Daylight Chaos: Zombie Invasion

Javier J. Cordero Álvarez

Índice

1 Descripción General del Juego 3

1.1 Personaje: 3

1.2 Enemigos: 4

1.3 Entorno: 5

2 Implementaciones de mecánicas y configuraciones clave en el código 6

2.1 Script PlayerLogic: 6

2.2 Script EnemiesLogic 7

2.3 Script EnemiesGenerator 8

2.4 Script Life 9

2.5 Script MovementSwing 10

2.6 Script WeaponSwing 11

2.7 Script BulletsScreen 12

2.8 Script WeaponController 12

2.9 Script WeaponLogic 13

3 Objetivos: 15

4 Capturas de Pantalla 15

5 Conclusión 20

6 Repositorio del juego 20

1 Descripción General del Juego

Título del Juego:

Daylight Chaos: Zombie Invasion

Descripción: ***"Daylight Chaos: Zombie Invasion"*** sumerge a los jugadores en un entorno 3D repleto de obstáculos estratégicos: contenedores imponentes, una casa y un tanque impenetrables, junto con cajas dispersas. En este escenario postapocalíptico, el desafío es claro: sobrevivir a la invasión zombi.

Los no-muertos emergen desde distintos puntos, acechando al jugador. La habilidad para esquivarlos y el manejo preciso de dos armas son cruciales para la supervivencia. La elección entre la 9 mm, efectiva pero con menor daño, y el poderoso Magnum, capaz de eliminar a los zombies con solo dos disparos, define el enfoque estratégico de defensa.

La tensión se intensifica con un marcador visual de balas en el cargador y en el cartucho, demandando una gestión inteligente de recursos. Sin indicadores de vida, cada embate zombi resta energía vital, y perderla significa el reinicio del desafío.

Con una mezcla vertiginosa de acción y estrategia, Daylight Chaos: Zombie Invasion desafía a los jugadores a mantenerse en pie en un mundo infestado, donde cada decisión puede significar la diferencia entre la supervivencia y la aniquilación."

Mecánicas Clave

1.1 Personaje:

Movimiento y Navegación:

El jugador puede moverse libremente por el entorno 3D, utilizando controles para esquivar obstáculos y posicionarse estratégicamente contra los zombies.

Selección y Cambio de Armas:

Capacidad para elegir entre dos armas: una pistola 9 mm y un Magnum más poderoso. El cambio entre armas permite adaptarse a las diferentes situaciones tácticas.

Disparo y Precisión:

Habilidad para disparar con precisión a los zombies que se aproximan, gestionando con cuidado las balas limitadas para eliminar las amenazas de manera efectiva, mediante el uso de zoom.

Gestión de Recursos:

Supervisión constante de las balas disponibles en el cargador y en reserva, requiriendo toma de decisiones estratégicas sobre el uso de municiones.

Reinicio ante Derrota:

La pérdida total de vida lleva al reinicio del juego, promoviendo un enfoque táctico y una estrategia más cuidadosa en cada encuentro.

1.2 Enemigos:

Aparición Dinámica:

Los zombies emergen desde múltiples puntos del entorno, proporcionando un desafío constante y variado para el jugador.

Persecución Inteligente:

Una vez detectan al jugador, los zombies persiguen activamente, obligando a decisiones tácticas rápidas y habilidades de evasión.

Amenaza Progresiva:

A medida que avanza el juego, la cantidad de zombies aumenta, intensificando el desafío y la presión sobre el jugador.

Ataque y Amenaza:

Al acercarse, los zombies intentan atacar al jugador, exigiendo una estrategia defensiva o la utilización efectiva de armas para neutralizar la amenaza.

Persistencia en la Invasión:

Constante flujo de enemigos zombi, creando una atmósfera de constante peligro y requiriendo tácticas sólidas de supervivencia.

Desafío ante Contacto:

El contacto con los zombies resta vida al jugador, lo que exige movimientos calculados y precisos para evitar daños y mantenerse con vida.

1.3 Entorno:

Obstáculos Estratégicos:

Contenedores, una casa y un tanque conforman obstáculos estratégicos que el jugador debe sortear para evitar el avance de los zombies y planificar la defensa.

Puntos de Aparición:

Los zombies emergen desde múltiples puntos del entorno, creando un desafío dinámico y obligando al jugador a estar alerta en todo momento.

Espacios Limitados:

El entorno limita el espacio de movimiento, promoviendo una estrategia táctica para eludir a los enemigos y maximizar las oportunidades de defensa.

Elementos No Interactivos:

Los elementos como la casa y el tanque no son accesibles ni interactuables, sirviendo únicamente como elementos visuales y obstáculos estáticos.

Configuración Inmersiva:

El diseño del entorno postapocalíptico crea una atmósfera inmersiva, añadiendo profundidad a la experiencia del jugador y estableciendo el tono del juego.

Peligro Ambiental:

A pesar de su estática, la disposición de los obstáculos puede influir en el flujo de la invasión zombi, agregando un nivel adicional de desafío estratégico.

2 Implementaciones de mecánicas y configuraciones clave en el código

En este apartado se describen las configuraciones y mecánicas de los principales scripts del proyecto.

2.1 Script PlayerLogic:

El script **PlayerLogic** es responsable de varias funciones clave en el juego:

Supervisión de la Vida del Jugador:

- Al inicio del juego, obtiene y almacena una referencia al componente **Life**, que gestiona la vida del jugador.

Verificación Continua de la Vida:

- En cada fotograma del juego, el método **LifeReview()** se llama desde el método **Update()**, verificando si la vida del jugador ha llegado a cero.

Detección de Pérdida de Vida:

- El método **LifeReview()** comprueba si la variable **life0** es falsa (indicando que el jugador aún no ha perdido toda su vida) y si la vida actual del jugador es igual o menor a cero.

Activación de Reinicio:

- Si la vida del jugador ha alcanzado cero, el script establece la variable **life0** como verdadera y programa el método **RestartGame()** para ejecutarse después de una pausa de 2 segundos utilizando **Invoke()**.

Reinicio del Juego:

- Cuando **RestartGame()** se ejecuta después de la pausa, carga de nuevo la escena actual utilizando **SceneManager.LoadScene()**, devolviendo al jugador al estado inicial del juego.

Estas funciones permiten al script detectar la pérdida de vida del jugador y, tras un breve intervalo, reiniciar la partida para brindar una oportunidad adicional de enfrentarse a la invasión zombi.

2.2 Script EnemiesLogic

El script **EnemiesLogic** está diseñado para controlar el comportamiento de los enemigos (zombies) en el juego, llevando a cabo las siguientes funciones:

Objetivo: Controlar la lógica de los zombies para su interacción con el jugador.

Funcionalidad:

- **Inicialización de Variables:** Al inicio del juego, el script configura las referencias necesarias, incluyendo el objeto del jugador, su vida y su lógica (**Life** y **PlayerLogic**).
- **Gestión de Acciones ante Pérdida de Vida:**
  - ✓ Verifica constantemente la vida del zombie. Cuando la vida alcanza cero:
    - ❖ Detiene la navegación (**NavMeshAgent**), desactiva el collider y activa una animación de muerte.
    - ❖ Programa la destrucción del objeto zombie después de un tiempo específico.
- **Persecución al Jugador:**
  - ✓ Determina la posición del jugador como destino del zombie para seguirlo utilizando **NavMeshAgent** si ambos, el jugador y el zombie, están vivos.
- **Evaluación y Realización de Ataques:**
  - ✓ Verifica la distancia y orientación con respecto al jugador para decidir si el zombie puede atacar.
  - ✓ Al atacar, reduce la vida del jugador, detiene temporalmente al zombie y activa una animación de ataque.
  - ✓ Tras el ataque, reinicia el estado de ataque del zombie para continuar persiguiendo al jugador.

Métodos Clave:

- **LifeReview()**: Controla las acciones del zombie ante la pérdida de vida.
- **Pursue()**: Define el objetivo del zombie para seguir al jugador.
- **AttackReview()**: Evalúa si el zombie puede atacar al jugador.
- **Attack()**: Realiza el ataque al jugador y activa una breve pausa en el zombie.
- **RestartAttack()**: Restablece el estado del ataque del zombie después de realizar un ataque.

Este script asegura la interacción dinámica y desafiante entre los zombies y el jugador, controlando su movimiento, ataque y reacción ante la pérdida de vida.

2.3 Script EnemiesGenerator

El script **EnemiesGenerator** se encarga de generar y controlar la aparición periódica de enemigos (zombies) en el juego, realizando las siguientes funciones:

Objetivo:

- Generar enemigos de manera periódica en diferentes puntos del escenario para mantener la dinámica del juego.

Funcionalidad:

- **Generación de Enemigos:**
  - **Inicialización de Puntos de Generación:** Al inicio del juego, el script identifica los puntos de generación disponibles dentro del objeto actual.
  - **Corrutina de Aparición de Enemigos:**
    - En un bucle continuo (**while (true)**), genera enemigos (**zombiePrefab**) en cada uno de los puntos de generación.
    - Espera un tiempo definido (**generationTime**) entre cada aparición de enemigos.

Métodos Clave:

- **Start():**
  - Inicializa los puntos de generación identificando los puntos disponibles dentro del objeto.
  - Comienza la corutina **EnemyAppear()** para la generación periódica de enemigos.
- **EnemyAppear():**
  - En un bucle infinito, itera sobre los puntos de generación para instanciar (**Instantiate()**) enemigos (**zombiePrefab**) en sus posiciones.
  - Espera un intervalo de tiempo definido antes de continuar con la próxima generación.
- **Update():**
  - No contiene funcionalidades específicas y no se utiliza en este contexto.

Este script asegura que los enemigos sean generados de forma periódica en múltiples puntos del escenario, manteniendo la presión sobre el jugador durante el transcurso del juego.

2.4 Script Life

El script **Life** controla y gestiona la cantidad de vida asociada a un objeto en el juego, y realiza las siguientes funciones:

Objetivo:

- Manejar la vida de un objeto específico en el juego y gestionar la reducción de su vida cuando recibe daño.

Funcionalidad:

- **Variable de Vida:**
  - ✓ **value**: Almacena el valor numérico que representa la cantidad de vida del objeto. Por defecto, inicia en 100.

Métodos:

- **TakeDamage(float damage):**
  - ✓ Este método es invocado cuando el objeto recibe daño.
  - ✓ Reduce el valor de la vida (**value**) según el daño recibido.
  - ✓ Verifica si el valor de vida ha caído por debajo de cero y, en ese caso, lo establece en cero para evitar valores negativos.

Start() y Update():

- Ambos métodos están vacíos y no contienen funcionalidad en este script, ya que no se requiere inicialización adicional o actualización constante de la vida en este contexto.

Este script permite que los objetos en el juego tengan una vida determinada y que esta disminuya cuando reciben daño. La reducción se controla a través del método **TakeDamage(float damage)**, que ajusta el valor de la vida según el daño recibido.

2.5 Script MovementSwing

El script **MovementSwing** controla el efecto de balanceo (bobbing) del movimiento del jugador, y realiza las siguientes funciones:

Objetivo:

- Simular un efecto de balanceo mientras el jugador se mueve en el juego para añadir realismo a su movimiento.

Variables:

- **timer**: Almacena el tiempo utilizado para calcular el movimiento de balanceo.
- **bobbingSpeed**: La velocidad del balanceo del movimiento.
- **bobbingAmount**: La amplitud del balanceo del movimiento.
- **midpoint**: La posición central (punto medio) del balanceo del movimiento.
- **weaponLogic**: Referencia al script **WeaponLogic** que controla la lógica del arma.

Método Update():

- Calcula el balanceo del movimiento en función de la entrada del jugador.
- Si la lógica del arma (**weaponLogic.activeADS**) indica que no está apuntando, se activa el balanceo.
- Calcula el balanceo utilizando la función **Mathf.Sin()** basada en el tiempo (**timer**) para simular el movimiento de balanceo mientras el jugador se mueve.
- Ajusta la posición local del objeto (**transform.localPosition**) en el eje Y para aplicar el efecto de balanceo.

Este script implementa un efecto de balanceo mientras el jugador se mueve, creando una sensación más realista de movimiento en el juego. El balanceo se controla en función de la entrada del jugador y se aplica si no está apuntando con el arma.

2.6 Script WeaponSwing

El script **WeaponsSwing** controla el movimiento de balanceo (swing) de las armas del jugador cuando se apunta con ellas, realizando las siguientes funciones:

Objetivo:

- Simular un efecto de balanceo cuando el jugador apunta con las armas para añadir realismo al juego.

Variables:

- **amount**: Define la cantidad de balanceo de las armas.
- **maxAmount**: La cantidad máxima permitida para el balanceo de las armas.
- **time**: El tiempo de duración del balanceo de las armas.
- **initPosition**: Almacena la posición inicial de las armas.
- **setWeaponSwing**: Determina si se aplica o no el balanceo de las armas.

Método Start():

- Establece la posición inicial de las armas al inicio del juego.

Método Update():

- Calcula el movimiento del mouse (**Input.GetAxis("Mouse X/Y")**) y aplica el balanceo de las armas en función de este movimiento.
- Limita la cantidad de balanceo utilizando **Mathf.Clamp()** para asegurar que esté dentro de un rango específico (**-maxAmount** a **maxAmount**).
- Si se activa el zoom (**Input.GetMouseButton(1)**), desactiva el balanceo.
- Aplica el balanceo de las armas (**transform.localPosition**) utilizando **Vector3.Lerp()** para suavizar el movimiento durante el tiempo definido (**time**).

Este script implementa un efecto de balanceo en las armas cuando el jugador las apunta, añadiendo un nivel de realismo al juego. El balanceo se activa con el movimiento del ratón y se suaviza para evitar cambios bruscos.

2.7 Script BulletsScreen

El script **BulletsScreen** controla la interfaz de usuario (UI) que muestra la cantidad de balas del arma del jugador, realizando las siguientes funciones:

Objetivo:

- Mostrar dinámicamente en la pantalla la cantidad de balas en el cargador, el tamaño del cargador y la cantidad restante de balas del jugador.

Método Update():

- Actualiza continuamente el texto mostrado en la pantalla (**text.text**) con la cantidad de balas en el cargador, el tamaño del cargador y la cantidad restante de balas del jugador.
- Obtiene esta información del componente **weaponLogic** asociado al arma del jugador.

Este script se encarga de mantener actualizada la UI mostrando información crucial sobre las balas del arma del jugador, permitiendo que esta información sea visible en tiempo real durante el juego.

2.8 Script WeaponController

El script **WeaponController** gestiona el cambio de armas del jugador y su activación, realizando las siguientes funciones:

Objetivo:

- Permitir al jugador cambiar entre diferentes armas disponibles.

Variables:

- **weapons**: Un array de objetos **WeaponLogic** que representan las armas disponibles para el jugador.
- **currentWeaponIndex**: El índice de la arma actualmente seleccionada.

Métodos:

- **Update():**
  - ✓ Llama continuamente al método **WeaponChangeReview()** para revisar si hay cambios en las armas.
- **changeCurrentWeapon():**
  - ✓ Desactiva todas las armas excepto la que está seleccionada (**currentWeaponIndex**) para activarla.

- **WeaponChangeReview():**
  - ✓ Revisa si el jugador ha utilizado la rueda del mouse para cambiar de arma.
  - ✓ Si hay un movimiento hacia arriba en la rueda del mouse (**mouseScroll > 0f**), selecciona el arma anterior.
  - ✓ Si hay un movimiento hacia abajo en la rueda del mouse (**mouseScroll < 0f**), selecciona el arma siguiente.
- **SelectPreviousWeapon():**
  - ✓ Selecciona el arma anterior en la lista de armas disponibles.
  - ✓ Si el jugador está en la primera arma, selecciona la última de la lista.
- **SelectNextWeapon():**
  - ✓ Selecciona el arma siguiente en la lista de armas disponibles.
  - ✓ Si el jugador está en la última arma, selecciona la primera de la lista.

Este script permite al jugador cambiar entre armas disponibles utilizando la rueda del mouse, activando y desactivando las armas correspondientes para garantizar que solo una esté activa en un momento dado.

2.9 Script WeaponLogic

El script **WeaponLogic** controla el comportamiento y las funciones principales del arma del jugador, realizando las siguientes acciones:

Objetivo:

- Gestionar el comportamiento del arma, incluyendo disparos, recargas, cambio de modos de disparo, y funcionalidades asociadas al uso del arma.

Variables:

- **Referencias a Objetos y Sonidos:**
  - ✓ **fireWeapon**: Sistema de partículas que representa el fuego del arma.
  - ✓ **mainCamera**: Referencia a la cámara principal del juego.
  - ✓ **shootPoint**: Punto de origen de los disparos.
  - ✓ Varios clips de audio para sonidos relacionados con el arma.

- **Atributos del Arma:**
  - ✓ **shootMode**: Define el modo de disparo del arma (SemiAutomático o Automático).
  - ✓ **damage**: Cantidad de daño infligido por cada disparo.
  - ✓ **shootRhythm**: Tiempo entre disparos.
  - ✓ **remainingBullets**: Cantidad total de balas restantes para recargar.
  - ✓ **bulletsInCartridges**: Cantidad actual de balas en el cargador.
  - ✓ **cartridgeSize**: Tamaño máximo del cargador.
  - ✓ **maxBullets**: Cantidad máxima total de balas disponibles.
  - ✓ Varios atributos relacionados con la mira, el tiempo de apuntado, etc.

Métodos Principales:

- **Update():**
  - ✓ Controla las acciones del arma, incluyendo disparos, recargas y cambios entre modos de apuntado.
- **ShootReview():**
  - ✓ Revisa si se puede realizar un disparo según las condiciones actuales del arma.
- **Shoot() y DirectShoot():**
  - ✓ Ejecutan el disparo del arma y manejan el raycast para detectar impactos.
- **WithoutBullets():**
  - ✓ Activa el sonido de sin balas cuando se intenta disparar sin munición.
- **ReloadReview() y Reload():**
  - ✓ Revisa y ejecuta la recarga del arma si es posible.
- **MunitionsReload() y métodos asociados al sonido de la recarga:**
  - ✓ Gestionan la recarga de balas en el cargador y los sonidos relacionados.
- **PlayAnimationShoot():**
  - ✓ Reproduce la animación de disparo correspondiente al arma.

Start() y Update():

- **Start()** inicializa algunas variables del arma al inicio del juego.
- **Update()** gestiona continuamente las acciones del arma en función de la entrada del jugador.

Este script controla de manera detallada el comportamiento del arma del jugador, incluyendo disparos, recargas, modos de disparo y animaciones asociadas. Además, administra efectos de sonido y cambios visuales en función de las acciones del jugador.

3 Objetivos:

Sobrevive al Asedio Zombie: Enfrenta oleadas de zombis, utilizando armas y habilidades para mantenerte con vida.

Explora y Resiste: Navega un entorno 3D lleno de obstáculos, busca refugio y recursos, evitando ser superado por los no muertos.

Administra Recursos: Gestiona con astucia la munición y los suministros dispersos para mantener tu ventaja.

Mejora tus Habilidades: Perfecciona la puntería, el movimiento estratégico y la toma de decisiones para sobrevivir más tiempo.

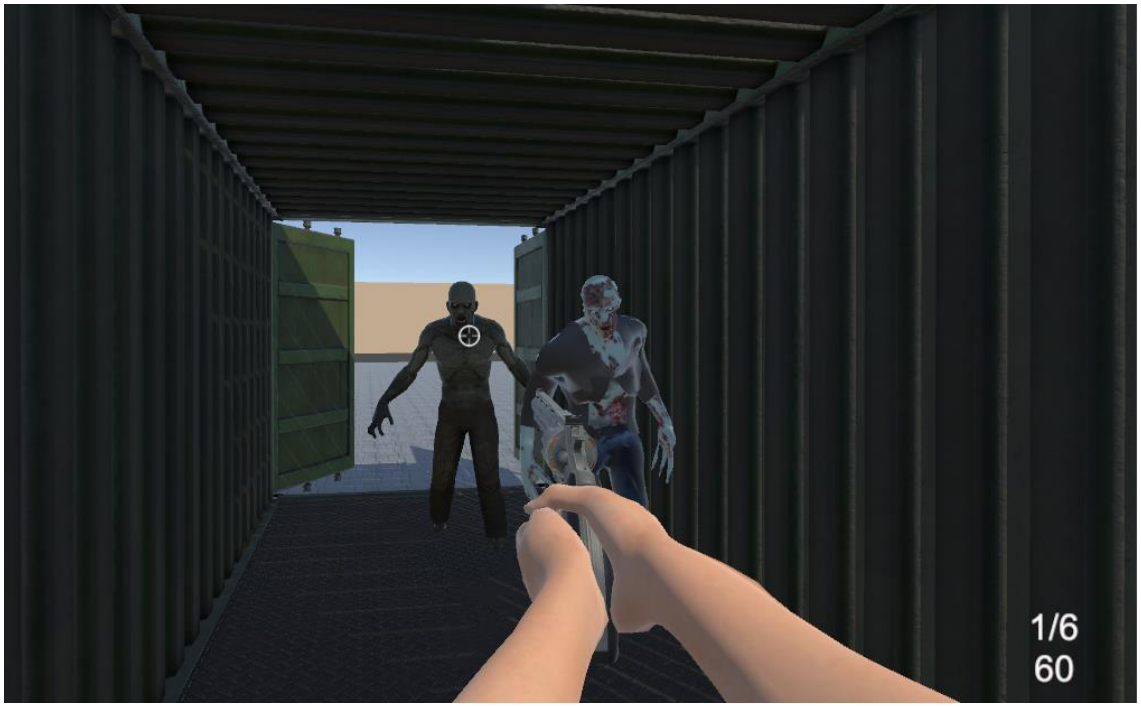
Alcanza la Victoria: Mantente vivo y resiste el asedio zombi, llegando a objetivos específicos.

4 Capturas de Pantalla

A continuación, se muestran las capturas más representativas en las que se incluyen momentos clave o características visuales distintivas del juego:

Escena de Combate:

Interacción con el Entorno:



Momentos de Combate Estratégico:

Zoom

Detalles de Armas o Personalización:

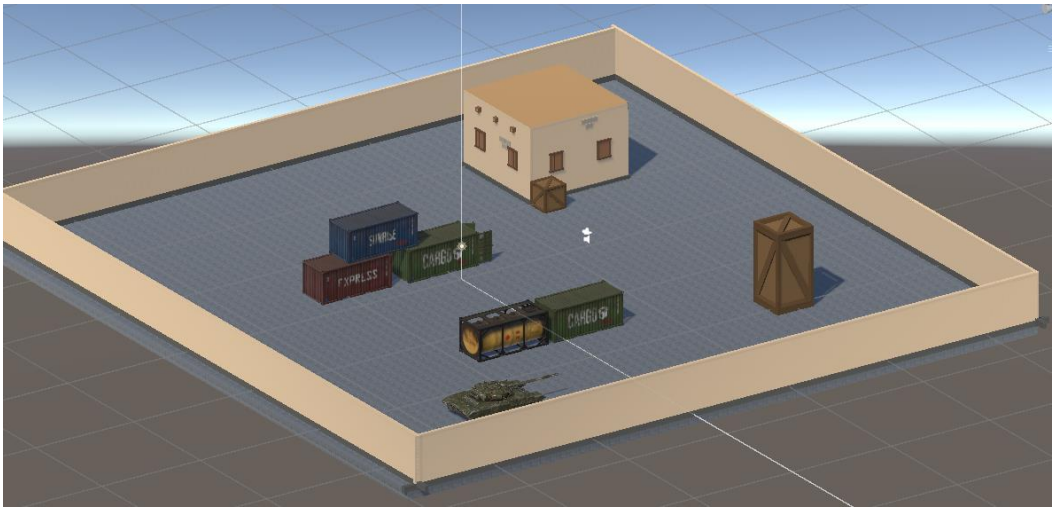
9 mm

Recarga 9 mm

Magnum

Recarga Magnum

Entorno del juego



Estas capturas podrían ofrecer una visión variada y completa del juego, mostrando tanto la acción intensa como los aspectos tácticos y estratégicos, junto con la esencia visual del entorno postapocalíptico y las herramientas disponibles para la supervivencia.

5 Conclusión

"Daylight Chaos: Zombie Invasion" es un juego envolvente y desafiante que sumerge a los jugadores en un mundo postapocalíptico lleno de zombis. La experiencia ofrece una combinación única de acción intensa, estrategia y supervivencia, invitando a los jugadores a enfrentarse a hordas de no muertos mientras exploran un entorno caótico y hostil.

Crear este juego no solo ha sido un ejercicio técnico, sino también una oportunidad para explorar la creatividad y el pensamiento estratégico. Ha sido gratificante ver cómo las ideas se materializan en un producto interactivo y cómo cada detalle contribuye a la experiencia del jugador.

En resumen, el desarrollo de "Daylight Chaos: Zombie Invasion" ha sido una aventura emocionante y educativa, proporcionando valiosas lecciones y una visión profunda en el fascinante mundo del desarrollo de videojuegos.

6 Repositorio del juego

https://github.com/javi-dev-79/Zombie_Invasion_3D