Universidad Pontificia Comillas

ICAI

iMAT Zombies

Paradigmas y técnicas de programación

Lydia Ruiz Martínez, Jorge Vančo Sampedro

PROYECTO FINAL

Curso 2024-2025

Índice

1. Propuesta de videojuego

1.1. Nombre

iMAT Zombies

1.2. Género

Survival/ Shooter

1.3. Descripción

Este juego de supervivenvia sitúa al jugador en un ambiente desafiante, donde hay zombies que acechan por todos lados. El jugador ha de enfrentarse a ellos, zombies que aparecen en grupos y se mueven hacia el jugador, que necesita frenar su avance y destruirlos usando para ello distintas estrategias y armas.

A medida que el jugador va eliminando zombies, puede recoger del suelo armas y trampas que le facilitan hacer frente a las oleadas que van apareciendo. Por un lado, el jugador emplea las armar para aniquilar a los zombies; por otro, las trampas le permiten ralentizar o detener temporalmente a los enemigos. Eventualmente aparece una niebla que reduce la visibilidad del jugador, aumenta la dificultad de localizar a los zombies y creat estrategias.

Además, hay un cofre que permite al jugador intercambiar puntos o kills acumulados por nuevas armas; de esta manera, puede recibir bonificaciones, como por ejemplo un coche bomba que permite limpiar el área de enemigos y ganar unos segundos de ventaja. Si el jugador tiene éxito en la estrategia y combate, podrá sobrevivir cada ola y enfrentarse a nuevos desafíos.

1.4. Objetivo

El principal objetivo es sobrevivir durante el mayor número de oleadas posibles de zombies, a los cuales hay que eliminar y así acumulando puntos que puede usarse para adquirir armas mediante el cofre. Hay que gestionar las armas adecuadas y otros recursos de forma estratégica para cada situación. También se han de activar trampas en el terreno para obtener ventaja táctica. En ocasiones aparece una niebla que dificulta el juego y a la que hay que adaptarse. Además, hay que lograr rachas para poder intercambiar los puntos/ kills por armas que hay en el cofre.

2. Diseño de arquitectura

2.1. Diagrama UML

El enlace al diagrama UML en el siguiente: Diagrama UML

La arquitectura propuesta se ha dividido en tres capas principales: **Manager Layer**, **Scene Manager** y **Agent Manager**. Esta organización permite una clara separación de responsabilidades y mejora tanto la escalabilidad como el mantenimiento del sistema.

- **Manager Layer:**
  - En esta capa se encuentra el **GameManager**, que actúa como el núcleo de control del sistema, supervisando y coordinando los diferentes componentes del juego.
  - El *GameManager* tiene una relación de **agregación** con los siguientes elementos:
    - **DataManager**: Encargado de gestionar los datos persistentes del juego.
    - **RoundManager**: Controla la lógica de las rondas en el juego.
    - **MenuManager**: Gestiona las interfaces de menú. Este componente, mediante una relación de **composición**, contiene tanto el menú principal como el menú de compra de items.
  - **AudioManager**: Responsable de la gestión de efectos de sonido en el juego. Posee una relación de **composición** con los efectos de sonido específicos.
  - **SceneManager**: Vinculado por una relación de composición con las distintas escenas del juego, como:
    - **GameScene**: La escena principal donde ocurre la acción del juego.
    - **MainMenuScene**: El menú inicial donde se encuentran las opciones de inicio y configuración del juego como la dificultad.
    - **BuyingMenu**: La interfaz de compra de objetos y otras mejoras para el jugador.
  - **UIManager**: Maneja los elementos de la interfaz de usuario y mantiene una relación de **composición** con componentes como la barra de vida, el contador de rondas y el inventario.

  Los *managers* se implementarán como **singletons**, permitiendo acceder a sus datos y métodos desde cualquier parte del sistema sin la necesidad de instanciar cada *manager* en otras clases. Por ejemplo, el *SoundManager* contiene referencias a todos los sonidos disponibles en el juego, de modo que cualquier objeto con sonido puede acceder a los métodos de esta instancia sin requerir atributos adicionales. Esta estructura permite una relación de **asociación**, en

la cual los objetos interactúan con los *managers* sin generar dependencias adicionales. Se propone implementar efectos de sonido cada vez que se dispare un arma de fuego o se destruya un zombie.

- **Scene Layer:**
  - En esta capa se encuentran componentes como la **Streak**, que mantiene el seguimiento de las rachas del jugador, y el **Inventory**, que gestiona los objetos y mejoras disponibles para el personaje.
  - La **Interfaz IUsable** es una interfaz que se utiliza para definir los objetos que pueden ser utilizados dentro del juego. Esta interfaz mantiene una relación de **agregación** con **Inventory** y **BuyingBox**. El **Inventory** contiene los objetos que el jugador puede usar, mientras que el **BuyingBox** se encarga de la gestión de compra de nuevos objetos durante el juego.
  - La interfaz **IUsable** es implementada por tres tipos de **Weapons**: **Mele**, **FireWeapon** y **Grenade**. Cada tipo de arma tiene características únicas: las **Mele** tienen un alcance, las **FireWeapon** tienen un *rate of fire* (velocidad de disparo), y las **Grenade** tienen un radio de daño determinado que afecta a los enemigos en un área.
  - Para crear estos objetos, se ha implementado una **Factory** que permite generar de manera dinámica los diferentes tipos de armas (**Grenade**, **Mele** y **FireWeapon**). Esta solución de patrón de diseño proporciona flexibilidad y facilidad de expansión en caso de que se quieran agregar más tipos de objetos en el futuro.
- **Agent Layer:**
  - En esta capa se encuentra el **Character**, el cual tiene una relación de **asociación** con los eventos del juego. El **Character** también tiene una relación de **composición** con el **Inventory**, ya que el personaje mantiene su propio inventario de objetos y mejoras. Además, el **Character** está relacionado por **agregación** con las **Rachas** (streaks), las cuales hacen seguimiento de las rachas de puntuación o logros del jugador a lo largo del juego.
  - Los **Zombies** se dividen en tres tipos diferentes: **FastZombie**, **NormalZombie** y **SlowZombie**. Cada tipo de zombie posee características únicas, tales como velocidad, daño y alcance, lo que determina su comportamiento y dificultad en el juego. Para la creación de estos zombi3s, se ha implementado un patrón de diseño **Factory**, lo que permite generar los distintos tipos de zombis de manera eficiente sin la necesidad de instanciarlos manualmente en cada lugar del código. Cabe destacar que, aunque inicialmente se consideró el uso del patrón de diseño **Flyweight** para la creación de los zombies, finalmente se optó por la implementación de la **Factory**. El patrón **Flyweight** es más comúnmente utilizado

en sistemas que gestionan un gran número de objetos similares, como partículas (miles), mientras que en nuestro caso solo existen tres tipos de zombis con características bien definidas.

2.2. Historias de usuario

- Como jugador, quiero matar a los zombies con las armas que aparecen en el suelo para no perder vidas.
  - **Objetivo:** El jugador necesita recoger las armas que se encuentran en el suelo para defenderse de los zombies y mantener sus vidas.
  - **Requisitos:**
    - Implementar una mecánica para que las armas sean recogidas al pasar cerca de ellas.
    - Añadir una función de ataque personalizada para cada arma (disparar, explotar o apuñalar).
    - Asignar una cantidad de daño a cada arma.
  - **Tareas:**
    - Crear el sistema de interacción con objetos en el suelo.
    - Desarrollar las mecánicas de combate de las armas.
- Como jugador, quiero recuperar vida al acabar cada ronda para tener más probabilidad de sobrevivir en cada ola.
  - **Objetivo:** Tras acabar la ronda, se le suma algo de vida al jugador para que tenga más posibilidades de sobrevivir la siguiente ola.
  - **Requisitos:**
    - Implementar una mecánica para que se sume algo de vida tras cada ronda.
    - Mostrar la vida del jugador en la pantalla.
  - **Tareas:**
    - Hacer el LifeBar de la interfaz.
    - Desarrollar las mecánicas de la vida del jugador.
- Como jugador, quiero tener un cofre con armas para comprar arsenal.
  - **Objetivo:** El jugador puede comprar armas en un cofre especial.
  - **Requisitos:**
    - El cofre contiene distintas armas que pueden ser compradas por el jugador.
    - Cada arma debe tener un precio.
  - **Tareas:**

    - Crear cofre de armas.
    - Asignar a cada arma un precio.
    - Desarrollar mecánica de compra de armas.

- Como jugador, quiero tener dinero para comprar arsenal en el cofre.

  - **Objetivo:** El jugador debe poder conseguir dinero al eliminar zombies o de otra forma similar para luego poder comprar mejores armas y equipamiento.
  - **Requisitos:**
    - El jugador debe poder conseguir dinero al eliminar zombies.
    - El jugador puede usar ese dinero para comprar equipamiento.
  - **Tareas:**
    - Implementar la mecánica de conseguir dinero.
    - Permitir al jugador gastarse ese dinero en el cofre.

- Como jugador, quiero tener puntos para activar rachas o para mejorar mis habilidades y herramientas.

  - **Objetivo:** El jugador necesita implemetar un sistema de puntación para activar rachas o mejorar su inventario, proporcionándole más opciones estratégicas a lo largo de la partida.
  - **Requisitos:**
    - Conseguir más puntos desbloquea más rachas.
    - Los puntos deben acumularse en función de las acciones realizadas en el juego, como eliminar zombies o completar rondas.
    - El sistema de puntos debe permitir la activación de rachas que otorguen beneficios temporales.
    - Los puntos deben ser visibles al usuario, permitiendo al jugador saber cuántos puntos ha acumulado en todo momento.
  - **Tareas:**
    - Crear un menú de compra dentro del BuyingMenu donde el jugador pueda usar los puntos acumulados para activar rachas.
    - Desarrollar un sistema de rachas que el jugador pueda activar con los puntos, ofreciendo beneficios estratégicos como aumento de poder o recuperación de salud.
    - Integrar el sistema de puntos con el Inventory y Character, de modo que las armas compradas o las rachas activadas se reflejen en el inventario del jugador.
    - Visualizar el total de puntos acumulados en la interfaz de usuario, asegurándose de que el jugador pueda ver la cantidad de puntos disponibles en todo momento.

    - Desarrollar las mecánicas de cada racha.
    - Asignar una cantidad de puntos necesarios para cada racha.

- Como zombie, quiero perseguir al jugador para atacarle y eliminarlo.

  - **Objetivo:** El zombie debe perseguir al jugador de manera autónoma, usando un sistema que permita detectar dónde está el jugador para poder atacarlo y hacer que pierda vidas.
  - **Requisitos:**
    - Los zombies deben detectar la posición del jugador y moverse hacia él de forma eficiente, utilizando IA para determinar su trayectoria
    - Los zombies deben tener diferentes comportamientos según su tipo (por ejemplo, velocidad de movimiento y alcance de ataque).
    - Los zombies deben atacar al jugador cuando estén lo suficientemente cerca, infligiendo daño al personaje del jugador.
    - Los zombies deben tener un sistema de vida para ser eliminados si reciben suficiente daño por parte del jugador.
  - **Tareas:**
    - Crear el sistema de interacción con objetos en el suelo.
    - Desarrollar las mecánicas de combate de las armas.

- Como partida, quiero tener un contador de rondas para contabilizar las rondas jugadas, de modo que el progreso de la partida se pueda medir.

  - **Objetivo:** Proporcionar una forma de seguir el progreso de la partida, ajustando la dinámica de los enemigos a medida que avanzan las rondas.
  - **Requisitos:**
    - El contador de rondas debe actualizarse automáticamente al final de cada ronda.
    - El juego debe aumentar su dificultad a medida que se completen las rondas (aumentando la velocidad o el número de zombies por ejemplo).
  - **Tareas:**
    - Implementar el **RoundManager** para gestionar el avance de las rondas.
    - implementar el RoundCounter para que el **UIManager** muestre en la escena la ronda actual.

- Como arma de fuego, quiero tener balas para poder disparar.

  - **Objetivo:** Las armas de fuego tienen que tener balas que puedan ser disparadas.

- **Requisitos:**
  - Cada arma de fuego debe poder disparar balas según su rate of fire.
- **Tareas:**
  - Implementar un BulletPool para las balas.
  - Permitir que el arma dispare las balas.