ZOMBIE ASSAULT 3D

Practica 2

Programación de Juegos

Juan Jiménez Serrano

Luis Miguel Pérez Martín

Índice

1. Primeras ideas y descripción del juego.

Desde que terminamos con el juego 2D y empezamos a trabajar con el juego 3D teníamos las ideas bastante claras en cuanto al tipo de juego que queríamos hacer.

La idea principal y final fué crear un juego de zombies por oleadas como los que encontramos en su día en Black Ops, que fué un juego muy jugado por nosotros.

Es un tipo de juego en el que sabíamos que encontraríamos ciertas facilidades a la hora de buscar información, assets y encima es un género en el que los algoritmos genéticos encajan muy bien.

2. Orden de trabajo.

Lo primero de todo fue empezar a buscar assets para los escenarios personajes y menús, dejar los menús, escenarios y personajes buscados e incorporados en el proyecto lo antes posible. Posteriormente trabajamos en sus scripts correspondientes y el último paso fue la creación del algoritmo genético.

3. Escenas

3.1. Escena principal.

La idea que tuvimos con las escenas era la de no hacer un menú 2D, queríamos en la medida de lo posible que nos encontrásemos con elementos 3D a pesar de estar en los menús. Encontramos dos zombies 3D y tres granadas a modo de decoración.

En el script de esta escena encontramos el código para pasar a la siguiente escena.

Script Asociado → PantallaInicio

3.2. Escena brillo.

En esta escena encontramos el sistema para ajustar el brillo a nuestro gusto, el cual guarda los valores que tengamos guardados en PlayerPrefs de sesiones anteriores dentro de un mismo ordenador.

En su script tenemos el código correspondiente al ajuste de brillo mediante un Post Process Layer y la carga de la siguiente escena.

Script Asociado → PantallaBrillo

3.3. Escena de menú.

Escena que hace de menú principal, en ella accedemos al juego, a la pantalla de records, las opciones o salir del juego. En cuanto a su código tenemos la carga de cada una de las escenas correspondientes a los botones.

Script Asociado → MenuPrincipal

3.4. Escena records.

Escena del menú de récords, es la parte del menú con más código asociado.

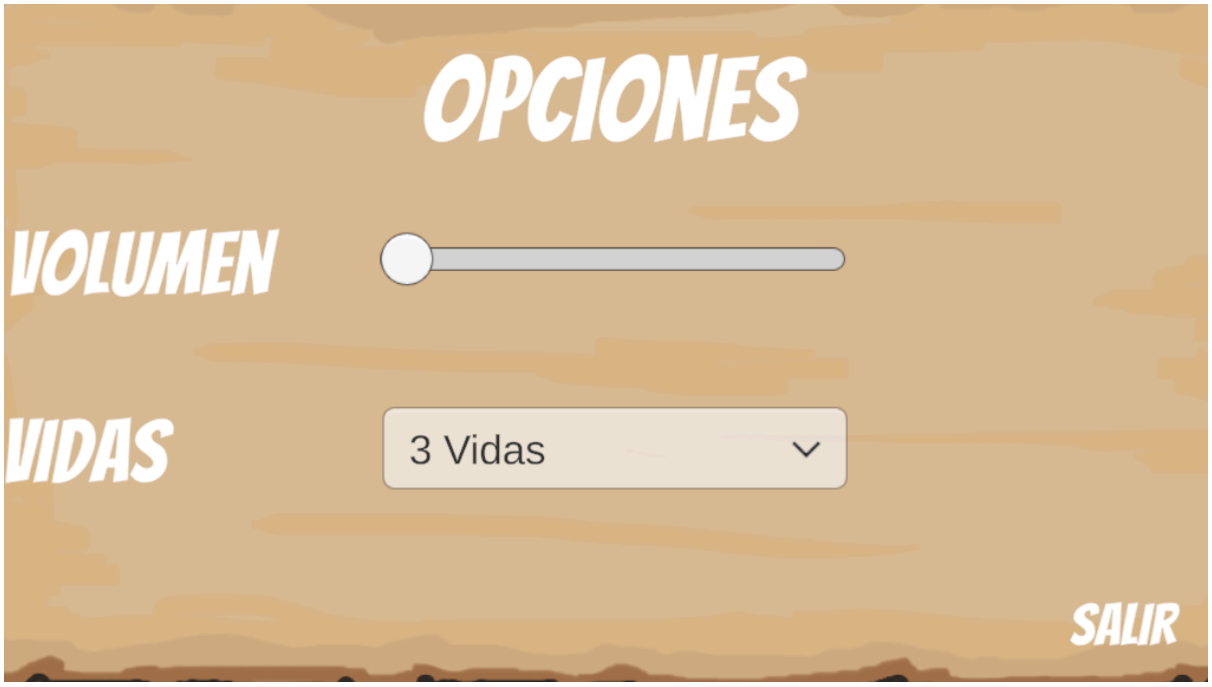
-Se encarga de guardar cada uno de los puestos en PlayerPrefs (Primero, Segundo Y Tercero).

-Cada vez que se entra a esta escena se hace una comparativa de el último valor guardado en las PlayerPrefs de los puestos y las de la puntuación de la última partida jugada.

-Se compara el número de rondas de la última partida con el número de rondas de cada puesto para ajustar la tabla de records.

Script Asociado → Records

3.5. Escena de opciones.



En esta escena podemos modificar el volúmen del juego al valor que queramos, al igual que el brillo se guarda en las PlayerPrefs y carga el valor al comienzo del juego.

El sistema para selección de vidas está implementado en caso de que quisiéramos utilizarlo pero no ha sido el caso para este juego, ya que implementamos un sistema de regeneración de vida automático y no lo vimos necesario.

3.6. Scripts

Tabla para facilitar la localización de Scripts en el sistema

Sistema	Script	Sistema	Script
Inicio	PantallaInicio		
Brillo	PantallaBrillo		
Menú Principal	MenuPrincipal		
Récords	Records		
Opciones	Opciones		

3.7. Assets

Zombie: https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232

Granada: https://assetstore.unity.com/packages/3d/props/weapons/m26-grenade-207012

Sonidos Armas:
https://assetstore.unity.com/packages/audio/sound-fx/shooting-sound-177096

Sonidos Zombies:
https://assetstore.unity.com/packages/audio/sound-fx/creatures/free-zombie-character-sounds-141740

UI: https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811

4. Lógica del juego

4.1. Radar Minimapa

Cámara: Una cámara colocada en una posición elevada para una vista cenital de la escena.

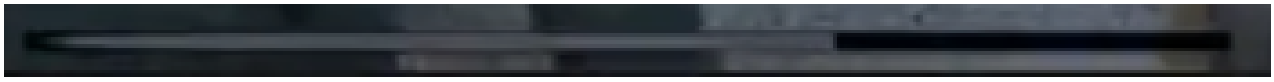
Render Texture: Un Render Texture asociado a la cámara para capturar la vista de la escena desde arriba.

Canvas UI: Un canvas UI colocado en la escena para mostrar el radar y los elementos visuales.

Imágenes: para representar elementos en el radar, como el jugador y otros objetos.

Esferas 3D: colocadas en la escena para representar objetos o jugadores en el radar.

4.2. Barra de Vida



barraVida: Imagen UI que representa la barra de vida.

vidaActual: Salud actual del jugador.

vidaMaxima: Salud máxima del jugador.

Update(): Método llamado en cada frame para comprobar y actualizar la longitud de la barra.

Actualización de la Barra de Vida:

En Update(), se actualiza el valor de **barraVida**.**fillAmount** basándose en la proporción de **vidaActual** sobre **vidaMaxima**.

4.3. Regeneración de Vida

Variables Clave:

regenTimer: Temporizador para la regeneración.

regenInterval: Intervalo de tiempo entre regeneraciones.

regenAmount: Cantidad de vida regenerada cada intervalo.

Lógica de Regeneración:

En **Update()**, se incrementa regenTimer con Time.deltaTime.

Si regenTimer supera **regenInterval**, se regenera la vida:

vidaActual se incrementa hasta un máximo de **vidaMaxima**.

regenTimer se reinicia.

4.4. Manejo de Daño

Método Principal:

TakeDamage(int damage): Disminuye **vidaActual** por el valor de damage.

Lógica de Daño:

regenTimer se reinicia para evitar la regeneración inmediata tras recibir daño.

Si **vidaActual** llega a 0 o menos:

Guarda la ronda actual en PlayerPrefs.

Carga la escena "**Records**" usando **SceneManager.LoadScene("Records").**

4.5. Armas

Componentes Clave:

Variables de configuración **(range, impactForce, fireRate, damage, maxAmmoLoad, ammoLoad, reloadTime).**

Componentes de Unity **(AudioClip, Animator, Camera, ParticleSystem, GameObject).**

Configuración Inicial

Método Start(): Inicializa ammoLoad con **maxAmmoLoad** si ammoLoad es -1.

Método **OnEnable**(): Resetea el estado de recarga y actualiza el animador.

Método Update():

Verifica si el arma está recargando y si es necesario recargar.

Gestiona la cadencia de fuego (**fireRate**).

Llama al método Shoot() si se presiona el botón de disparo.

Método Shoot():

Reproduce el sonido de disparo y el efecto visual del disparo.

Disminuye **ammoLoad**.

Realiza un Raycast para detectar impactos y aplicar daño.

Método Reload():

Inicia una corrutina para gestionar la recarga.

Reproduce el sonido de recarga.

Actualiza el animador y espera el tiempo de recarga antes de resetear **ammoLoad**.

Efectos Visuales:

muzzleFlash.Play(): Reproduce el efecto visual del disparo.

Crea y destruye efectos de impacto (**impactEffect**).

Efectos Sonoros:

reproducirAudio(AudioClip clip1): Crea un objeto temporal para reproducir sonidos de disparo y recarga.

4.6. Sistema de Cambio de Armas

Componentes Principales:

int selectedWeapon: Índice del arma actualmente seleccionada.

void Start(): Inicializa la selección de armas.

void Update(): Maneja la lógica de cambio de armas.

void SelectWeapon(): Activa o desactiva las armas basándose en la selección.

Método Update: Almacena el arma seleccionada previamente.

Cambia de arma usando la rueda del ratón:

Input.GetAxis("Mouse ScrollWheel") > 0f: Cambia al arma siguiente.

Input.GetAxis("Mouse ScrollWheel") < 0f: Cambia al arma anterior.

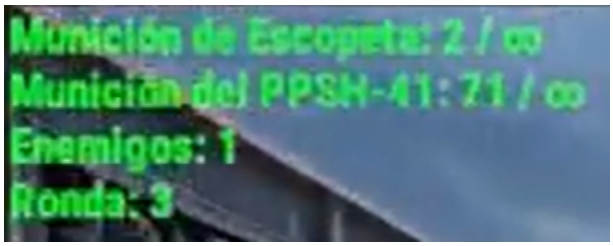
Cambia de arma usando teclas numéricas:

Input.GetKeyDown(KeyCode.Alpha1): Selecciona el arma 1.

Input.GetKeyDown(KeyCode.Alpha2): Selecciona el arma 2.

Llama a **SelectWeapon()** si el arma seleccionada ha cambiado.

4.7. Sistema de Interfaz en pantalla



Funcionalidades principales:

Mostrar Munición de Armas

Mostrar Número de Enemigos

Mostrar Ronda Actual

Componentes Principales:

Text texto: Componente de UI para mostrar la información.

Escopeta2Canones escopeta: Referencia al script de la escopeta de 2 cañones.

PPSH41 ppsh: Referencia al script del PPSH-41.

RoundManager RM: Referencia al script que maneja las rondas del juego.

texto.text: Se actualiza con la información de la munición de las armas, número de enemigos y ronda actual.

4.8. Transición de Escena

Condiciones para Transición:

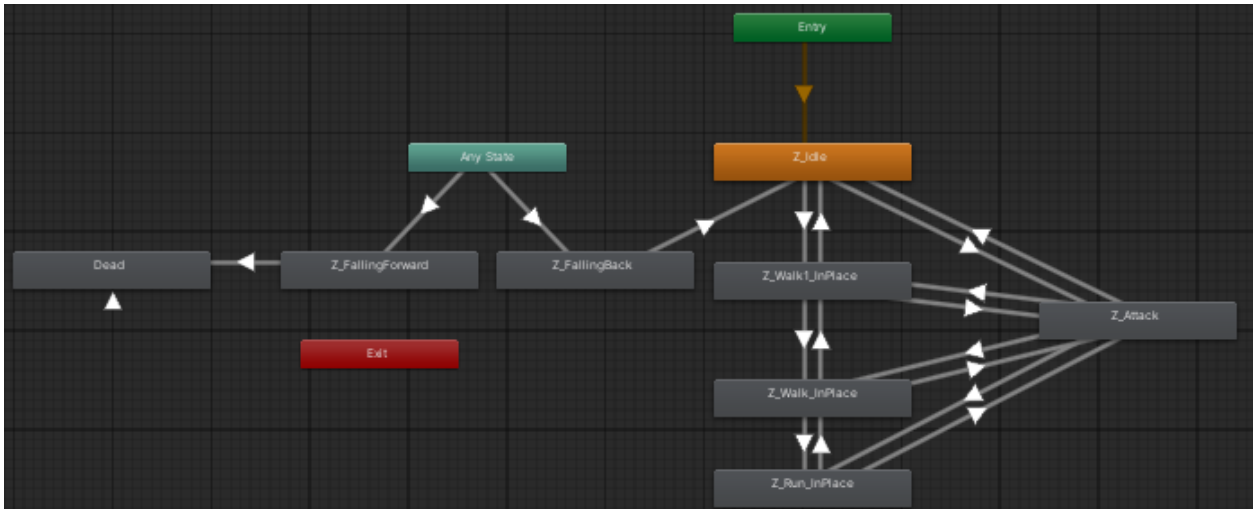
Si vidaActual es menor o igual a 0 en **TakeDamage().**

Acciones:

Guarda la ronda actual (**rm.GetRound()**) en PlayerPrefs.

Carga la escena "**Records**" usando **SceneManager.LoadScene("Records").**

4.9. Inteligencia Artificial de Zombie



Funcionalidades principales:

Patrullaje

Persecución y Ataque al Jugador

Reproducción de Sonidos

Gestión de Salud y Daño

Método Update:

Determina si el jugador está en el rango de visión o de ataque del zombie.

Decide si patrullar, perseguir al jugador o atacar al jugador.

Actualiza la distancia más cercana al jugador.

Métodos de Patrullaje, Persecución y Ataque:

Patroling(), ChasePlayer(), AttackPlayer(): Implementan el comportamiento del zombie en diferentes situaciones.

Reproducen sonidos específicos en cada acción.

Para el mapeado se ha utilizado **Nam Mesh Surface**, que delimita el área por donde el zombie puede caminar.

Gestión de Daño y Muerte

TakeDamage(int damage): Reduce la salud del zombie y gestiona su reacción al daño.

DestroyEnemy(): Destruye el zombie cuando su salud llega a cero, agregando datos de muerte al **RoundManager**.

4.10. Gestión de Rondas

Variables Clave:

currentRound: Ronda actual del juego.

EnemyCounts: Número actual de enemigos.

ZombiesMuertos: Número de zombies muertos.

Métodos Principales:

GetRound(): Retorna la ronda actual.

GetEnemies(): Retorna el número actual de enemigos.

ZombieMuerto(): Incrementa el contador de zombies muertos.

IncrementRound(): Incrementa el número de rondas.

Evaluación de Fin de Ronda:

En Update(), se verifica si no hay zombies **(EnemyCounts == 0)** y si ha pasado suficiente tiempo **(gameTime > 10f).**

Transición a Nueva Ronda:

HandleRoundTransition(): Coroutine que maneja la transición entre rondas.

Llama a **EvaluateZombies()** para ajustar los parámetros genéticos.

Incrementa la ronda (**IncrementRound()**).

Espera 5 segundos **(WaitForSeconds(5f)).**

Inicia la nueva ronda **(StartNewRound()).**

4.11. Almacenamiento de Datos del Zombie

Constructor:

Asigna valores a los campos de la clase basados en los parámetros proporcionados por el sistema genetico.

Creación de Instancias:

ZombieData zombie = new ZombieData(strong, health, timeAlive, closestDistanceToPlayer);

Se utiliza para almacenar información sobre cada zombie después de su muerte. en el **RoundManager**

4.12. Sistema Genético

Variables Clave:

deadZombiesData: Lista que almacena datos de zombies muertos.

nextRoundHealth: Salud de los zombies en la próxima ronda.

nextRoundStrong: Fuerza de los zombies en la próxima ronda.

Evaluación Genética:

AddDeadZombieData(ZombieData zombieData): Añade datos de un zombie muerto a la lista.

EvaluateZombies(): Analiza **deadZombiesData** para determinar los mejores atributos de los zombies muertos (mayor tiempo vivo, menor distancia al jugador).

4.13. Respawn

Variables Clave:

respawnPoint1 y respawnPoint2: Puntos de reaparecimiento para zombies.

strong y health: Atributos genéticos para los zombies reaparecidos.

puntos: Array de puntos de spawn.

zombie: Prefab de zombie.

tiempoDeSpawn: Tiempo entre **spawns** de zombies.

numeroTotalSpawnear: Cantidad de zombies a spawnear.

Inicialización y Reaparición:

Start() y Awake(): Calcula los límites de spawn.

SetGenetic(int strong, float health): Ajusta los atributos genéticos.

Update(): Controla el tiempo de spawn y crea zombies según **numeroTotalSpawnear**.

CrearZombie(): Genera un zombie en una posición aleatoria dentro de los límites.

Ronda1(int NZ): Inicializa la ronda con **NZ zombies.**

CreaMasZombies(int cantidad): Ajusta el número de zombies a spawnear.

4.14. Enlaces al Proyecto

4.15. OneDrive

https://universidadhuelva-my.sharepoint.com/:f:/g/personal/juan_jimenez022_alu_uhu_es/ErtYAcHwU0dLmHIsh3TmujMBl17IdY2Y2XN8ocW1_4uJpg?e=ka4wB0

4.16. GitHub

https://github.com/juanjimenez022/PJ_P2

5. Conclusión.

Sin duda para los que estamos más metidos en el mundo de los videojuegos hacer un juego tanto en 2D como en 3D resulta interesante, pero es cierto que cuando empiezas a ver la cantidad de tiempo que se necesita para hacer cualquier cosa en este mundillo, te das cuenta de que es bastante más complejo de lo que parece.

El tiempo precisamente durante el curso no sobra y hacer un juego elaborado es bastante complicado cuando el tiempo no acompaña, aún así hemos aprendido mucho tanto en el manejo de Unity como en C sharp.