Entrega 1
Proyecto: Dungeon Fighter

Contenido

1. Descripción y Justificación del Proyecto 3
1.1 Descripción 3
1.2 Justificación 3

2. Mecánicas del Proyecto 4

3. Valoración de Alternativas de Mercado 4

4. Stack Tecnológico Elegido 5

5. Objetivos del Proyecto 5
5.1 Objetivos Específicos 5

6. Requisitos del Sistema 6
6.1 Requisitos Funcionales 6
6.2 Requisitos No Funcionales 6
6.3 Requisitos de Interfaz 6

7. Casos de Uso 7

8. Modelo y Diseño de Base de Datos (Firebase Firestore) 8

9. Descripción Detallada del Videojuego 9
9.1 Forma 9
9.2 Mecánica 9
9.3 Contexto 9
9.4 El estilo artístico 9

1. Descripción y Justificación del Proyecto

1.1 Descripción

Dungeon Fighter es un videojuego de acción y exploración en 2D con una perspectiva "Top-Down", comúnmente conocido como "Falso 3D". Desarrollado en el motor de videojuegos Unity y programado en C#. El proyecto situará al jugador en el rol de un luchador que deberá aventurarse en una serie de mazmorras interconectadas.

El bucle principal del juego consistirá en explorar diferentes salas, derrotar a los enemigos que habitan, gestionar recursos limitados (como la vida y las llaves) y avanzar hasta encontrar y vencer al jefe final de la mazmorra. El juego se inspira en clásicos del género "Dungeon Crawler y "Roguelite", pero se enfocará en una experiencia de un solo jugador.

El jugador controlará al personaje con movimiento en cuatro direcciones "W, A, S, D" y dispondrá de un ataque básico. A medida que explore, encontrará cofres que alberguen recompensas (monedas, llaves para abrir puertas bloqueadas, o pociones para restaurar la vida). Los enemigos contarán con inteligencia artificial básica, permitiéndoles patrullar zonas y perseguir al jugador al entrar en su rango de detección.

El proyecto implementará un sistema completo de interfaz de usuario "UI", incluyendo un menú principal, un menú de pausa, una pantalla de "Game Over" y un HUD (Heads-Up Display) en partida que mostrará la información vital como la vida del jugador y su inventario.

Finalmente, una característica clave será el sistema de persistencia de datos. El juego contará con un sistema de guardado y carga de partidas que almacenará el estado del juego (posición, vida, inventario...) y el estado del mundo (enemigos derrotados, cofres abiertos...) en una base de datos en la nube (Firebase), permitiendo al usuario continuar su progreso en cualquier momento.

1.2 Justificación

Este proyecto se justifica como una buena síntesis de las competencias y contenidos adquiridos durante el Ciclo Formativo de Desarrollo de Aplicaciones Multiplataforma (DAM). Su desarrollo nos permitirá aplicar de forma práctica una amplia gama de conocimientos:

1. Programación Orientada a Objetos (POO): El desarrollo de scripts en C# para Unity requerirá un sólido diseño de clases para gestionar al jugador, los enemigos, lis ítems, la UI y la lógica general del juego.

2. Gestión de Base de Datos: La implementación de un sistema de guardado funcional nos obligará a diseñar y consumir una base de datos. Al optar por Firebase, se abordarán contenidos de NoSQL (Firestore) como sistemas de autentificación de usuarios.

3. Interfaces de Usuarios (UI/UX): El diseño de los menús y el HUD del juego pondrá en práctica los principios de diseño de interfaces, asegurando que la experiencia del usuario sea intuitiva y agradable.

4. Arquitectura de Software: Deberemos diseñar una arquitectura de proyecto limpia para que el código sea mantenible y escalable (por si queremos modificar a futuro).

2. Mecánicas del Proyecto

Las mecánicas de este proyecto además de todo lo que queremos abarcar, tendremos que poderla completar en el plazo asignado para el proyecto final.

Características Incluidas:

- Sistema de Control de Jugador (movimiento en 4 direcciones, ataque).
- Un tipo de arma (espada, arco...).
- Varios tipos de enemigos básicos con IA (patrulla, persecución y ataque).
- Un Jefe Final con mecánicas de ataque diferentes a los enemigos básicos.
- Un nivel de mazmorra compuesto por 5-10 salas interconectadas.
- Ítems consumibles (pociones, llaves...)
- Sistema de vida para jugadores y recompensas.
- Sistema de cofres con recompensas.
- Sistema de guardado/carga de partida.
- Sistema de autenticación de usuarios (Login/Registro) usando Firebase Authentication.
- Flujo de UI completo: Menú principal, pausa, game over, HUD...
- Efectos de sonido SFX para acciones clave y música de fondo.

3. Valoración de Alternativas de Mercado

El género de "Dungeon Crawler" top-down es un mercado maduro y popular, especialmente en la escena "indie".

Algunos referentes clave son:

- The Binding of Isaac: Un pilar del género "roguelite". Su éxito radica en la generación procedural y la inmensa cantidad de ítems. Dungeon Fighter no competirá en la generación procedural, sino que ofrecerá una experiencia más diseñada y lineal.
- Enter the Gungeon: Similar en perspectiva y género, pero enfocado en el combate con armas de fuego. Comparte la estética pixel art y el bucle de "limpiar salas.
- Clásicos (ej. The Legend of Zelda: A Link to the Past): Aunque son juegos de aventura, establecieron las convenciones del movimiento top-down, la exploración de mazmorras y el combate que Dungeon Fighter busca emular en su sistema.

Diferenciación: Dungeon Fighter no pretende competir comercialmente con estos títulos. Su objetivo será ser un proyecto académico robusto que demuestre competencias técnicas. Su principal diferenciador será ser una experiencia de autor, corta y pulida, que use tecnologías modernas de backend (Firebase) para la gestión de datos del jugador, algo que muchos juegos indie pequeños resuelven localmente.

4. Stack Tecnológico Elegido

Categoría	Tecnología	Justificación
Motor	Unity	Es el estándar de la industria para el desarrollo de juegos 2D y 3D multiplataforma. Ofrece un ecosistema robusto, un sistema de Tilemaps 2D y una amplia documentación.
Lenguaje	C#	Es el lenguaje de scripting nativo de Unity.
IDE	Visual Studio	Es el IDE de elección para el desarrollo en C# y se integra perfectamente con Unity.
Gráficos	Sistema Tilemap 2D de Unity	Herramienta integrada de Unity para crear niveles 2D basados en "tiles" de 16x16, permitiendo un diseño rápido y eficiente.
Animaciones	Unity Animator	Sistema de máquinas de estado de Unity para gestionar las animaciones 2D basadas en "sprites".
BD	Firebase	Usa los servicios de Firebase, usando colecciones y documentos.
Rol de Admin	Firebase Console	La propia consola de Firebase permitirá un "rol de administrador" para consultar, editar o eliminar datos de usuarios y partidas, cumpliendo el requisito sin necesidad de desarrollar una ventana de admin desde cero.
Control de Versiones	Git / Github	Estándar indiscutible para el control de versiones, permitiendo el seguimiento de cambios, el trabajo en ramas y la entrega del proyecto.

5. Objetivos del Proyecto

Desarrollar un videojuego 2D funcional, completo y pulido de tipo "dungeon crawler" utilizando Unity y C#, integrando servicios en la nube (Firestore) para la persistencia de datos y autenticación de usuarios.

5.1 Objetivos Específicos

1. Implementar un sistema de control de jugador preciso y responsivo.

2. Diseñar y programar la IA de al menos dos tipos de enemigos con comportamientos de patrulla, persecución y ataque.

3. Construir un nivel de juego funcional utilizando el sistema 2D Tilemap de Unity, con salas interconectadas.

4. Desarrollar el flujo completo de la interfaz de usuario (menú, pausa, game over, HUD...).

5. Integrar Firebase Authentication para permitir a los usuarios crear cuentas y hacer login.

6. Implementar un sistema de guardado y carga en Firebase Firestore que almacene el estado del jugador y del mundo.

7. Asegurar el cumplimiento de todos los requisitos (funcionales, no funcionales y de interfaz) defiindos.

6. Requisitos del Sistema

6.1 Requisitos Funcionales

- RF-01: El usuario debe poder crear una cuenta y autenticarse en el sistema (Firebase Auth).
- RF-02: El sistema debe guardar el progreso de la partida en la nube asociada al usuario autenticado.
- RF-03: El sistema deberá permitir cargar una partida guardada previemante.
- RF-04: El jugador deberá poder mover su personaje en 4 direcciones (arriba, abajo, izquierda y derecha).
- RF-05: El jugador deberá poder realizar una acción de ataque.
- RF-06: El jugador y los enemigos deberá tener un sistema de vida.
- RF-07: El jugador deberá morir y ver una pantalla de "Game Over" cuando su vida llegue a 0.
- RF-08: Los enemigos deberán ser eliminados cuando su vida llegue a 0.
- RF-09: Los enemigos deberán patrullar un área designada cuando el jugador no esté cerca.
- RF-10: Los enemigos deberán perseguir al jugador cuando entren en su rango de detección.
- RF-11: El jugador deberá poder pausar el juego, mostrando un menú de pausa.
- RF-12: El jugador deberá poder abrir cofres para obtener recompensas.
- RF-13: El estado de los cofres deberán persistir en el guardado.
- RF-14: El estado de los enemigos deberán persistir en el guardado.
- RF-15: El jugador deberá poder usar llaves para abrir puertas bloqueadas.
- RF-16: Un administrador deberá poder ver los datos de las partidas guardadas.

6.2 Requisitos No Funcionales

- RNF-01: El juego deberá ejecutarse en plataformas de escritorio como Windows.
- RNF-02: El juego deberá mantener una estética visual coherente (Pixel Art 16x16).
- RNF-03: El juego deberá incluir efectos de sonido para acciones y música de fondo.
- RNF-04: El tiempo de guardado o carga de partida no deberá demorarse demasiado.
- RNF-05: El código fuente deberá estar escrito en C#, siguiendo estándares de nomenclatura de C# adquiridos en el grado superior, y deberá estar debidamente comentado.
- RNF-06: El proyecto deberá estar gestionado bajo un sistema de versiones (Git).

6.3 Requisitos de Interfaz

- RI-01: El menú principal deberá mostrar las opciones "Nueva Partida, "Cargar Partida" y "Salir".
- RI-02: El HUD deberá mostrar en todo momento la vida del jugador.
- RI-03: El menú de pausa deberá mostrar las opciones "Continuar", "Guardar Partida" y "Volver al Menú".
- RI-04: La pantalla "Game Over" deberá mostrar las opciones "Cargar Último Guardado" y "Volver al Menú".
- RI-06: El juego deberá mostrar pantallas de Login y Registro antes del menú.

7. Casos de Uso

- CU-01: Autenticación de Usuario
  - Actor: Jugador.
  - Descripción: El jugador inicia el juego. Se le presenta una pantalla de Login. Si no tiene cuenta, puede ir a Registro, ingresar un email y contraseña, y crear una cuenta (Firebase Auth). Si ya tiene cuenta, ingresa sus credenciales y accede al menú principal.

- CU-02: Iniciar y Jugar Partida
  - Actor: Jugador.
  - Descripción: Desde el menú principal, el jugador selecciona "nueva partida" o "cargar partida". El juego carga el nivel. El jugador explora salas, evitando trampas y combatiendo enemigos.

- CU-03: Combatir contra Enemigo
  - Actor: Jugador, Enemigo.
  - Descripción: El jugador entra en el rango de visión de un enemigo. El enemigo persigue al jugador. El jugador ataca al enemigo, reduciendo su vida. El enemigo ataca al jugador si está en el rango, reduciendo la vida del jugador. El combate terina cuando uno de los dos llegue a 0 de vida.

- CU-04: Recolectar Ítem
  - Actor: Jugador.
  - Descripción: El jugador derrota a todos los enemigos de una sala y aparece un cofre. El jugador interactúa con el cofre. El cofre se abre (y su estado se marca como "abierto"), y un ítem (ej. una poción) se añade al inventario del jugador.

- CU-05: Guardar Progreso
  - Actor: Jugador.
  - Descripción: El jugador abre el menú de pausa y selecciona "Guardar Partida". El sistema recompila el estado actual (posición, vida, inventario del jugador, estado de enemigos y cofres) y lo envía a la base de datos Firestore, asociándolo al userId del jugador autenticado.

- CU-06: Gestión de Datos (Rol Admin)
  - Actor: Administrador.
  - Descripción: El administrador podrá acceder a la consola web de Firebase. Navegar a la colección de "partidasGuardadas". Puede visualizar los documentos JSON de las partidas, editarlos manualmente (ej. para dar un ítem a un jugador) o eliminarlos (ej. resetear el progreso de un juagador de prueba).

8. Modelo y Diseño de Base de Datos (Firebase Firestore)

Se utilizará una base de datos NoSQL (Firestore) debido a su flexibilidad y perfecta integración con Unity y Firebase Auth. La estructura de datos principal estará organizada en colecciones.

- Colección: usuarios
    - Almacena información pública o de perfil de los usuarios. Se creará automáticamente un documento por cada usuario que se registre en Firebase Auth. Usando su UID como ID del documento.
    - usuarios/{idUsuario}
        - email: (String) adrian@gmail,com
        - nombreUsuario: (String) “Adri”
        - fechaCreacion: (Timestamp) Fecha de creación de la cuenta.
        - esAdmin: (Boolean) false (Este campo permitirá dar roles de admin si el juego lo necesitara, aunque el requisito se cumple con la consola.

- Colección: partidasGuardadas
    - Almacena los datos de las partidas guardadas. Cada documento es una partida.
    - partidasGuardadas/{idUsuario}
        - idUsuario: (String) “uid_usuario_propietario” (Referencia al documento en la colección “usuarios”.
        - ultimaActualizacion: (Timestamp) Fecha y hora del guardado.
        - nombrePartida: (String) “Mazmorra Nivel 1”
        - datosJugador: (Map)
            - vida: (Number) 3
            - vidaMaxima: (Number) 3
            - posX: (Number) 15
            - poxY: (Number) -5
            - nombreEscena: (String) “Nivel_Mazmorra_1”
        - datosInventario: (Map
            - monedas: (Number) 100
            - llaves: (Number) 10
            - pociones: (Number) 2
        - estadoMundo: (Map)
            - enemigosElinimados: (Array<String>) [“id_enemigo01”, “id_enemigo_02”] (Almacena los IDs de los enemigos ya derrotados.
            - cofresAbiertos: (Array<String>) [id_cofre_01”] (Almacena los IDs de los cofres ya abiertos).

9. Descripción Detallada del Videojuego

9.1 Forma

Dungeon Fighter es un videojuego 2D que utiliza una perspectiva cenital (top-down). Aunque todos los gráficos son "sprites" 2D, la vista desde arriba simulará una profundidad (falso 3D), permitiendo al jugador moverse en un plano X-Y mientras percibe la altura de los muros.

9.2 Mecánica

El bucle principal se basará en la exploración y el combate.

1. Explorar: El jugador entra en una nueva sala de la mazmorra.
2. Combatir: Si la sala contiene enemigos, las puertas pueden cerrarse. El jugador deberá derrotar a todos los enemigos usando su ataque de espada, esquivando los ataques enemigos.
3. Recompensa: Al limpiar la sala, las puertas se abrirán y podrá aparecer un cofre.
4. Avanzar: El jugador elegirá la siguiente sala a la que entrar, gestionando su vida y los ítems que haya encontrado.
5. Jefe: El jugador eventualmente encontrará la sala del jefe, que requerirá una llave especial, o haber limpiado ciertas salas.
6. Victoria: Al derrotar al jefe, el juego se completará.

9.3 Contexto

El jugador asumirá el papel de un valiente Luchador. Este héroe se adentrará en la "Mazmorra" con el objetivo de erradicar el mal que la habita. La narrativa es minimalista y se encuentra a través del entorno, centrándose en la atmósfera y la acción.

9.4 El estilo artístico

El estilo será Pixel Art de 16x16 bits. Este estilo retro buscará evocar la sensación de los juegos de la era antigua de los videojuegos.

Target de Usuarios:

El público objetivo serán los jugadores entre 16 y 30 años que disfrutarán de:

- Juegos de acción "indie".
- El género "dungeon crawler" o "roguelite".
- La estética retro Pixel Art.
- Una experiencia de juego directa, desafiante pero corta, ideal para sesiones de juego de 20-30 min.