

PROYECTO VIDEOJUEGO IA

OBJETIVO GENERAL DEL VIDEOJUEGO

El objetivo del videojuego es que el jugador sobreviva el mayor número de rondas posibles dentro de un escenario cerrado mientras se enfrenta a oleadas de enemigos de tipo zombi. En cada ronda, los zombis aumentan en número y dificultad, obligando al jugador a mejorar su equipamiento y estrategia. El jugador obtiene puntos y dinero eliminando zombis, y puede utilizar estos recursos para comprar armas y mejoras situadas en distintas zonas del mapa. El juego finaliza cuando la vida del jugador llega a cero, momento en el cual se presenta una pantalla de fin de partida desde la que se puede reiniciar el juego o regresar al menú principal. El proyecto cumple así con los requisitos de disponer de un bucle de juego cerrado, un menú principal y una inteligencia artificial enemiga capaz de tomar decisiones para provocar la derrota del jugador.

HISTORIAS DE USUARIO Y SU ANÁLISIS

HU1. Acceder al menú principal

Historia de usuario:

Como jugador, quiero acceder a un menú principal al iniciar el juego, para disponer de un punto de entrada claro antes de jugar.

Objetivo:

Establecer una pantalla inicial que organice el flujo del juego.

Requisitos funcionales:

1. Debe existir una escena de menú principal.
2. El menú debe mostrar opciones básicas (jugar, salir).

Requisitos no funcionales:

1. La interfaz debe ser clara y fácil de entender.
2. El tiempo de carga al entrar al menú debe ser mínimo.

Tareas:

1. Crear la escena del menú principal.
2. Diseñar la interfaz principal.
3. Configurar la navegación inicial del juego.

HU2. Iniciar partida desde el menú

Historia de usuario:

Como jugador, quiero pulsar un botón “Jugar” para iniciar una partida, con el fin de acceder al juego desde el menú principal.

Objetivo:

Permitir que el jugador pueda comenzar la partida de manera directa.

Requisitos funcionales:

1. El botón “Jugar” debe cargar la escena del juego.
2. Debe iniciarse correctamente el bucle principal del juego.

Requisitos no funcionales:

1. El cambio de escena debe ser fluido.
2. El tiempo de carga debe ser razonable.

Tareas:

1. Programar el botón “Jugar”.
2. Configurar la carga de la escena de juego.
3. Validar que el juego inicia correctamente.

HU3. Salir del juego desde el menú

Historia de usuario:

Como jugador, quiero poder salir del juego desde el menú principal para cerrar la aplicación cuando lo desee.

Objetivo:

Dar al jugador control sobre la salida del juego.

Requisitos funcionales:

1. Debe existir un botón de salida.
2. Debe cerrarse la aplicación al pulsarlo.

Requisitos no funcionales:

1. El botón debe estar claramente identificado.

Tareas:

1. Crear el botón “Salir”.
2. Programar la función de abandono de la aplicación.

HU4. Regresar al menú al terminar la partida

Historia de usuario:

Como jugador, quiero volver al menú principal después de finalizar la partida, para iniciar otra o abandonar el juego.

Objetivo:

Cerrar el ciclo de juego regresando al menú.

Requisitos funcionales:

1. Debe existir una opción para regresar al menú.
2. Debe cargarse correctamente la escena del menú.

Requisitos no funcionales:

1. La transición debe ser rápida.

Tareas:

1. Programar el retorno al menú desde la pantalla final.
2. Enlazar la UI de fin de partida con el menú principal.

HU5. Mover al personaje

Historia de usuario:

Como jugador, quiero desplazarme libremente por el escenario para explorar y esquivar zombis.

Objetivo:

Establecer un sistema de movimiento fluido.

Requisitos funcionales:

1. El jugador debe poder moverse en todas direcciones.
2. La rotación debe responder al ratón.

Requisitos no funcionales:

1. El control debe ser fluido.
2. La velocidad debe ser constante y estable.

Tareas:

1. Programar el controlador de movimiento.
2. Configurar sensibilidad y cámara.
3. Realizar ajustes de físicas si son necesarios.

HU6. Disparar un arma básica

Historia de usuario:

Como jugador, quiero poder disparar un arma inicial para defenderme de los zombis.

Objetivo:

Permitir el ataque básico al enemigo.

Requisitos funcionales:

1. El arma inicial debe poder disparar.
2. El disparo debe detectarse mediante raycast.
3. El impacto debe dañar al enemigo.

Requisitos no funcionales:

1. El disparo debe sentirse responsivo.

Tareas:

1. Implementar el sistema de disparo.
2. Crear raycast y detección de impacto.
3. Conectar el daño con los zombis.

HU7. Usar munición y recargar

Historia de usuario:

Como jugador, quiero que mis disparos consuman munición y pueda recargar cuando se agote, para que el combate sea más realista.

Objetivo:

Controlar la disponibilidad de disparos.

Requisitos funcionales:

1. Cada disparo debe consumir munición.
2. El jugador debe poder recargar.
3. Debe mostrarse la munición en la interfaz.

Requisitos no funcionales:

1. La recarga debe tener una animación o transición coherente.

Tareas:

1. Crear el sistema de munición.
2. Implementar recarga.
3. Añadir los elementos de UI relacionados.

HU8. Sistema de vida del jugador

Historia de usuario:

Como jugador, quiero tener una barra de vida que disminuya al recibir daño, para saber cuándo estoy en peligro.

Objetivo:

Administrar la supervivencia del jugador.

Requisitos funcionales:

1. El jugador debe tener un valor de vida.
2. Al recibir daño debe disminuir.
3. La UI debe reflejarlo.

Requisitos no funcionales:

1. La actualización debe ser instantánea.

Tareas:

1. Implementar el componente de vida.
2. Conectar el daño recibido.
3. Crear la barra de vida.

HU9. Gestionar rondas de enemigos

Historia de usuario:

Como jugador, quiero que el juego esté dividido en rondas sucesivas para aumentar la dificultad progresivamente.

Objetivo:

Organizar el progreso del juego mediante rondas.

Requisitos funcionales:

1. Debe existir un contador de rondas.
2. Cada ronda genera un número de zombis determinado.
3. La ronda termina cuando no quedan zombis.

Requisitos no funcionales:

1. La progresión debe estar balanceada.

Tareas:

1. Crear el gestor de rondas.
2. Implementar inicio y fin de ronda.
3. Mostrar la ronda en la UI.

HU10. Escalar dificultad según ronda

Historia de usuario:

Como jugador, quiero que los zombis se vuelvan más fuertes cada ronda para mantener el desafío.

Objetivo:

Adaptar parámetros de los enemigos.

Requisitos funcionales:

1. La vida y cantidad de zombis debe escalar.

Requisitos no funcionales:

1. Evitar picos bruscos de dificultad.

Tareas:

1. Programar el escalado de estadísticas.
2. Ajustar valores para el balance.

HU11. Zombis con movimiento autónomo

Historia de usuario:

Como jugador, quiero que los zombis se desplacen por el mapa sin quedarse atascados para que representen una amenaza real.

Objetivo:

Crear movimiento autónomo para los enemigos.

Requisitos funcionales:

1. Los zombis deben usar NavMesh.
2. Deben moverse automáticamente hacia el jugador.

Requisitos no funcionales:

1. La navegación debe funcionar con varios zombis simultáneos.

Tareas:

1. Generar el NavMesh.
2. Programar movimiento con NavMeshAgent.

HU12. Perseguir al jugador

Historia de usuario:

Como jugador, quiero que los zombis me sigan cuando me detecten, para que el combate tenga tensión.

Objetivo:

Implementar comportamiento de persecución.

Requisitos funcionales:

1. El zombi debe detectar al jugador.
2. Debe comenzar la persecución cuando esté en rango.

Requisitos no funcionales:

1. El sistema debe reaccionar rápido.

Tareas:

1. Crear detección del jugador.
2. Cambiar al estado de persecución.

HU13. Ataque de zombis

Historia de usuario:

Como jugador, quiero que los zombis me ataquen cuando estoy cerca, para provocar riesgo de muerte.

Objetivo:

Permitir al enemigo infligir daño al jugador.

Requisitos funcionales:

1. Debe existir un rango de ataque.
2. Los ataques deben reducir la vida del jugador.

Requisitos no funcionales:

1. El ataque debe tener cadencia coherente.

Tareas:

1. Implementar el sistema de ataque.
2. Conectar daño con la vida del jugador.

HU14. Muerte de los zombis

Historia de usuario:

Como jugador, quiero que los zombis mueran al recibir suficiente daño, para poder progresar en las rondas.

Objetivo:

Gestionar correctamente la muerte del enemigo.

Requisitos funcionales:

1. Cada zombi debe tener vida.
2. Al llegar a cero debe morir y destruirse.

Requisitos no funcionales:

1. La eliminación debe ser limpia y sin errores.

Tareas:

1. Programar vida y muerte del zombi.
2. Gestionar destrucción y efectos visuales.

HU15. Recibir puntos al matar zombis

Historia de usuario:

Como jugador, quiero obtener puntos al matar zombis para medir mi rendimiento.

Objetivo:

Crear un sistema básico de puntuación.

Requisitos funcionales:

1. Se deben sumar puntos al matar un zombi.
2. La UI debe mostrar la puntuación.

Requisitos no funcionales:

1. Los puntos deben actualizarse sin retraso.

Tareas:

1. Crear gestor de puntuación.
2. Conectar muerte del zombi con el aumento de puntos.

HU16. Obtener dinero al eliminar zombis

Historia de usuario:

Como jugador, quiero ganar dinero por cada zombi derrotado para poder comprar mejoras.

Objetivo:

Implementar la economía interna del juego.

Requisitos funcionales:

1. Los zombis deben otorgar dinero.
2. La UI debe mostrarlo.

Requisitos no funcionales:

1. Las cantidades deben estar balanceadas.

Tareas:

1. Añadir componente de dinero.

2. Conectar las recompensas con la muerte del zombi.

HU17. Comprar armas en puntos del mapa

Historia de usuario:

Como jugador, quiero comprar armas en lugares específicos del mapa usando mi dinero para poder mejorar mi equipamiento.

Objetivo:

Permitir la progresión mediante compras.

Requisitos funcionales:

1. Deben existir puntos de compra.
2. Cada punto debe tener una arma con un coste.
3. Debe verificarse si el jugador tiene suficiente dinero.

Requisitos no funcionales:

1. Debe ser un sistema claro y fácil de entender.

Tareas:

1. Crear los puntos de compra.
2. Definir armas como objetos configurables.
3. Conectar compra con el sistema de dinero.

HU18. Recoger drops de los zombis

Historia de usuario:

Como jugador, quiero que algunos zombis suelten objetos especiales al morir, para obtener ventajas temporales.

Objetivo:

Introducir variedad en el combate.

Requisitos funcionales:

1. Debe existir probabilidad de drop.
2. Los objetos deben ser visibles en el suelo.

Requisitos no funcionales:

1. Deben estar bien diferenciados visualmente.

Tareas:

1. Crear prefabs de drops.

2. Implementar probabilidad de aparición.

HU19. Aplicar efectos al recoger drops

Historia de usuario:

Como jugador, quiero que al recoger un drop su efecto se aplique inmediatamente para obtener la ventaja correspondiente.

Objetivo:

Dar funcionalidad a los objetos especiales.

Requisitos funcionales:

1. Los drops deben activar su efecto al recogerse.
2. Deben existir al menos:
 - Munición
 - Curación
 - Bonus temporales

Requisitos no funcionales:

1. Los efectos deben ser claros y predecibles.

Tareas:

1. Programar efectos individuales.
2. Añadir interfaz para efectos temporales si es necesario.

HU20. Finalizar la partida al morir

Historia de usuario:

Como jugador, quiero que la partida termine cuando mi vida llega a cero para cerrar el ciclo de juego.

Objetivo:

Detectar el fin de partida.

Requisitos funcionales:

1. Debe detectarse la muerte del jugador.
2. Debe detenerse el juego.

Requisitos no funcionales:

1. La detección debe ser inmediata.

Tareas:

1. Implementar detección de muerte.
2. Detener enemigos y acciones.

HU21. Ver la pantalla final con resultados

Historia de usuario:

Como jugador, quiero ver una pantalla final con mi puntuación y ronda para conocer mi rendimiento.

Objetivo:

Mostrar resultados del jugador.

Requisitos funcionales:

1. La pantalla final debe aparecer al morir.
2. Debe mostrar ronda y puntuación.

Requisitos no funcionales:

1. La UI debe ser clara.

Tareas:

1. Crear la pantalla de fin de partida.
2. Mostrar los datos necesarios.

HU22. Reiniciar o volver al menú tras morir

Historia de usuario:

Como jugador, quiero tener la opción de reiniciar la partida o regresar al menú principal después de morir.

Objetivo:

Permitir reiniciar el ciclo de juego.

Requisitos funcionales:

1. La pantalla final debe incluir botones de reinicio y menú.
2. Debe cargar correctamente la escena correspondiente.

Requisitos no funcionales:

1. La transición debe ser rápida y estable.

Tareas:

1. Implementar los botones.
2. Conectar cada botón con su escena.

PATRONES DE DISEÑO

Patrón Facade

El GameManager actúa como un punto de coordinación para el resto de sistemas. Esta decisión está alineada con el patrón Facade, ya que centraliza el control del flujo de la partida y simplifica la interacción entre subsistemas como las rondas, la interfaz o la puntuación.

Patrón Singleton

Los principales gestores pueden implementarse como instancias únicas, lo cual sigue la idea del patrón Singleton. Esto garantiza que solo exista un controlador de interfaz, de puntuación o de flujo de partida durante el juego y evita conflictos entre múltiples instancias.

Patrón Factory Method

El proceso de creación de zombis y la posible generación de objetos especiales sigue la lógica del patrón Factory Method. La responsabilidad de instanciar los enemigos o los drops se concentra en clases dedicadas, lo que simplifica su gestión y permite modificar o ampliar tipos sin alterar código externo.

Patrón Strategy

Tanto el sistema de armas como el sistema de objetos recogibles aplican la idea del patrón Strategy. Cada arma y cada tipo de drop define su propio comportamiento, pero todos comparten una misma estructura. Esto permite intercambiar comportamientos sin modificar el resto del sistema.

Patrón State

La inteligencia artificial del zombi está planteada para utilizar el patrón State, ya que identifica distintos modos de comportamiento, como búsqueda, persecución, ataque o muerte. Separar esta lógica en estados facilita mantener y ampliar la IA sin multiplicar condiciones dentro de una sola clase. El GameManager también gestiona estados globales de la partida, lo que encaja con el mismo patrón.

Patrón Observer

Este patrón se aplica especialmente en la comunicación entre la muerte de un zombi y los sistemas que reaccionan a ese evento, como el avance de rondas, la puntuación o la generación de objetos. Gracias a Observer, los zombis no necesitan conocer directamente a estos sistemas; simplemente emiten un aviso al que otros componentes están suscritos.