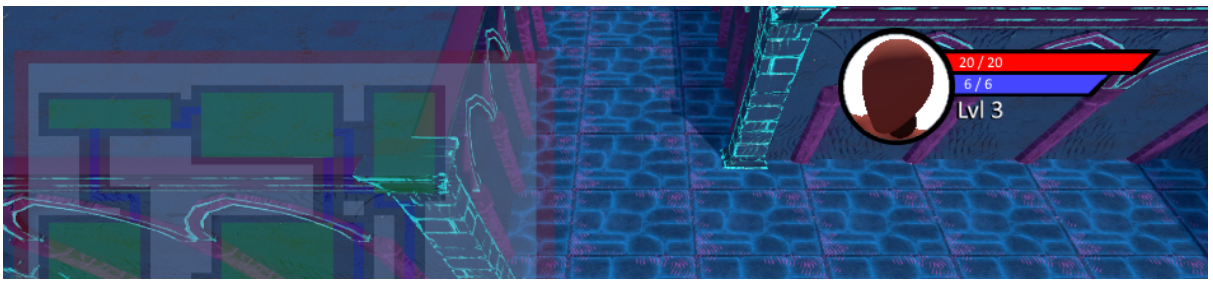

Figura 27: captura de pantalla de un nivel en el que se muestra la información del jugador.

- Indicador de daño: cuando un personaje recibe daño saldrá un número encima suyo que indicará cuánta vida se le ha quitado, tal y como se muestra en

- Historial de mensajes: cada vez que ocurre algo importante, ya sea información de un combate, indicar que un objeto se ha recogido o demás, se mostrará un mensaje por pantalla que lo indique. El historial de mensajes muestra los tres últimos mensajes, solo aparece cuando se añade uno nuevo y desaparece cuando pasan unos segundos sin haber cambiado nada.

Figura 28: captura del juego mostrando el historial de mensajes en su primera versión. También se ve el daño recibido encima del enemigo.

Se mejoró el historial al añadir elementos gráficos estéticos, usar una nueva fuente de texto y destacar las palabras clave de un mensaje. Por ejemplo, un texto que indique que el jugador recibió daño mostrará la palabra “Jugador” en verde y la cantidad de daño en rojo.

Figura 29: captura del juego mostrando el historial de mensajes mejorado.

El script “HistoryUIManager” se encarga de hacer los cálculos para que quepa la cantidad indicada de mensajes y se aproveche todo el espacio de la caja del historial.

```
// Space occupied by all lines: line height * shown lines
// Rest of space: All space - space occupied by all lines
// Space unoccupied (1): Rest of space / shown lines [because up and down difference is half of the other's]
// They will start to appear at the space's y - space unoccupied (1).
// separationBetweenLines: line height + space unoccupied (1)

float lineHeight = textOG.rect.height;
float maxSpace = bgSpace.rect.height;
maxSpace -= verticalPadding * 2;

float totalLinesSpace = lineHeight * maxShownTexts;
float restOfSpace = maxSpace - totalLinesSpace;

float freeSpace = restOfSpace / maxShownTexts;
separationBetweenLines = lineHeight + freeSpace;

// LineHeight / 2 because of pivot
float startingPointDiff = (freeSpace * 0.5f) + (lineHeight * 0.5f);
originalPosition = new Vector3(bgSpace.position.x + leftPadding, bgSpace.position.y - startingPointDiff, bgSpace.position.z);
```

Figura 30: función que aprovecha al máximo el tamaño del historial de mensajes e indica cómo deben colocarse las siguientes líneas de texto a aparecer.

- Menú de juego: al pulsar una tecla se mostrará el menú de juego. Sus contenidos son los mismos que se explicaron en el apartado de diseño.

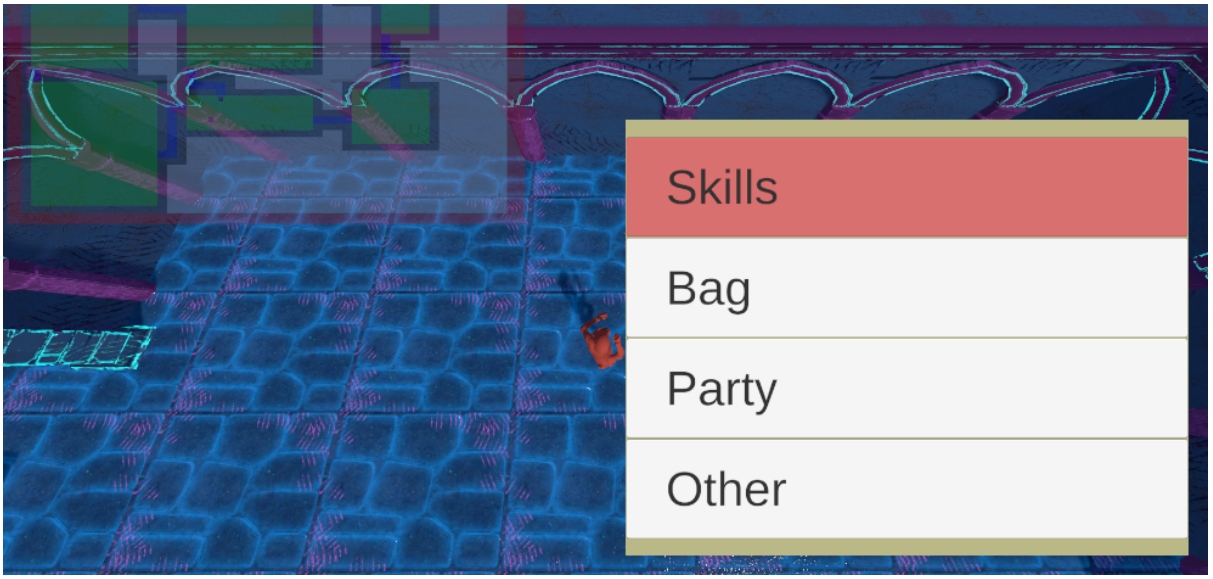

Figura 31: captura en la que se muestra el menú del juego.

4. Conclusiones

Durante el desarrollo del proyecto se han investigado a fondo los roguelikes y se han definido los puntos que definen al género en su estado más puro. Además, se han explorado distintos algoritmos para generar contenido de forma procedural, así como métodos de pathfinding efectivos. Se ha planteado también una mecánica que no había explorado con anterioridad. Si bien es cierto que es necesario realizar más pruebas y ajustes para refinar el sistema de dificultad dinámica para probar su efectividad, puede resultar como mínimo un punto llamativo para los jugadores habituales de este género.

Por supuesto hubo dificultades e imprevistos a lo largo del desarrollo, causados principalmente por adentrarse en terrenos no tan familiares. Sin embargo, se ha sabido encontrar alternativas y soluciones gracias a la planeación e investigación.

Se pudo cumplir el objetivo principal, que consistía en crear una demo jugable con un mapa generado proceduralmente, un personaje jugable, algún enemigo y un sistema de turnos para moverlos y realizar un combate básico.

Para realizar el proyecto se pusieron en práctica varios conocimientos adquiridos a lo largo de la carrera principalmente en lo referente a la programación. El resultado es un prototipo completamente escalable y que sirve como base para seguir creando un juego completo a futuro.

Para un desarrollo futuro se pueden añadir elementos que no se implementaron debido al tiempo, como la “zona neutral” en la que se encontrarán tiendas, tablón de misiones, zonas de entrenamiento, personajes con los que interactuar y demás. Se deben pulir los elementos presentes en las interfaces para que cuadren con la ambientación del juego. También resultaría interesante añadir el ranking online, donde se recojan las puntuaciones más altas en cada mazmorra y los mejores tiempos.

Bibliografía

[1] O. Korn y N. Lee, *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*. Springer Int. Publishing AG, 2017.

[2] J. Harris, *Exploring Roguelike Games*. CRC Press, 2020. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://doi.org/10.1201/9781003053576

[3] T. M. Terrasa, “El alma oscura del juego: Teoría y motivos recurrentes de la dificultad como estética ludoficcional”, doctoral thesis, Univ. Illes Balear., 2021. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: http://hdl.handle.net/10803/673118

[4] S. Zapata. *A Classic Roguelik*e. Temple of The Roguelike, 2014. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://blog.roguetemple.com/roguelike-definition/

[5] Some Body. *How AI works in Pokémon Mystery Dungeon: Red/Blue Rescue Team.* (31 de mayo de 2021). Accedido el 27 de septiembre de 2023. [Video en línea]. Disponible: https://www.youtube.com/watch?v=plke4eU_PU8

[6] S. Risi, J. Lehman, D. Dambrosio y K. Stanley, “Automatically Categorizing Procedurally Generated Content for Collecting Games”. In: Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at the 9th International Conference on the Foundations of Digital Games, FDG 2014.

[7] A. A. Garzia, *Roguelike Development with JavaScript*. Berkeley, CA: Apress, 2020. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://doi.org/10.1007/978-1-4842-6059-3

[8] M. B. Garda, *Neo-rogue and the essence of roguelikeness*. Homo Ludens, *1*(5), 59-72, 2017.

[9] J. P. Zagal y R. Altizer, “Examining 'RPG elements': Systems of character progression”. In *FDG*, 2014

[10] Edmund McMillen y Florian Himsl. The Binding of Isaac. (2011). Edmund McMillen. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/

[11] Mossmouth. Spelunky. (2013). Mossmouth. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://store.steampowered.com/app/239350/Spelunky

[12] Poncle. Vampire Survivors. (2022). poncle. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://store.steampowered.com/app/1794680/Vampire_Survivors

[13] Spike Chunsoft Co., Ltd. Shiren the Wanderer: The Tower of Fortune and the Dice of Fate. (2020). Spike Chunsoft Co., Ltd. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://store.steampowered.com/app/1178790/Shiren_the_Wanderer_The_Tower_of_Fortune_and_the_Dice_of_Fate/

[14] Spike Chunsoft Co., Ltd. Pokémon Mystery Dungeon: Rescue Team DX. 2020 [Videojuego].

[15] L. J. Arce, “Desarrollo de videojuegos”. Mendoza: Universidad de Aconcagua, 2011.

[16] M. Dhule, *Beginning Game Development with Godot*. Berkeley, CA: Apress, 2022. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://doi.org/10.1007/978-1-4842-7455-2

[17] J. I. L. Lizarraga, “Motores de desarrollo de videojuegos más populares”. Universidad Politécnica de Sinaloa, 2017.

[18] Sunny Valley Studio. *Random Walk Algorithm - P4 - Unity Procedural Generation of a 2D Dungeon*. (18 de diciembre de 2020). Accedido el 27 de septiembre de 2023. [Video en línea]. Disponible: https://www.youtube.com/watch?v=F_Zc1nvtB0o

[19] TheZZAZZGlitch. *Pokémon Mystery Dungeon - deconstructing the dungeon generation algorithms*. (5 de julio de 2021). Accedido el 27 de septiembre de 2023. [Video en línea]. Disponible: https://www.youtube.com/watch?v=fudOO713qYo

[20] A. Patel, "Introduction to the A* Algorithm", Red Blob Games, 2014.

[21] “HUD | Definición en GamerDic”. DeVuego | Base de Datos del Videojuego Español. Accedido el 27 de septiembre de 2023. [En línea]. Disponible: https://www.devuego.es/gamerdic/termino/hud