Proyecto de PJ
Practica 2

Juan Jiménez Serrano
Luis Miguel Pérez Martín

Índice

Pantalla Inicio.

-Pantalla en movimiento.

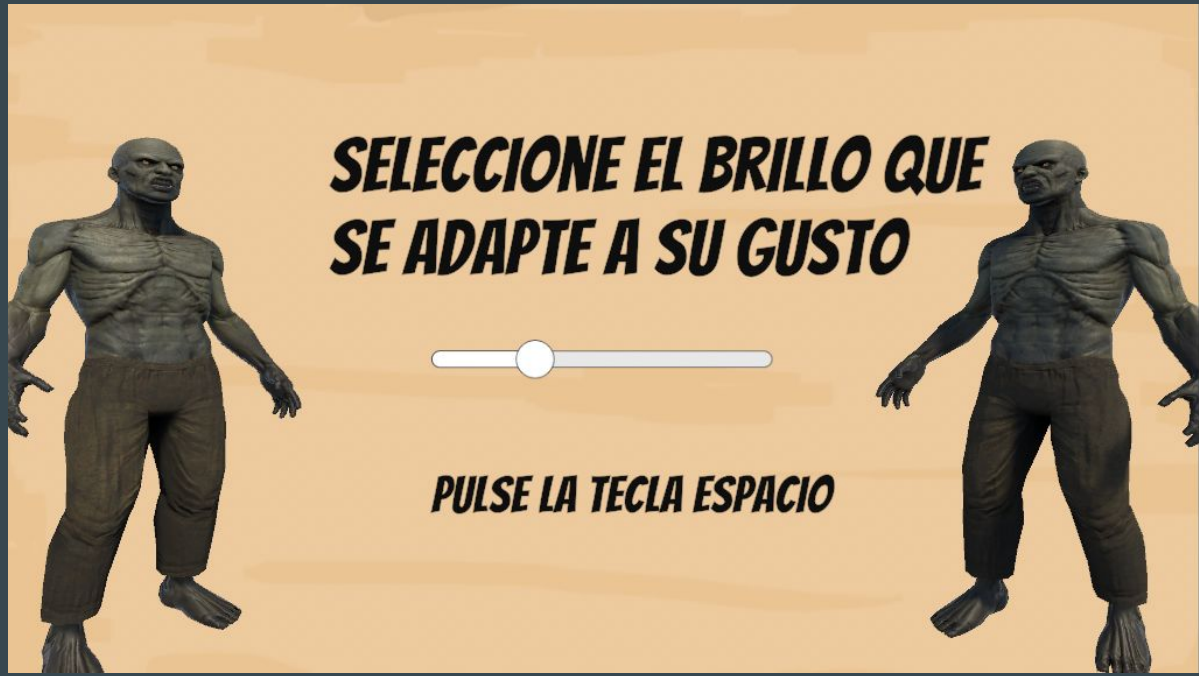
-Sensación de menú 3D.

Pantalla selección brillo.

-Slider para ajustar brillo.

-Recomendado no aumentar mucho el brillo.

-Se guarda en PlayerPrefs.



Menú principal.

-Acceso a:

- Iniciar Partida
- Menú records
- Menú Opciones
- Salir

Menú records.

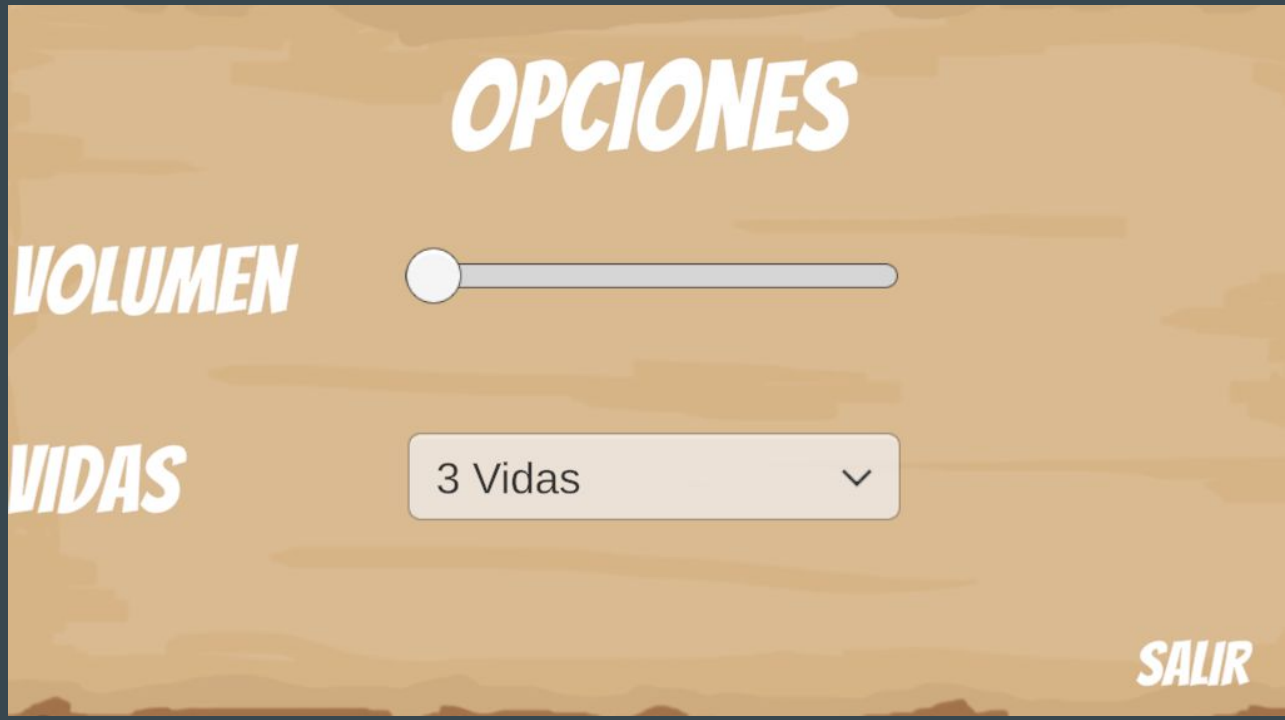
-Número rondas y puestos en PlayerPrefs.

-Carga escena y ajusta los puestos en función de los valores.

Menú opciones.

—Selector volumen.

-Selector vidas.



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.

Barra de Vida

barraVida: Imagen UI que representa la barra de vida.

vidaActual: Salud actual del jugador.

vidaMaxima: Salud máxima del jugador.

Actualización de la Barra de Vida:

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

Lógica de Actualización de la Barra de Vida

Update(): Método llamado en cada frame.

Actualización de la Barra:

Si **vidaActual** es mayor que 0:

Actualiza barraVida.**fillAmount** basándose en la proporción de **vidaActual** sobre **vidaMaxima**.

Regeneración de Vida

Variables Clave:

regenTimer: Temporizador para la regeneración.

regenInterval: Intervalo de tiempo entre regeneraciones.

regenAmount: Cantidad de vida regenerada cada intervalo.

Regeneración de Vida

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.

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").**

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.

Armas: Lógica de Disparo

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.

Armas: Lógica de Recarga

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.

Armas: Efectos Visuales y Sonoros

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.

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.

Sistema de Cambio de Armas

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.

Sistema de Interfaz en pantalla

Funcionalidades principales:

- Mostrar Munición de Armas
- Mostrar Número de Enemigos
- Mostrar Ronda Actual

Sistema de Interfaz en pantalla

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.

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").

Inteligencia Artificial de Zombie

Funcionalidades principales:

Patrullaje

Persecución y Ataque al Jugador

Reproducción de Sonidos

Gestión de Salud y Daño

Inteligencia Artificial de Zombie

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.

Inteligencia Artificial de Zombie

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

Inteligencia Artificial de Zombie

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.

Gestión de Rondas

Variables Clave:

currentRound: Ronda actual del juego.

EnemyCounts: Número actual de enemigos.

ZombiesMuertos: Número de zombies muertos.

Gestión de Rondas

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 ronda.

Transición 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 de Rondas

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()**).

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

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.

Sistema Genético

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).

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.

Respawn

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.

Gameplay

¿Donde Descargar?

OneDrive

GitHub