Instituto Puig Castellar
Ciclo formativo: Desarrollo de aplicaciones Multiplataforma
Grado: CFGS
Crédito de Síntesis / Proyecto intermodular

PROJECT: BERSERK

Videojuego desarrollado en Unity

Autor: Adrian Lara Zafra
Tutor: Daniel Colomer Travé
Curso: 2025-2026
Fecha de entrega: 16/05/2026

Resumen del proyecto

Este proyecto tiene como finalidad el desarrollo de un videojuego, impulsado por el interés personal en este medio como forma de expresión artística, comparable al cine o la literatura cuando se emplea de manera adecuada. Los videojuegos no solo representan una herramienta de entretenimiento, sino también un canal creativo capaz de transmitir emociones, ideas y experiencias interactivas al jugador.

El desarrollo se ha llevado a cabo utilizando el motor de desarrollo Unity y el lenguaje de programación C#, herramientas ampliamente utilizadas en la industria por su versatilidad y potencia. Estas tecnologías han permitido implementar mecánicas de juego, sistemas de interacción y elementos visuales de manera eficiente y estructurada.

Para garantizar una experiencia satisfactoria, se ha realizado un análisis previo de videojuegos de temática similar, estudiando sus mecánicas, diseño de niveles y dinámicas de jugabilidad. A partir de este estudio, se han aplicado principios de diseño centrados en el usuario, buscando un equilibrio entre accesibilidad y desafío.

El resultado es un videojuego concebido para ofrecer una experiencia entretenida, intuitiva y apta para todo tipo de jugadores, combinando creatividad, técnica y planificación en un proceso de desarrollo estructurado.

Palabras clave

Juego Unity 3D Indie Simple

Abstract

This project aims to develop a video game, driven by a personal interest in this medium as a form of artistic expression, comparable to film or literature when used appropriately. Video games are not only a form of entertainment but also a creative channel capable of transmitting emotions, ideas, and interactive experiences to the player.

The development was carried out using the Unity engine and the C# programming language, tools widely used in the industry for their versatility and power. These technologies allowed for the efficient and structured implementation of game mechanics, interaction systems, and visual elements.

To ensure a satisfying experience, a preliminary analysis of video games with similar themes was conducted, studying their mechanics, level design, and gameplay dynamics. Based on this study, user-centered design principles were applied, seeking a balance between accessibility and challenge.

The result is a video game designed to offer an entertaining, intuitive experience suitable for all types of players, combining creativity, technical skill, and planning in a structured development process.

Keywords
Simple Indie Unity 3D Game

Índice

1. Presentación del proyecto

1.1 Introducción

Mi proyecto es un videojuego en 3D y en tercera persona en Unity con pocos recursos que intenta ser una experiencia atractiva para todos los públicos y pasar un buen rato.

El juego ha sido desarrollado en su totalidad con *Unity* un motor de desarrollo gratuito con buen soporte a la hora de tener problemas o si eres "inexperto" a la hora de desarrollar tu primer videojuego, como es mi caso, junto con un amplio contenido de tutoriales en base de videos de *YouTube* o webs como *Reddit* en las que encontrarás ayuda a la mayoría de problemas.

Ahora hablando del juego en cuestión, es un juego *Vampire Survivor* el cual se basa en sobrevivir oleadas de enemigos mientras tu personaje va mejorando y subiendo de nivel. Para no hacerlo muy básico he decidido también que contará con un modo historia, aunque bastante simple, para la gente que le gusta ese apartado. Los modelos de personajes y de enemigos no son creados por mi por cuestión de tiempo, igual para las animaciones, pero se ha intentado no poner modelos simples o poco detallados.

Entrando en detalle sobre el *gameplay* el jugador simplemente tendrá que moverse por el mapa y sobrevivir a los enemigos. El ataque del mismo será automático y en área, como son en la mayoría de juegos de ese estilo. Por parte del enemigo he programado una IA, aunque simple, funcional ya que simula que está "vivo" al moverse libremente por el mapa y se acercará al jugador cuando entre en su área de ataque. Los enemigos los dividiremos en tres grupos: los normales (hacen lo anteriormente comentado), los acosadores (siempre perseguirán al jugador) y los jefes (aparecerá uno al final del nivel y el jugador deberá vencer para avanzar al próximo). El propósito de esto es evitar dar una sensación de repetitividad a la hora de enfrentar enemigos.

Volviendo al jugador, este podrá subir de nivel al conseguir experiencia matando a enemigos. Al subir de nivel le aparecerá una selección de objetos/habilidades/mejoras con las que quiere meter también un factor de estrategia, porque dichas mejoras pueden hacerte subir una estadística, pero te bajara otra (ej: +5 de VT | - 3 de daño). Con esto lo que quiero conseguir es que cada partida sea un poco diferente a la anterior y pensarte que te beneficia más a la hora de enfrentarse a la ronda.

Para aplicar todo lo que he comentado anteriormente me he documentado y analizado lo que hace especial a esa clase de videojuegos (*Vampire Survivor, Megabonk, Brotato*…).

1.2 Contexto

El crecimiento y la popularización de la industria de los videojuegos a lo largo de los años han facilitado el desarrollo de juegos independientes en todo el mundo. Esto ha sido posible, en gran parte, gracias a motores gráficos proporcionados por empresas como *Epic Games* con *Unreal Engine*, *Unity Technologies* con *Unity*, y la organización *Godot Foundation* con *Godot Engine*, una opción más reciente y de código abierto.

Estos motores destacan porque, en muchos casos, son gratuitos o tienen modelos de uso accesibles, lo que permite que personas con presupuestos limitados puedan iniciarse en el desarrollo de videojuegos sin una gran inversión inicial.

He utilizado *Unity* principalmente por su amplia comunidad de usuarios, que ofrece una gran cantidad de recursos, como foros, documentación y contenido compartido. Además, la disponibilidad de numerosos tutoriales facilita el aprendizaje y la resolución de problemas, lo que resulta especialmente útil para quienes se inician en el desarrollo de videojuegos.

Dentro de la industria de los videojuegos existen numerosos géneros (shooters, plataformas, battle royale, entre otros), pero hay uno que destaca por su simplicidad y su carácter adictivo: el género popularmente conocido como *"Vampire Survivors*. Este género se basa en mecánicas sencillas, como el ataque automático, la acción constante y la toma de decisiones estratégicas a la hora de elegir habilidades para mejorar al personaje.

1.3 Justificación

El desarrollo de videojuegos constituye un ámbito especialmente relevante dentro del campo del software, ya que integra conocimientos de programación, diseño interactivo y creación de experiencias para el usuario. En los últimos años, la accesibilidad a motores de desarrollo como *Unity*, desarrollado por *Unity Technologies*, ha permitido que proyectos de menor escala puedan llevarse a cabo sin necesidad de grandes recursos económicos ni equipos amplios, facilitando así la entrada a nuevos desarrolladores.

En este contexto, el desarrollo de un videojuego en 3D y en tercera persona basado en el género popularizado por títulos como Vampire Survivors resulta especialmente adecuado. Este tipo de juegos se caracteriza por mecánicas simples pero altamente adictivas, como el ataque automático, la supervivencia frente a oleadas de enemigos y la progresión del personaje mediante mejoras. Estas características permiten centrarse en la implementación de sistemas clave como el control del jugador, la inteligencia artificial de los enemigos o la gestión de la progresión, manteniendo al mismo tiempo una experiencia entretenida para el usuario.

Además, la incorporación de elementos adicionales, como un modo historia sencillo o la variedad en los tipos de enemigos, aporta un valor añadido al proyecto al intentar evitar la repetitividad y enriquecer la jugabilidad. De este modo, no solo se replican las bases del género, sino que también se introducen pequeñas innovaciones que mejoran la experiencia global del jugador.

Por otro lado, el uso de recursos externos para modelos y animaciones responde a una limitación realista de tiempo, lo cual refleja una práctica habitual en el desarrollo independiente. Esto permite centrar los esfuerzos en aspectos fundamentales como la programación del gameplay, incluyendo sistemas de experiencia, selección de mejoras con impacto estratégico y comportamiento de los enemigos mediante inteligencia artificial básica.

Por todo ello, el desarrollo de este proyecto resulta relevante, ya que permite aplicar y consolidar conocimientos de programación en un entorno realista, al mismo tiempo que se crea un producto interactivo completo. Asimismo, demuestra cómo, con recursos limitados pero haciendo uso de herramientas accesibles y una buena planificación, es posible desarrollar una experiencia jugable atractiva para un público amplio.

1.4 Objetivos

El objetivo principal de este proyecto es desarrollar un videojuego funcional en 3D y tercera persona con Unity, basado en el género de *Vampire Survivors*, que sea entretenido y accesible para cualquier tipo de jugador, aplicando lo aprendido sobre programación y desarrollo de videojuegos.

Para lograrlo, se han definido los siguientes objetivos específicos:

- Implementar un sistema de control del jugador en tercera persona con un movimiento fluido y un ataque automático en área, tal y como es habitual en este tipo de juegos.
- Programar una IA básica para los enemigos con tres comportamientos distintos: enemigos normales, acosadores que persiguen al jugador, y jefes finales que aparecen al acabar cada nivel.
- Crear un sistema de progresión donde el jugador suba de nivel matando enemigos y pueda elegir mejoras que afecten a sus estadísticas, añadiendo un pequeño factor estratégico a cada partida.
- Integrar modelos 3D y animaciones de terceros de forma que encajen bien con el estilo visual del juego y funcionen correctamente en el gameplay.
- Organizar el juego en niveles con oleadas de enemigos cada vez más difíciles, con un jefe final que el jugador debe derrotar para avanzar.
- Conseguir que el juego esté completo y sea jugable de principio a fin, con un acabado suficientemente bueno para presentarlo como proyecto finalizado.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

(Explica cómo planteas desarrollar el proyecto y si es viable a nivel técnico, temporal y de recursos.)

El desarrollo de este proyecto se llevará a cabo, como se ha mencionado anteriormente, utilizando el motor de desarrollo Unity y el lenguaje de programación C#, el cual representa una herramienta nueva para mí. Debido a esta falta de experiencia previa, la fase inicial del proyecto se centró en la adquisición de los conocimientos fundamentales necesarios para su correcto uso.

Para ello, se recurrió principalmente a tutoriales y recursos formativos en formato audiovisual, especialmente videos, que permitieron comprender los conceptos básicos del lenguaje, como la sintaxis, la estructura de los programas, el uso de variables, funciones y la lógica de programación orientada a objetos. Este proceso de aprendizaje resultó esencial para familiarizarme tanto con el entorno de desarrollo como con la manera en que Unity integra el código en la creación de videojuegos.

Una vez adquirida una base sólida en estos aspectos, se dio paso a la siguiente fase, centrada en el desarrollo práctico del videojuego. En esta etapa, se comenzaron a aplicar los conocimientos teóricos previamente adquiridos, abordando la implementación de mecánicas, la interacción entre objetos y la construcción progresiva del proyecto. Este enfoque gradual permitió no solo avanzar en la creación del juego, sino también consolidar y ampliar los conocimientos sobre C# y su aplicación dentro del entorno de desarrollo.

2.2 Metodología de trabajo

Para el desarrollo del proyecto se ha seguido una metodología de trabajo iterativa basada en la experimentación y la mejora continua.

En primer lugar, para cada nueva funcionalidad, se ha realizado una fase de investigación previa con el objetivo de comprender cómo implementar dicha característica dentro del

juego. A continuación, se ha llevado a cabo una implementación inicial en forma de prototipo o prueba.

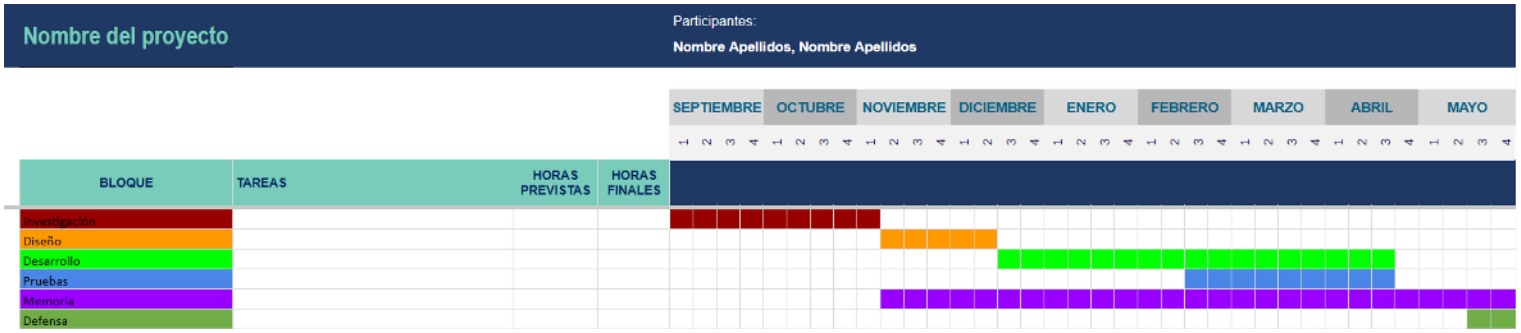
Una vez implementada la funcionalidad, se ha verificado su correcto funcionamiento. En caso de que el resultado haya sido satisfactorio, se ha procedido a integrar en el desarrollo principal del juego. Por el contrario, si se detectaban errores o comportamientos no deseados, se revisaba el código y los elementos implicados, repitiendo el proceso hasta lograr el resultado esperado.

Este proceso se ha aplicado de manera individual a cada una de las funcionalidades del proyecto, con el objetivo de centrarse en tareas pequeñas y manejables. De este modo, se ha buscado minimizar la aparición de errores y facilitar su localización, evitando abordar múltiples elementos complejos de forma simultánea que dificultan identificar el origen de los fallos.

Finalmente, cada funcionalidad validada se ha probado en el entorno real del juego para comprobar su comportamiento en conjunto con el resto de sistemas.

Esta metodología ha permitido avanzar de forma progresiva, detectar errores de manera temprana y asegurar la correcta integración de cada elemento desarrollado.

2.3 Planificación



3. Análisis

3.1 Casos de uso

Código	Nombre del caso de uso	Actor principal	Descripción	Resultado esperado
CU1	Iniciar el juego	Usuario	El usuario puede iniciar sin problemas el juego	El usuario inicia el juego en el menú principal y listo para jugar
CU2	Modificar Opciones	Usuario	El usuario puede ir a la sección de opciones del menú principal y modificarlas a su gusto como podría ser el volumen de música y efectos, pantalla completa…	Se puede seleccionar las opciones necesarias para ajustarse a cada jugador
CU3	Iniciar partida nueva	Usuario	El usuario inicia una partida nueva, lo cual significa que empieza de cero y desde el primer nivel	El jugador comienza desde el principio
CU4	Continuar partida	Usuario	El usuario con una partida comenzada podrá desde el menú principal volver al nivel de donde lo había dejado por última vez	La opción de continuar haga que el jugador empiece en el último nivel que se quedó
CU5	Avanzar al siguiente nivel	Usuario	Una vez cumplidos los requisitos necesarios para completar el nivel al jugador se le dará la opción de avanzar al siguiente nivel o de volver al menú principal si lo desea.	El jugador selecciona siguiente nivel y cambia al siguiente nivel, en caso contrario (selecciona volver al menú) volverá al menú principal pero al continuar partida empieza en el nuevo nivel

3.2 Requisitos funcionales

RF1	El juego debe iniciar desde 0 si es la primera vez que se inicia, esto implica que en el menú principal solamente puedas abrir el apartado de opciones y comenzar nueva partida, evidentemente la opción de salir del juego también estará habilitada.
RF2	Cada partida nueva implica que el nivel del personaje tanto sus estadísticas son las *stats base* (estadísticas predefinidas) y que el nivel en el que comienza es el 1.
RF3	Los enemigos aparecerán de forma aleatoria por todo el mapa y se dividirán por oleadas a modo que hasta que no elimines a todos los enemigos de una oleada no van a aparecer la siguiente.
RF4	El jugador atacará de forma automática en cuanto un enemigo entre en su rango de alcance (se puede modificar a lo largo de la partida en base a las mejoras). Una vez dentro se verá como lanza un proyectil (o múltiples si se tiene la mejora) al enemigo.
RF5	Al eliminar un enemigo el personaje conseguirá experiencia (*XP)* que cuando tenga la suficiente subirá un nivel, permitiendo al jugador seleccionar una mejora que te dará un beneficio a cambio de una consecuencia.

3.3 Requisitos no funcionales

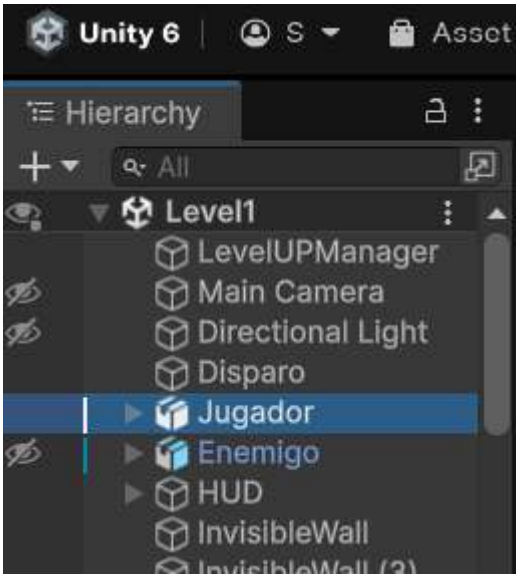
RNF1	El juego debe ser compatible con Windows, funcionando correctamente en las versiones más recientes y al menos en dos versiones anteriores de los sistemas operativos.
RNF2	La interfaz del juego debe ser intuitiva y accesible, permitiendo que un nuevo jugador entienda cómo jugar sin necesidad de tutoriales extensos. Los menús y opciones deberán ser fácilmente navegables.
RNF3	El juego deberá soportar múltiples idiomas (por ejemplo, español e inglés) para llegar a un público más amplio, y facilitar la incorporación de más idiomas en futuras actualizaciones.
RNF4	El juego debe funcionar de manera estable en las plataformas soportadas, manteniendo un rendimiento fluido (mínimo 60 FPS) incluso en situaciones de alta carga gráfica, como cuando se generan grandes cantidades de enemigos o efectos visuales.

3.4 Análisis de alternativas tecnológicas

Para la realización de este proyecto se ha llevado a cabo una investigación centrada principalmente en los entornos de desarrollo de *Godot* y *Unity*. Ambos motores presentan características similares en cuanto a las funcionalidades que ofrecen para el desarrollo de videojuegos. No obstante, tras el análisis realizado, se ha considerado que uno de ellos se ajusta mejor a las necesidades del proyecto.

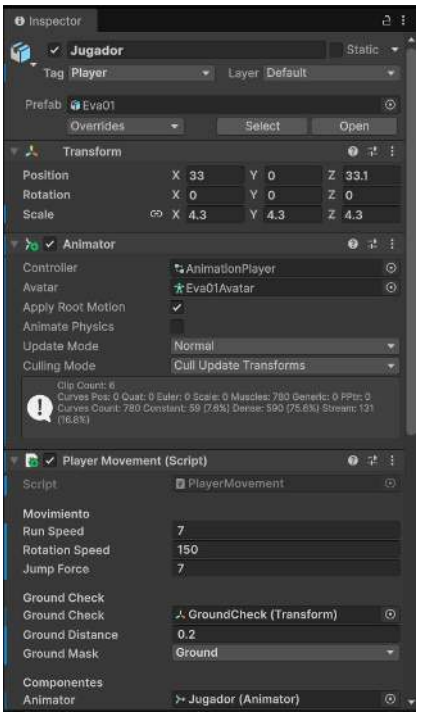
Durante el proceso de investigación se observó que Godot destaca especialmente en el desarrollo de videojuegos en dos dimensiones (2D), mientras que *Unity* ofrece un rendimiento y unas herramientas más avanzadas para el desarrollo en tres dimensiones (3D). En base a estos criterios, se optó por utilizar *Unity* como motor principal.

Además, la elección de *Unity* también se fundamenta en la organización de su interfaz y en la forma en que gestiona los elementos dentro de un proyecto. Este motor permite una clara separación entre los objetos de la escena y los componentes asociados a cada uno de ellos (como scripts, físicas o elementos visuales), lo que facilita tanto el desarrollo como la comprensión del proyecto.



Por un lado, en la jerarquía de la escena se pueden observar los distintos objetos que componen el juego. En este apartado predominan los elementos visuales, aunque no todos los objetos tienen necesariamente una representación gráfica, ya que en algunos casos es necesario crear objetos que contengan únicamente scripts para gestionar determinadas funcionalidades.

Por otro lado, Unity dispone de un panel de inspección (Inspector), donde se configuran todos los componentes asociados a cada objeto. Es en este espacio donde se añaden los elementos necesarios para la lógica del juego, como scripts, sistemas de colisión y otros comportamientos que definen la interacción dentro del entorno.



Asimismo, *Unity* ofrece una amplia variedad de recursos (*assets*) a través de su tienda oficial, conocida como *Asset Store*. En ella es posible encontrar tanto contenidos gratuitos como de pago, los cuales están diseñados para integrarse correctamente en el entorno de desarrollo, garantizando así su funcionalidad y compatibilidad.

Esta disponibilidad de recursos supone una ventaja significativa, ya que facilita y agiliza el proceso de desarrollo al permitir reutilizar elementos ya creados, como modelos, scripts o materiales.

Por el contrario, *Godot* no dispone de una plataforma equivalente con el mismo nivel de integración y variedad, por lo que en muchos casos es necesario buscar recursos externos o desarrollarlos de forma propia.

4. Diseño

4.1 Arquitectura del sistema

La arquitectura del sistema define la estructura general del proyecto, especificando los distintos componentes que lo forman, sus responsabilidades y la forma en que se relacionan entre sí. En este caso, el sistema se ha desarrollado utilizando el motor Unity, el cual propone una arquitectura basada en objetos (GameObjects) y componentes (Component-Based Architecture).

El sistema se organiza en varios bloques principales. En primer lugar, se encuentran las **escenas**, que actúan como contenedores donde se desarrolla la acción del juego. Cada escena incluye diferentes objetos (GameObjects) que representan elementos del entorno, personajes o elementos interactivos.

Cada uno de estos objetos no posee funcionalidad por sí mismo, sino que esta se define mediante componentes que se les añaden. Estos componentes pueden ser de distintos tipos: visuales (modelos, materiales), físicos (colisiones, rigidbodies) o lógicos (scripts). Esta separación permite que un mismo objeto pueda combinar diferentes comportamientos de forma modular.

La lógica del sistema se encuentra principalmente en los scripts, ubicados en la carpeta correspondiente. Estos scripts se encargan de gestionar el comportamiento de los objetos, la interacción entre ellos y las reglas del juego. De esta forma, los scripts actúan como el núcleo funcional del sistema.

Por otro lado, los recursos del proyecto (modelos 3D, animaciones, texturas, sonidos, etc.) se organizan en carpetas específicas dentro del proyecto. Estos recursos son utilizados por los objetos de las escenas a través de los componentes, estableciendo así una relación clara entre datos (assets) y comportamiento (scripts).

La comunicación entre los distintos elementos del sistema se produce principalmente a través de la interacción entre componentes dentro de un mismo objeto o entre distintos objetos de la escena. Por ejemplo, un script puede acceder a otros componentes del mismo objeto o detectar colisiones con otros objetos para desencadenar eventos.

Esta organización se ha elegido por su flexibilidad y escalabilidad, ya que permite modificar o ampliar el comportamiento del sistema sin necesidad de alterar su estructura global.

Además, el enfoque basado en componentes facilita la reutilización de código y recursos, así como una mayor claridad en la separación de responsabilidades.

En conjunto, la arquitectura adoptada permite mantener una estructura clara, modular y fácilmente ampliable, adecuada para el desarrollo de videojuegos dentro del entorno de Unity.

4.2 Modelo de datos

Mi proyecto no usa una arquitectura de base de datos, pero aun así hay interacción entre elementos, como puede ser el Jugador y el Enemigo. A continuación dejo una tabla de las más importantes:

Entidad	**Descripción**	**Atributos**
Jugador	Representa la jugador y su progreso en el juego	- Puntos de vida (HP) - Nivel del jugador - Recompensas obtenidas - XP obtenida para subir de nivel
Enemigo	Representa a los enemigos que atacarán al jugador	- Tipo - Salud - Daño - Velocidad - XP
Oleada	Representa las oleadas de enemigos que debe sobrevivir el jugador	- Número de enemigos - Tiempo de aparición - Tipo de enemigos - Dificultad - Intervalo entre oleadas
Mejora	Elementos que el jugador obtiene o mejora	- Efecto - Nivel de mejora

Enemigo → Jugador

Es la interacción más directa. Cuando un enemigo colisiona físicamente con el jugador, *OnCollisionEnter* en *EnemyController.cs* llama a *RecibirGolpe(1)* en *PlayerMovement.cs*, reduciendo la vida del jugador en 1. Si la vida llega a 0 se activa Morir(), que llama a *GameManager.GameOver()* mostrando la pantalla de Game Over. En sentido contrario, cuando el jugador dispara un proyectil y este impacta al enemigo, Proyectil.cs llama a *TakeDamage()* en *EnemyController.cs* reduciendo su salud.

Enemigo → Oleada

Cuando un enemigo muere, *Die()* en *EnemyController.cs* llama a *EnemyDefeated()* en *EnemySpawner.cs*, decrementando el contador *enemigosVivos*. Cuando ese contador llega a 0 y todos los enemigos de la oleada han sido spawneados, el spawner lanza automáticamente la siguiente oleada. Si ya se han completado todas las oleadas, se llama a *WaveUIManager.MostrarFinDeNivel()*.

Oleada → Jugador

Cada vez que un enemigo muere otorga 25 XP al jugador mediante *GanarXP(25f)* en *PlayerMovement.cs*. Cuando la XP acumulada supera *xpParaSiguienteNivel*, el jugador sube de nivel, se cura un 25% de su vida máxima, y se dispara la interacción con las mejoras. La oleada también influye indirectamente en la presión sobre el jugador: más enemigos vivos significa más colisiones posibles y más daño recibido.

Mejora → Jugador

Al subir de nivel, *LevelUpManager.MostrarMejoras()* pausa el juego mediante PauseManager *y presenta dos mejoras aleatorias del pool de 8. Al elegir una, se modifica* directamente un atributo de *PlayerMovement* — por ejemplo aumentar *attackDamage*, reducir *attackCooldown*, o cambiar *runSpeed* — siempre con una penalización en otro atributo. Estas modificaciones son permanentes durante la partida y se guardan en PlayerStats al completar el nivel, por lo que se acumulan a lo largo de las escenas.

4.3 Diseño de interfaz

La interfaz del juego sigue un diseño minimalista que mantiene la pantalla despejada durante el combate, con los elementos de HUD distribuidos en las esquinas para no obstruir la acción central.

Durante el juego (imagen 1) hay tres elementos visibles. En la esquina superior izquierda aparece el nivel del jugador en azul y debajo dos barras horizontales: la roja representa la

vida y la azul la experiencia, permitiendo al jugador monitorizar su estado de un vistazo sin necesidad de números. En la esquina superior derecha se muestra el indicador de oleada en formato "Wave 1 | 5" con el número actual en amarillo y el total en gris, informando del progreso en la partida.

Durante el level up (imagen 2) el juego se pausa y aparece un panel central semitransparente oscuro que no bloquea completamente el escenario. El título "¡ LEVEL UP !" en amarillo llama la atención, seguido del subtítulo "Chose one upgrade:" en gris. Las dos mejoras se presentan como tarjetas lado a lado con el nombre en amarillo negrita y la descripción del efecto en blanco, haciendo fácil comparar las opciones antes de elegir. El diseño de las tarjetas es deliberadamente simple para que el jugador pueda leer y decidir rápidamente sin perder el contexto visual del nivel donde se encuentra.

En general la interfaz prioriza la legibilidad con texto en negrita sobre fondos oscuros o semitransparentes, y usa el amarillo como color de énfasis para los elementos más importantes.

5. Desarrollo

5.1 Estructura del proyecto

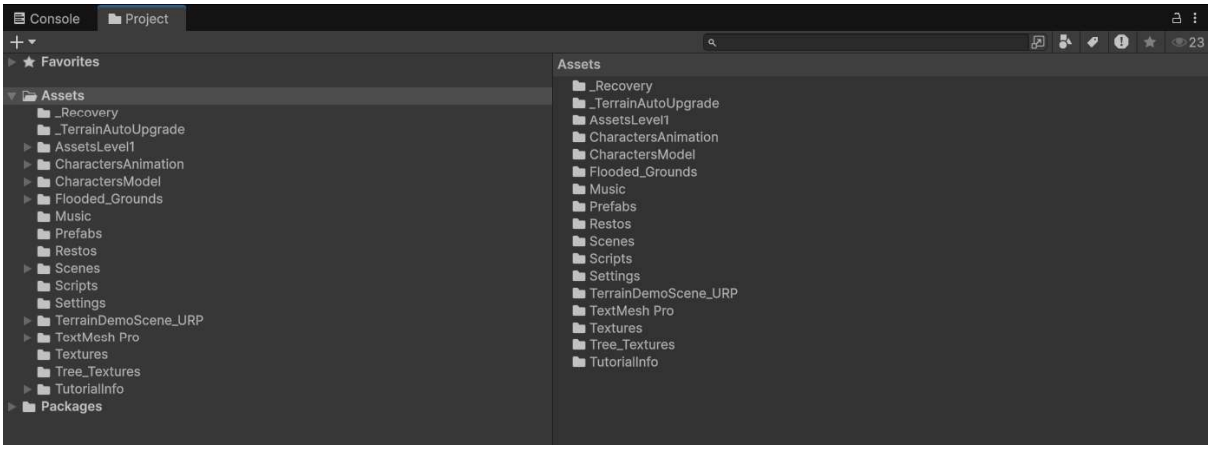
La arquitectura del proyecto se basa en un sistema de organización jerárquica mediante carpetas, siguiendo la estructura que proporciona por defecto *Unity*. Este enfoque permite mantener una correcta gestión de los recursos y facilita tanto el desarrollo como el mantenimiento del proyecto a medida que aumenta su complejidad.

En este caso, se ha optado por una organización modular, en la que los distintos elementos del proyecto se agrupan en carpetas específicas según su funcionalidad o naturaleza. De este modo, se consigue una clara separación entre los diferentes tipos de recursos, lo que mejora la legibilidad y el acceso a los mismos.

Tal y como se puede observar en la estructura del proyecto, existen carpetas destinadas a almacenar scripts (lógica del juego), escenas, *prefabs*, recursos multimedia como música y texturas, así como modelos y animaciones de personajes. También se incluyen directorios específicos para configuraciones del sistema y paquetes adicionales utilizados por el motor.

Además, esta organización permite diferenciar entre recursos propios del proyecto y aquellos importados o generados automáticamente por *Unity*, como puede ser el caso de determinadas herramientas o paquetes integrados. Esta separación resulta especialmente útil para evitar conflictos, mantener el orden y agilizar futuras modificaciones o ampliaciones del proyecto.

En conjunto, esta estructura basada en carpetas no solo responde a las recomendaciones del propio entorno de desarrollo, sino que también constituye una buena práctica en la gestión de proyectos, ya que facilita el trabajo individual y colaborativo, así como la escalabilidad del sistema.



5.2 Implementación de funcionalidades

5.2.1 Control del jugador

El script *PlayerMovement* desarrollado en Unity se encarga de gestionar tanto el movimiento del jugador como otros sistemas clave, como la vida, la experiencia y el ataque automático. A continuación, se describen los principales métodos y su aplicación dentro del funcionamiento del juego.

Método Start()

Este método se ejecuta al inicio de la escena y tiene como objetivo inicializar los componentes y variables necesarias. En él se obtiene la referencia al Rigidbody del jugador, así como a otros sistemas globales como el gestor de nivel (*LevelUpManager*) y el gestor del juego (GameManager).

Además, se aplican las estadísticas iniciales del jugador a través de una instancia externa (PlayerStats). Por último, se inicializa el temporizador de ataque (*attackTimer*) con un valor cercano al máximo, permitiendo que el jugador pueda atacar casi inmediatamente al comenzar la partida.

Aplicación en el juego: prepara todos los sistemas necesarios para que el jugador funcione correctamente desde el inicio.

Método Update()

Se ejecuta en cada frame y gestiona la lógica principal del jugador. En este método se recogen las entradas del usuario (movimiento horizontal y vertical) y se actualizan las animaciones en función de estos valores y de si el jugador está en el suelo.

También controla el sistema de ataque automático. Para ello:

- Incrementa un temporizador independiente del tiempo del juego (unscaledDeltaTime).
- Comprueba si hay enemigos dentro de un radio determinado.
- Selecciona el enemigo más cercano mediante otro método auxiliar.

- Rota al jugador hacia dicho enemigo.
- Dispara un proyectil.

Aplicación en el juego: gestiona la interacción en tiempo real, incluyendo movimiento, animaciones y combate automático.

Método FixedUpdate()

Este método se ejecuta a intervalos fijos y está orientado a la física del juego. Se encarga de aplicar:

- La rotación del jugador en función de la entrada horizontal.
- El desplazamiento hacia adelante según la entrada vertical.

Utiliza el Rigidbody para garantizar un comportamiento físico estable.

Aplicación en el juego: controla el movimiento del personaje de forma fluida y coherente con el sistema de físicas.

Método RecibirGolpe(int daño)

Este método reduce la vida del jugador cuando recibe daño. Se asegura de que la vida no sea inferior a cero y, en caso de agotarse, llama al método Morir().

Aplicación en el juego: gestiona el sistema de daño y supervivencia del jugador.

Método Morir()

Se ejecuta cuando la vida del jugador llega a cero. Cambia el estado del jugador a “muerto”, reinicia las estadísticas y notifica al gestor del juego para activar el estado de Game Over.

Aplicación en el juego: define el comportamiento del sistema cuando el jugador pierde la partida.

Método GanarXP(float cantidad)

Permite aumentar la experiencia del jugador. Cuando se alcanza el umbral necesario:

- Se incrementa el nivel.
- Se aumenta la experiencia necesaria para el siguiente nivel.
- Se aplica una pequeña curación al jugador.
- Se muestra el sistema de mejoras mediante el *LevelUpManager*.

Aplicación en el juego: implementa la progresión del personaje y la mejora de habilidades.

Método DispararProyectil(Transform objetivo)

Se encarga de instanciar un proyectil dirigido hacia un enemigo concreto. Calcula la dirección del disparo y configura el proyectil con el daño correspondiente.

Aplicación en el juego: permite al jugador atacar a distancia de forma automática.

Método GetClosestEnemy(Collider[] enemigos)

Recorre una lista de enemigos detectados y devuelve el más cercano al jugador.

Aplicación en el juego: optimiza el sistema de ataque seleccionando siempre el objetivo más próximo.

5.2.2 Sistema de XP y niveles(Jugador)

Al derrotar enemigos, estos soltarán *XP* (puntos de experiencia), que el jugador recogerá automáticamente. La experiencia obtenida se irá acumulando hasta alcanzar la cantidad necesaria para subir de nivel. Cada vez que el jugador aumente de nivel, la cantidad de XP requerida para el siguiente incrementará en un factor de 1,4.

Al subir de nivel, aparecerán dos mejoras aleatorias entre las siguientes opciones, de las cuales el jugador deberá escoger una para potenciar a su personaje:

- ***Fuerza Bruta:*** aumenta el daño del jugador a cambio de reducir su salud máxima.
- ***Espíritu Resistente:*** incrementa la salud máxima del jugador, pero disminuye su velocidad.
- ***Paso Veloz:*** aumenta la velocidad del jugador a costa de reducir su salud máxima.
- ***Alcance Lejano:*** incrementa el rango de ataque, pero reduce el daño infligido.
- ***Golpe Preciso:*** aumenta el daño de los ataques, aunque disminuye su alcance.
- **Frenesí:** reduce el *cooldown* (tiempo de recarga entre ataques) a cambio de disminuir la vida máxima.
- ***Cuerpo Templado:*** incrementa considerablemente la salud máxima del jugador, pero reduce su daño.
- ***Paso Sigiloso:*** aumenta la velocidad del jugador, aunque incrementa el *cooldown* de sus ataques.

5.2.3 Funcionamiento de los enemigos

Los enemigos funcionan gracias a dos *scripts*, uno que es su lógica de funcionamiento y el otro es el encargado de hacerlos aparecer de forma aleatoria en el mapa.

El script *EnemyController* controla el comportamiento de un enemigo en Unity, gestionando su movimiento, detección del jugador, patrulla, sistema de vida y muerte.

El enemigo posee dos comportamientos principales: patrulla aleatoria y persecución del jugador. Mientras el jugador se encuentre fuera del radio de detección, el enemigo se desplazará de forma aleatoria por el escenario, seleccionando periódicamente nuevos puntos de destino dentro de un rango definido. Cuando el jugador entra en el radio de detección, el enemigo abandona la patrulla y comienza a perseguirlo directamente.

El movimiento y la rotación del enemigo se realizan mediante un componente *Rigidbody*, permitiendo desplazamientos y giros suaves. Además, el script actualiza parámetros del *Animator* para reproducir correctamente las animaciones de movimiento y muerte.

El sistema de vida permite que el enemigo reciba daño mediante el método *TakeDamage()*. Cuando la vida llega a cero, se ejecuta el método *Die()*, que detiene el movimiento, activa la animación de muerte, otorga experiencia al jugador y notifica al generador de enemigos

(*EnemySpawner*) que el enemigo ha sido derrotado. Finalmente, el objeto se destruye tras un breve retraso.

Por último, el script detecta colisiones con el jugador mediante *OnCollisionEnter()*. Cuando ocurre una colisión, el enemigo inflige daño al jugador utilizando el método *RecibirGolpe()*.

El script *EnemySpawner* gestiona la generación de enemigos mediante un sistema de oleadas. Controla cuántas oleadas habrá, el número de enemigos por cada una, el tiempo entre oleadas y la velocidad de aparición de los enemigos.

Los enemigos se generan en posiciones aleatorias dentro de un área definida del escenario. El script también lleva un control de los enemigos vivos y detecta cuándo una oleada ha sido completada para iniciar automáticamente la siguiente.

Además, actualiza la interfaz del juego mediante *WaveUIManager* para mostrar el progreso de las oleadas y, al finalizar todas, indica que el nivel ha terminado.

Por último, incluye una representación visual en el editor de Unity para mostrar la zona donde aparecerán los enemigos.

5.3 Pruebas

Las pruebas realizadas durante el desarrollo del proyecto se llevaron a cabo de forma progresiva y continua. En primer lugar, tras implementar nuevas funcionalidades, se realizaban pruebas directas jugando al proyecto con el objetivo de detectar errores, fallos de funcionamiento o posibles aspectos a mejorar.

Posteriormente, se efectuaban partidas completas comenzando desde cero, simulando la experiencia de un jugador nuevo. Esto permitía comprobar que las mecánicas ya implementadas continuaban funcionando correctamente y que no aparecían errores derivados de las nuevas modificaciones.

Además, se solicitaron opiniones externas sobre la interfaz del juego a personas ajenas al mundo de los videojuegos, con el fin de evaluar si esta resultaba intuitiva, clara y fácil de utilizar.

Por último, dichas personas realizaron partidas de prueba para recopilar feedback sobre la experiencia de juego, lo que permitió identificar posibles mejoras tanto en la jugabilidad como en la interfaz y el equilibrio general del proyecto.

6. Conclusiones

6.1 Conclusiones generales

El proyecto ha logrado cumplir el objetivo planteado desde el inicio: desarrollar un videojuego inspirado en *Vampire Survivors* utilizando Unity y el lenguaje de programación C#, el cual era completamente desconocido para mí antes de comenzar el desarrollo.

En cuanto al resultado final del videojuego, se ha conseguido transmitir correctamente la esencia del género, ofreciendo una experiencia basada en acción constante y progresión continua del personaje a medida que avanza la partida. La implementación de las animaciones resulta satisfactoria, ya que aporta fluidez al movimiento y cumple adecuadamente a nivel visual.

Respecto al desarrollo técnico, el proyecto ha alcanzado un resultado sólido, logrando una jugabilidad estable y funcional. Además, se han implementado correctamente diversas características adicionales, como el sistema de guardado, la posibilidad de continuar niveles y las mejoras progresivas del jugador, integrándose todas ellas de manera adecuada en la experiencia de juego.

La parte más compleja del proyecto ha sido, sin duda, el aprendizaje de todos los aspectos implicados en el desarrollo de un videojuego. Muchas tareas que inicialmente parecían sencillas, como el modelado de personajes o la implementación de animaciones, terminaron siendo más difíciles de lo esperado. Esto se debió especialmente a la necesidad de buscar recursos externos para optimizar tiempo, encontrando en numerosas ocasiones modelos incompatibles o animaciones que no funcionaban correctamente.

6.2 Consecución de objetivos

En general, se puede considerar que los objetivos planteados al inicio del proyecto se han cumplido de forma satisfactoria, a excepción de uno de ellos.

El objetivo principal, desarrollar un videojuego funcional en 3D y tercera persona con Unity que sea entretenido y accesible, se ha logrado. El juego es jugable de principio a fin y transmite correctamente la esencia del género.

En cuanto a los objetivos específicos:

- El sistema de control del jugador en tercera persona funciona correctamente, con movimiento fluido y ataque automático en área que se activa al detectar enemigos cercanos.
- La IA de los enemigos se ha implementado con dos de los tres comportamientos previstos: patrulla aleatoria para los normales y persecución activa para los acosadores. El enemigo jefe no llegó a implementarse por falta de tiempo, quedando como una mejora pendiente para futuras versiones del proyecto.
- El sistema de progresión funciona tal y como se planteó, con ganancia de XP, subida de nivel y selección de mejoras con consecuencias en las estadísticas del jugador.
- Los modelos y animaciones de terceros se han integrado correctamente, aunque este punto fue uno de los más costosos del desarrollo, ya que encontrar recursos compatibles requirió más tiempo del esperado.
- La estructura de niveles con oleadas progresivas se ha implementado y funciona de forma estable.
- El juego se ha entregado como un producto completo y jugable, con sistemas adicionales como el guardado de partida y la opción de continuar, que enriquecen el resultado final.

6.3 Valoración de la metodología y planificación

La forma de trabajo ha sido bastante satisfactoria, ya que durante el desarrollo he logrado implementar la mayoría de las funcionalidades planteadas inicialmente. No obstante, el proceso resultó más complejo de lo esperado debido a mi falta de experiencia en el ámbito del desarrollo de videojuegos y, especialmente, en el uso de tecnologías como Unity y el lenguaje de programación C#.

A pesar de estas dificultades, considero que el tiempo se ha aprovechado de manera eficiente, ya que cada reto ha servido como una oportunidad de aprendizaje y mejora. Gracias a ello, el proyecto ha evolucionado de forma positiva y ha permitido obtener un resultado sólido y funcional. Aunque todavía existen aspectos que podrían ampliarse o

perfeccionarse en el futuro, el producto final cumple correctamente con los objetivos propuestos y ofrece una experiencia cercana a una demo completa y bien estructurada.

6.4 Visión de futuro

De cara al futuro, me gustaría continuar desarrollando el proyecto e implementar la historia que tenía planteada desde el inicio, con el objetivo de aportar una mayor profundidad narrativa y enriquecer la experiencia del jugador. Considero que este aspecto puede dar más personalidad al juego y hacer que el usuario se sienta más involucrado durante la partida.

Además, sería interesante añadir un modo multijugador competitivo en el que varios jugadores puedan enfrentarse para conseguir la puntuación más alta. Una de las ideas principales sería incorporar un modo de juego basado en oleadas infinitas de enemigos, aumentando progresivamente la dificultad y fomentando tanto la competitividad como la rejugabilidad.

Por otro lado, también me gustaría mejorar el apartado sonoro del juego mediante la inclusión de efectos de sonido para acciones como el movimiento del personaje, los disparos, el daño recibido, la subida de nivel y otros eventos importantes. De esta forma, la experiencia sería mucho más inmersiva y dinámica, aportando una mayor sensación de interacción y calidad al producto final.

7. Glosario

Unity — Motor de desarrollo de videojuegos multiplataforma creado por Unity Technologies, que permite crear proyectos 2D y 3D mediante una interfaz visual y scripting en C#.

C# — Lenguaje de programación orientado a objetos desarrollado por Microsoft, utilizado en Unity como lenguaje principal para programar la lógica del juego.

GameObject — Elemento base de Unity que representa cualquier objeto dentro de una escena, al que se le pueden añadir componentes para definir su comportamiento.

Script — Archivo de código que define el comportamiento y la lógica de un objeto dentro del juego.

Prefab — Plantilla reutilizable de un GameObject en Unity que permite instanciar objetos con las mismas propiedades en distintos puntos del juego.

Rigidbody — Componente de Unity que aplica física real a un objeto, permitiendo que sea afectado por fuerzas, gravedad y colisiones.

Collider — Componente de Unity que define el área física de un objeto para detectar colisiones con otros elementos de la escena.

Animator — Componente de Unity que gestiona y controla las animaciones de un objeto mediante una máquina de estados.

Asset — Cualquier recurso utilizado en el proyecto, como modelos 3D, texturas, sonidos, scripts o animaciones.

Asset Store — Tienda oficial de Unity donde los desarrolladores pueden descargar recursos gratuitos o de pago para usar en sus proyectos.

HUD — Interfaz visual superpuesta en pantalla durante el juego que muestra información relevante al jugador, como la vida o la experiencia.

XP (Experiencia) — Puntos que el jugador acumula al derrotar enemigos y que permiten subir de nivel al alcanzar una cantidad determinada.

Cooldown — Tiempo de espera obligatorio entre dos usos consecutivos de una acción o habilidad.

Spawn / Spawner — Acción de generar un objeto o enemigo en el juego en una posición determinada. El Spawner es el sistema encargado de gestionarlo.

FPS (Frames Per Second) — Número de fotogramas que el juego renderiza por segundo, indicador del rendimiento y fluidez visual.

Indie — Término que hace referencia a videojuegos desarrollados de forma independiente, sin el respaldo de grandes compañías distribuidoras.

Inspector — Panel de Unity que muestra y permite configurar todos los componentes y propiedades del GameObject seleccionado.

Instanciar — Proceso de crear en tiempo de ejecución una copia de un objeto o prefab en la escena del juego.

Component-Based Architecture — Patrón de diseño usado en Unity donde los objetos adquieren funcionalidades a través de componentes independientes que se les añaden.

OnCollisionEnter — Método de Unity que se ejecuta automáticamente cuando un objeto con Rigidbody colisiona físicamente con otro objeto.

unscaledDeltaTime — Variable de Unity que representa el tiempo transcurrido entre frames independientemente de si el juego está en pausa o a cámara lenta.

8. Bibliografía

Canales de YouTube:
https://www.youtube.com/@KanarianDev

https://www.youtube.com/@LuisCanary

https://www.youtube.com/@devrychz

Assets y modelos:

https://assetstore.unity.com/?srsltid=AfmBOop-lhJq-jbmrUna75wqmupQgfMOghJe48OcCyeF8Do34-edrYN-

https://sketchfab.com/feed

https://www.mixamo.com/#/

Inteligencia artificial utilizadas como soporte:
https://claude.ai/new

https://chatgpt.com/

9. Anexos

Códigos:

https://github.com/Adri1228/Proyect-Berserk/tree/main/Scripts

Nota sobre el uso de inteligencia artificial

(En el contexto actual, tanto si has utilizado herramientas de inteligencia artificial como si no, es importante dejar constancia de ello de forma clara. Esta nota no tiene carácter penalizador: su objetivo es reflejar un uso consciente y responsable de las herramientas disponibles. En caso de haber utilizado IA, debes indicar de qué forma se ha utilizado y asegurarte de que todo el contenido ha sido comprendido, revisado y asumido por el autor. Puedes inspirarte en los ejemplos proporcionados en la guía, pero el texto debe reflejar tu uso real de la herramienta.)

Licencia