HooliganHavoc

Paradigmas y Técnicas de Programación: Proyecto final

Jorge Kindelán Navarro
Ignacio Queipo de Llano Pérez-Gascón

Introducción

En este documento se presenta una descripción detallada del desarrollo de este proyecto, desde su diseño inicial hasta el producto final. Además, se exponen los problemas enfrentados durante el proceso y los cambios realizados para resolverlos, justificando cada decisión tomada.

Enlace a GitHub: https://github.com/iqueipopg/HooliganHavoc

Diseño Inicial

En las etapas iniciales del proyecto, el objetivo principal fue conceptualizar un juego de acción y supervivencia basado en oleadas. Este diseño inicial se basó en los siguientes componentes clave:

1. **Jugador:**
    - Movimiento fluido controlado por el usuario mediante entradas del teclado.
    - Un sistema de vida representado de forma visual para reflejar el estado de salud del jugador.
    - Una mecánica de enfrentamiento directo contra los enemigos.
2. **Enemigos:**
    - Diseño de dos tipos de enemigos:
        - **Enemigo estándar:** Con un patrón de movimiento básico hacia el jugador.
        - **"Charger":** Un enemigo especializado que se preparaba antes de realizar una carga rápida hacia el jugador. Cuando el jugador se acerca demasiado, el Charger activaba un comportamiento único: primero carga su ataque, indicando visualmente que vas atacar, y luego realizaba una carga rápida hacia el jugador. Si logra alcanzarlo, lo arrastra durante un breve período antes de soltarlo en otra parte del mapa, infligiendo daño en el proceso.
    - Comportamientos diferenciados gestionados por un sistema de estados.
3. **Sistema de oleadas:**
    - Generación de oleadas progresivas, aumentando el número y dificultad de enemigos conforme avanzaba el juego.
    - Presentación de información al jugador, como el número de enemigos restantes en cada oleada.
4. **Sistema de armas:**
    - Incorporación de armas que el jugador podría obtener conforme avanzaba en el juego, con posiciones aleatorias alrededor del personaje.
    - Implementación de un sistema de proyectiles para atacar a los enemigos.
5. **Interfaz de usuario (UI):**
    - Diseño de una interfaz básica que incluía:
        - Pantalla de inicio para comenzar el juego.
        - Un HUD que mostrara la salud del jugador, la oleada actual y los enemigos restantes.
        - Una pantalla de "Game Over" para reiniciar o salir del juego.
6. **Cámara:**
    - Una cámara que seguía al jugador de manera suave, mejorando la experiencia de juego en mapas de mayor escala.

Problemas Enfrentados y Cambios Realizados

1. Movimiento del Jugador

- **Problema:**
  Inicialmente, se planificó un sistema de movimiento con animaciones complejas que representaran no solo la dirección del movimiento, sino también las transiciones entre estados.
- **Cambio:**
  Se simplificó el movimiento utilizando el sistema Input.GetAxisRaw, lo que permitió controlar direcciones básicas de movimiento.
  Las animaciones se limitaron a reflejar la velocidad y dirección con un enfoque minimalista, como se muestra en Player.cs.

2. Sistema de Oleadas

- **Problema:**
  El sistema inicial no gestionaba adecuadamente el aumento progresivo de dificultad. El incremento del número de enemigos por oleada no siempre era desafiante, y el código carecía de flexibilidad.
- **Cambio:**
  Se introdujo un cálculo dinámico del número de enemigos basado en un multiplicador ("enemigos = oleada * 5"), gestionado en WaveManager.cs.
  Además, se mejoró la transición entre oleadas para incluir una pausa que indicara al jugador el final de una oleada y el inicio de la siguiente.

3. Comportamiento de los Enemigos

- **Problema:**
  Los enemigos "Charger" no siempre se comportaban de forma consistente, lo que llevaba a errores como cargas repentinas o bloqueos cuando se preparaban para atacar.
- **Cambio:**
  Se implementó un sistema de estados para los enemigos, con variables como isCharging, isStunned e isPreparingCharge (Enemy.cs). Esto aseguró que las acciones de los enemigos se ejecutaran en el orden correcto.
  Se añadió un tiempo de preparación antes de cargar para permitir al jugador reaccionar. El comportamiento de arrastrar al jugador fue cuidadosamente calibrado para que no rompiera el flujo del juego.

4. Sistema de Armas

- **Problema:**
  Las armas aparecían en posiciones aleatorias, lo que a menudo resultaba en confusión visual y desorden en el juego.
- **Cambio:**
  Se definieron posiciones fijas alrededor del jugador para las armas adicionales (GunManager.cs), creando un sistema más claro y organizado. Además, se limitó el número de armas simultáneas para evitar la saturación visual.

5. Interfaz de Usuario (UI)

- **Problema:**
  La implementación de una barra de salud visual y otros elementos complejos del HUD no se priorizó inicialmente para enfocarse en garantizar la funcionalidad básica del juego.
- **Cambio**
  Se utilizó texto con TextMeshProUGUI para mostrar información clave, como la salud del jugador y los enemigos restantes (Player.cs, WaveManager.cs). Esto simplificó la implementación sin comprometer la claridad.

6. Gestor de la Cámara

- **Problema:**
  La cámara inicialmente era estática, lo que limitaba la experiencia del jugador en mapas grandes.
- **Cambio**
  Se implementó una cámara que sigue al jugador con suavidad, utilizando interpolación para evitar movimientos bruscos (CameraFollow.cs).

Diseño Final

El diseño final resultante del desarrollo logró mantener los elementos esenciales planificados inicialmente, al tiempo que se ajustó a las limitaciones técnicas. Las características clave del producto final incluyen:

- **Jugador:**
  - Movimiento fluido y responsivo.
  - Sistema de vida simplificado pero efectivo, representado mediante texto.
- **Enemigos:**
  - Comportamientos diferenciados bien definidos.
  - Incremento gradual de dificultad a través de oleadas.
- **Oleadas:**
  - Sistema progresivo con transiciones claras entre fases.
- **Armas:**
  - Posicionamiento fijo y progresión limitada para evitar saturación visual.
- **Interfaz de Usuario:**
  - Minimalista pero funcional, mostrando información clave como salud, oleadas y enemigos restantes.
- **Cámara:**
  - Seguimiento suave del jugador para mejorar la experiencia en el mapa.

Posibles Mejoras y Adiciones

1. **Nuevos tipos de enemigos:**
   - La estructura de los enemigos basada en sistemas de estados está preparada para integrar nuevos comportamientos. Por ejemplo, enemigos voladores o que ataquen a distancia.
2. **Más variedad de armas:**
   - El sistema de armas admite la incorporación de armas con características únicas, como proyectiles explosivos o con efectos de área.
3. **Niveles personalizados:**
   - Incorporar diferentes mapas con obstáculos o características específicas que añadan diversidad al juego.
4. **Mejoras en la interfaz de usuario:**
   - Añadir barras de salud gráficas, temporizadores visuales para las oleadas y notificaciones animadas.
5. **Sistema multijugador:**
   - Expandir el juego para incluir modos cooperativos o competitivos entre jugadores.

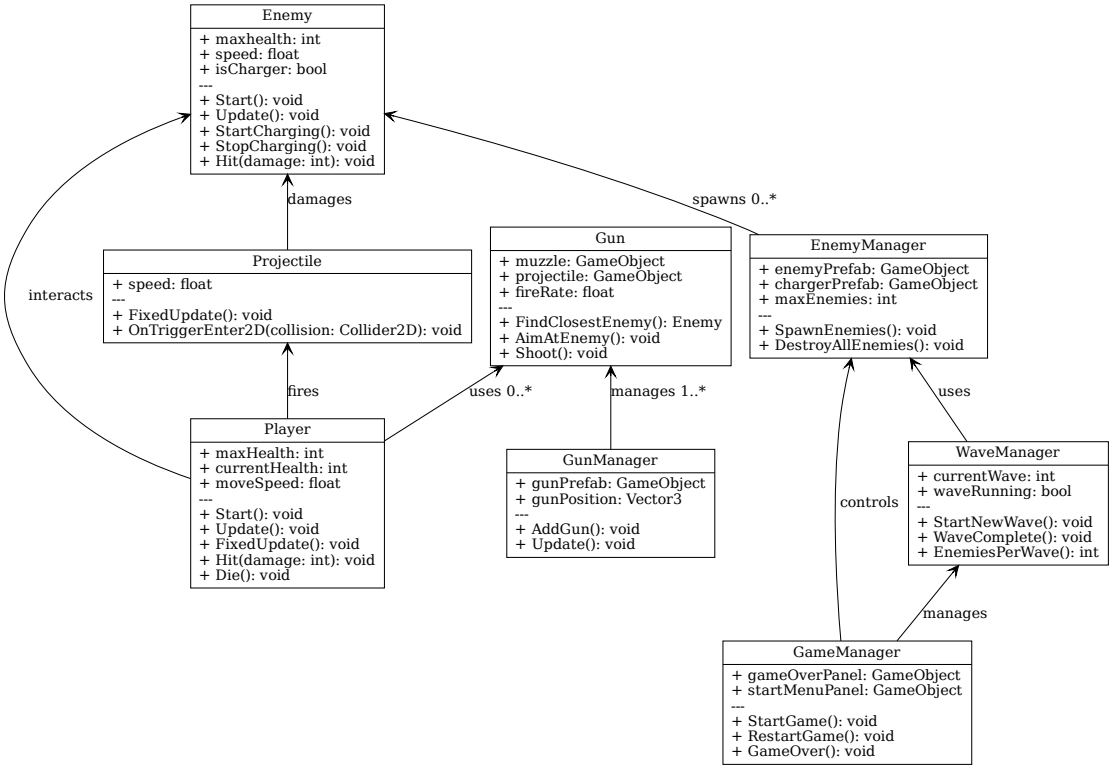
Conclusión

Este proyecto ha sido una experiencia enriquecedora que permitió aplicar conceptos clave de programación y desarrollo de videojuegos, como la modularidad del código, la implementación de sistemas de estados y la gestión dinámica de recursos.

Durante el desarrollo, priorizar funcionalidades básicas y garantizar un diseño escalable se convirtió en uno de los pilares fundamentales del enfoque adoptado. Este enfoque no solo permitió cumplir con los objetivos iniciales del proyecto de manera eficiente, sino que también estableció una base sólida para facilitar la incorporación de futuras mejoras. Además, fomentó un entorno idóneo para resolver los problemas surgidos durante el proceso mediante soluciones creativas y estructuradas.

A lo largo de este proyecto, hemos reforzado habilidades fundamentales en el diseño y desarrollo de sistemas interactivos, estableciendo una base sólida para emprender futuros proyectos más complejos. El proceso ha demostrado cómo un enfoque bien planificado y estructurado permite equilibrar la funcionalidad inmediata con la posibilidad de evolucionar y expandirse.

Diagrama UML final


Enemy
+ maxhealth: int
+ speed: float
+ isCharger: bool

+ Start(): void
+ Update(): void
+ StartCharging(): void
+ StopCharging(): void
+ Hit(damage: int): void
damages
spawns 0..*
Projectile
+ speed: float

+ FixedUpdate(): void
+ OnTriggerEnter2D(collision: Collider2D): void
interacts
Gun
+ muzzle: GameObject
+ projectile: GameObject
+ fireRate: float

+ FindClosestEnemy(): Enemy
+ AimAtEnemy(): void
+ Shoot(): void
EnemyManager
+ enemyPrefab: GameObject
+ chargerPrefab: GameObject
+ maxEnemies: int

+ SpawnEnemies(): void
+ DestroyAllEnemies(): void
fires
uses 0..*
manages 1..*
uses
Player
+ maxHealth: int
+ currentHealth: int
+ moveSpeed: float

+ Start(): void
+ Update(): void
+ FixedUpdate(): void
+ Hit(damage: int): void
+ Die(): void
GunManager
+ gunPrefab: GameObject
+ gunPosition: Vector3

+ AddGun(): void
+ Update(): void
WaveManager
+ currentWave: int
+ waveRunning: bool

+ StartNewWave(): void
+ WaveComplete(): void
+ EnemiesPerWave(): int
controls
manages
GameManager
+ gameOverPanel: GameObject
+ startMenuPanel: GameObject

+ StartGame(): void
+ RestartGame(): void
+ GameOver(): void