Documentación de scripts Nova Effect

Spawner.cs

Este script, EnemySpawner, se encarga de la generación de enemigos en oleadas. Utiliza una corutina para controlar el tiempo entre oleadas y la aparición de los enemigos.

- **Variables Públicas:**
  - eliteEnemyPrefab: El GameObject que se utilizará como plantilla para los enemigos.
  - spawnInterval: El tiempo en segundos que el script espera entre cada oleada de enemigos.
  - minEnemiesPerWave: El número mínimo de enemigos que aparecerán en una oleada.
  - maxEnemiesPerWave: El número máximo de enemigos que aparecerán en una oleada.
  - amplitude: Amplitud común para el movimiento sinusoidal de los enemigos.
  - frequency: Frecuencia común para el movimiento sinusoidal de los enemigos.
  - xSpawnRangeMin: La posición mínima en el eje X para la aparición de los enemigos.
  - xSpawnRangeMaxDefault: Rango máximo base para la aparición en el eje X, que es ajustado aleatoriamente.
- **Métodos Principales:**
  - Start(): Inicia la corutina SpawnSynchronizedWaves() al comienzo del juego.
  - SpawnSynchronizedWaves(): Una corutina que genera oleadas de enemigos de forma continua. Calcula un número aleatorio de enemigos por oleada, ajusta su posición de aparición y les añade los scripts Destructible y MoveSin. Los enemigos se distribuyen de manera equidistante en el eje X y tienen un desfase de fase para el movimiento sinusoidal, creando un movimiento sincronizado. La corutina espera a que todos los enemigos de la oleada actual sean destruidos antes de generar la siguiente.

Balas.cs

El script Bullet gestiona el comportamiento de los proyectiles en el juego.

- **Variables Públicas:**
  - direction: Un Vector2 que define la dirección del movimiento de la bala. Por defecto, se mueve hacia la derecha.
  - speed: Un float que determina la velocidad de la bala.

- velocity: Un Vector2 privado que almacena la velocidad calculada de la bala.

- **Métodos Principales:**
  - Start(): Llama a Destroy(gameObject, 3) para destruir la bala automáticamente después de 3 segundos, evitando que las balas que no golpean nada permanezcan en la escena.
  - Update(): Calcula la velocidad de la bala multiplicando la dirección por la velocidad.
  - FixedUpdate(): Actualiza la posición de la bala en cada paso de física, moviéndola en la dirección de su velocidad.

Destruible.cs

El script Destructible permite que un objeto sea destruido y afecta a la puntuación del juego.

- **Variables Públicas:**
  - scoreValue: Un int que representa la cantidad de puntos que el jugador gana al destruir este objeto.
  - canBeDestroyed: Un bool que controla si el objeto puede ser destruido.
- **Métodos Principales:**
  - Update(): Si la posición en el eje X del objeto es menor a 10f, la variable canBeDestroyed se establece en true.
  - OnTriggerEnter2D(Collider2D collision): Se activa cuando el objeto entra en colisión con otro Collider2D.
    - Verifica si canBeDestroyed es true.
    - Busca el componente Bullet en el objeto con el que colisionó.
    - Si el objeto colisionado tiene un script Bullet, el script suma puntos a través del GameManager, destruye la bala y destruye el propio GameObject.

Jugador.cs

El script Player controla el movimiento del jugador, el disparo y la colisión con los enemigos.

- **Variables Públicas:**
  - acceleration: La velocidad a la que el jugador acelera.
  - deceleration: La velocidad a la que el jugador desacelera.

- maxSpeed: La velocidad máxima del jugador.

- **Métodos Principales:**
  - Start(): Inicializa el Animator del motor, calcula los límites de la pantalla para evitar que el jugador se salga de ella y obtiene todos los componentes Gun adjuntos al jugador.
  - Update():
    - Maneja las entradas del teclado para el movimiento horizontal y vertical.
    - Ajusta la animación del motor según si el jugador se está moviendo.
    - Detecta si se presiona la tecla de espacio para llamar al método Shoot() de cada Gun.
    - Utiliza la función Lerp para un movimiento suave y asegura que el jugador se mantenga dentro de los límites de la pantalla.
  - OnTriggerEnter2D(Collider2D collision): Se activa al colisionar con otro Collider2D.
    - Si el objeto colisionado tiene el componente Destructible, el script destruye al enemigo y resta una vida al jugador a través del GameManager.

Menu.cs

Este script, MainMenu, es el encargado de crear y gestionar la interfaz de usuario del menú principal del juego de forma procedural.

- **Variables Privadas:**
  - canvas, titleText, playButton, quitButton: Referencias a los componentes de UI.

- **Métodos Principales:**
  - Start(): Este método es el corazón del script.
    - Crea y configura un Canvas para la interfaz.
    - Crea un objeto de texto para el título del juego ("Nova Effect").
    - Crea un objeto de texto para el autor del juego ("Vercetti Productions").
    - Llama al método CreateButton para crear el botón "Jugar" que inicia el juego y el botón "Salir" que cierra la aplicación.

- CreateButton(string name, string text, Vector2 position, UnityEngine.Events.UnityAction action): Un método auxiliar para generar botones.
- PlayGame(): Carga la escena del juego (índice 1).
- QuitGame(): Cierra la aplicación.

Arma.cs

El script Gun controla la funcionalidad de disparo de las armas.

- **Variables Públicas:**
  - bullet: Una referencia al script Bullet del prefab de la bala que se va a disparar.
  - direction: Un Vector2 que almacena la dirección actual de la bala.
- **Métodos Principales:**
  - Update(): Calcula la dirección de la bala basándose en la rotación local del GameObject de la pistola.
  - Shoot(): Instancia una nueva bala a partir del prefab, le asigna la dirección calculada y la dispara.

Fondo.cs

El script Background se encarga de crear un fondo espacial dinámico y procedural con estrellas y nebulosas.

- **Clases Anidadas:**
  - StarLayer: Almacena la configuración para una capa de estrellas (cantidad, velocidad de desplazamiento, tamaño, parpadeo).
  - NebulaLayer: Almacena la configuración para una capa de nebulosas (cantidad, velocidad, tamaño, colores posibles).
- **Variables Públicas:**
  - starLayers: Un array de StarLayer para tener múltiples capas de estrellas con diferentes configuraciones.
  - nebulaLayer: Un objeto de la clase NebulaLayer para las nebulosas.

    - horizontalBoundary, verticalBoundary: Límites de la pantalla para reposicionar los objetos del fondo.

- **Métodos Principales:**
    - Start(): Llama a InitializeBackground() para generar las estrellas y nebulosas iniciales.
    - InitializeBackground(): Genera todas las capas de estrellas y nebulosas.
    - CreateStar(StarLayer layer, List<ProceduralStar> stars): Crea un GameObject de estrella, le asigna una posición aleatoria, un tamaño y un SpriteRenderer.
    - CreateNebula(): Crea un GameObject de nebulosa, le asigna una posición y tamaño aleatorios, y un color de una paleta definida.
    - CreateStarSprite(): Genera un Sprite de estrella de forma procedural con un degradado circular.
    - CreateNebulaSprite(): Genera un Sprite de nebulosa utilizando ruido de Perlin.
    - Update(): Actualiza la posición de las estrellas y nebulosas para simular un efecto de paralaje. También gestiona el efecto de parpadeo de las estrellas y reposiciona los objetos cuando salen de la pantalla.
    - GetRandomPosition(): Devuelve una posición aleatoria dentro de los límites definidos.

RightLeftMovement.cs

El script MoveRightLeft mueve un GameObject de derecha a izquierda y lo destruye cuando sale de la pantalla.

- **Variables Públicas:**
    - moveSpeed: La velocidad a la que el objeto se mueve hacia la izquierda.
- **Métodos Principales:**
    - FixedUpdate(): Actualiza la posición del objeto en cada paso de física, moviéndolo hacia la izquierda. Si la posición en X es menor a -20, el objeto es destruido.

GameManager.cs

El GameManager gestiona la lógica principal del juego, como la puntuación, las vidas, la interfaz de usuario y el estado del juego.

- **Variables Públicas:**

- score: La puntuación actual del jugador.
- lives: Las vidas restantes del jugador.
- pointsPerExtraLife: Los puntos necesarios para ganar una vida extra.
- ScoreText, LivesText: Referencias a los componentes Text de la UI.
- gameOverPanel: La referencia al GameObject del panel de "Game Over".
- mainMenuSceneIndex: El índice de la escena del menú principal.

- **Propiedades:**
  - Instance: Implementa el patrón Singleton para asegurar que solo haya una instancia del GameManager en la escena.
- **Métodos Principales:**
  - Awake(): Implementa el patrón Singleton.
  - Start(): Inicializa las variables, oculta el panel de "Game Over" y actualiza la UI inicial.
  - AddScore(int points): Suma puntos a la puntuación del jugador y otorga una vida extra si se alcanza un umbral de puntos.
  - LoseLife(): Resta una vida al jugador y, si las vidas llegan a cero, llama al método GameOver().
  - UpdateUI(): Actualiza los textos de la UI con la puntuación y las vidas actuales.
  - GameOver(): Muestra el panel de "Game Over", pausa el juego y llama a una corutina para regresar al menú principal después de un retraso.
  - ReturnToMainMenuAfterDelay(float delay): Una corutina que espera un tiempo determinado y luego vuelve a la escena del menú principal.
  - RestartGame(): Reinicia la escena actual.

MoveSin.cs

Este script MoveSin hace que un objeto se mueva de forma sinusoidal en el eje Y.

- **Variables Públicas:**
  - amplitude: La altura del movimiento sinusoidal.
  - frequency: La frecuencia de la onda sinusoidal.

- phaseOffset: Un desfase para el movimiento, útil para sincronizar múltiples objetos.
- inverted: Un bool para invertir el movimiento.

- **Métodos Principales:**
  - Start(): Almacena la posición inicial en Y del objeto como el centro de la onda sinusoidal.
  - FixedUpdate(): Calcula el nuevo valor de la posición en Y del objeto basándose en la función Mathf.Sin y lo actualiza en cada paso de física.