

|  |  |
| --- | --- |
| Programación de Juegos 3D en Unity  OLD BLOOD |  |
| Juan Alberto Domínguez Vázquez  Saúl Rodríguez Naranjo  Universidad de Huelva | Ingeniería Informática | Programación de Juegos (2024) |  |

**ÍNDICE**

[INTRODUCCIÓN 3](#_Toc168654319)

[PANTALLA DE TÍTULO 3](#_Toc168654320)

[PRIMER NIVEL 3](#_Toc168654321)

[PANTALLA DE PUNTUACIÓN 3](#_Toc168654322)

[RECURSOS EXTERNOS 3](#_Toc168654323)

# INTRODUCCIÓN

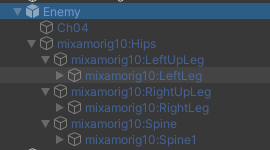
# PANTALLA DE TÍTULO

# PRIMER NIVEL

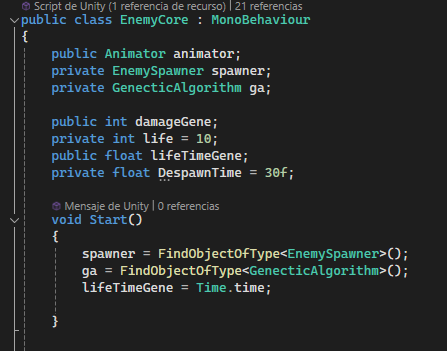
* **IMPLEMENTACIÓN DE ENEMIGOS**

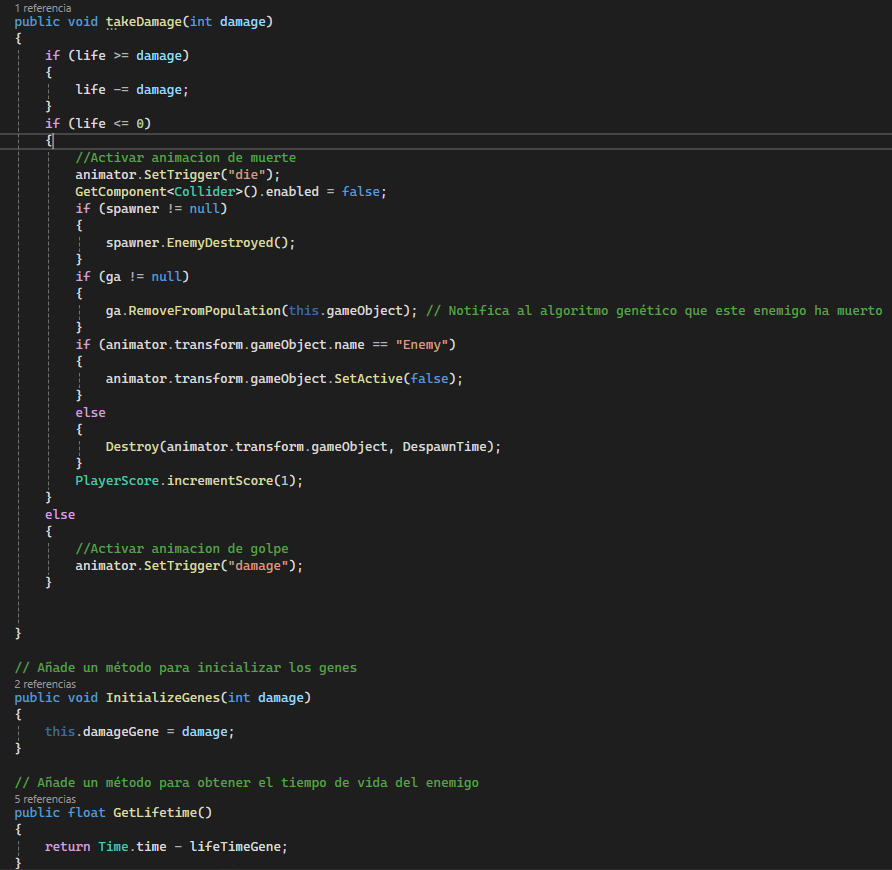
Para este proyecto solo contamos con un tipo de enemigo, que en Unity tiene por nombre Enemy. Dicho objeto, es importado de la página de creación de personajes y animaciones Mixamo.com, por tanto, ya incluye ciertos elementos para la correcta visualización de las animaciones y sus físicas.





El script adherido a este objeto que describe los atributos de estos enemigos se llama EnemyCore y en él se define el daño que pueden hacer, su vida y el tiempo que tardan en “despawnear”. También en dicho script, están los objetos de tipo EnemySpawner y GenecticAlgorithm que más tarde se explicarán sus usos.





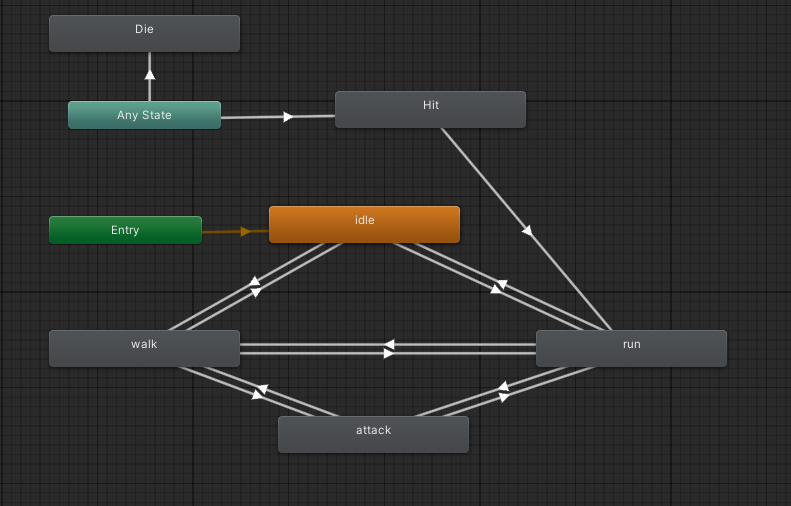
Analizando el script, en primer lugar, en el método start buscamos los objetos asignados al enemigo para su posterior uso. Después vemos el método takeDamage que se encarga de asignar daño a los enemigos en base al parámetro que se le pasa, también es encargado de activar las animaciones de muerte y daño cuando las condiciones se cumplan y de destruir al enemigo cuando el tiempo de “despawnear” se cumpla.

Por último, el método initializeGenes servirá para dotar de los genes necesarios a los enemigos a través del algoritmo genético.

* **TÉCNICAS DE IA UTILIZADAS**

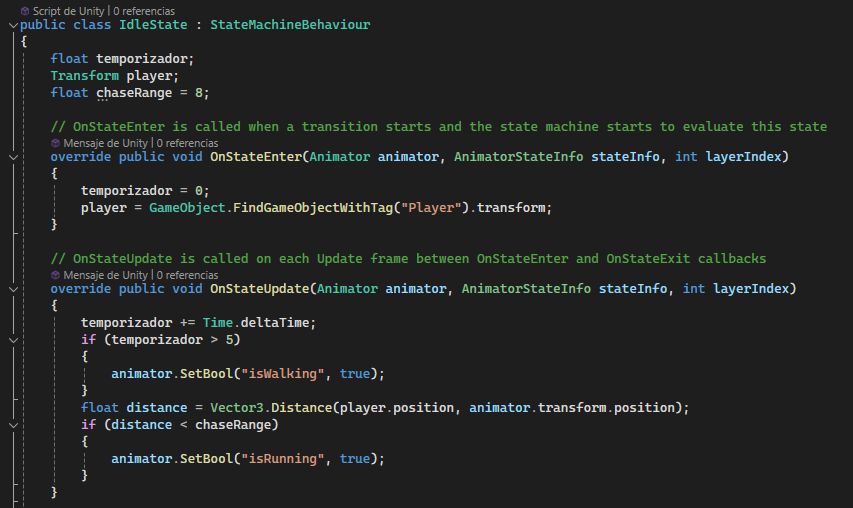
Para el seguimiento del jugador se ha optado por usar la herramienta de NavMesh de Unity, añadiéndole al objeto Enemy un Nav Mesh Agent y configurando en el mapa un objeto de tipo Nav Mesh Surface donde se configura que zona del mapa puede ser usada por el algoritmo para seguir al enemigo, que en este caso es todo el mapa visible.

Por otro lado, para la gestión del comportamiento del enemigo y de sus animaciones se ha usado una máquina de estados finitos de la siguiente forma:



**Estado IDLE:**

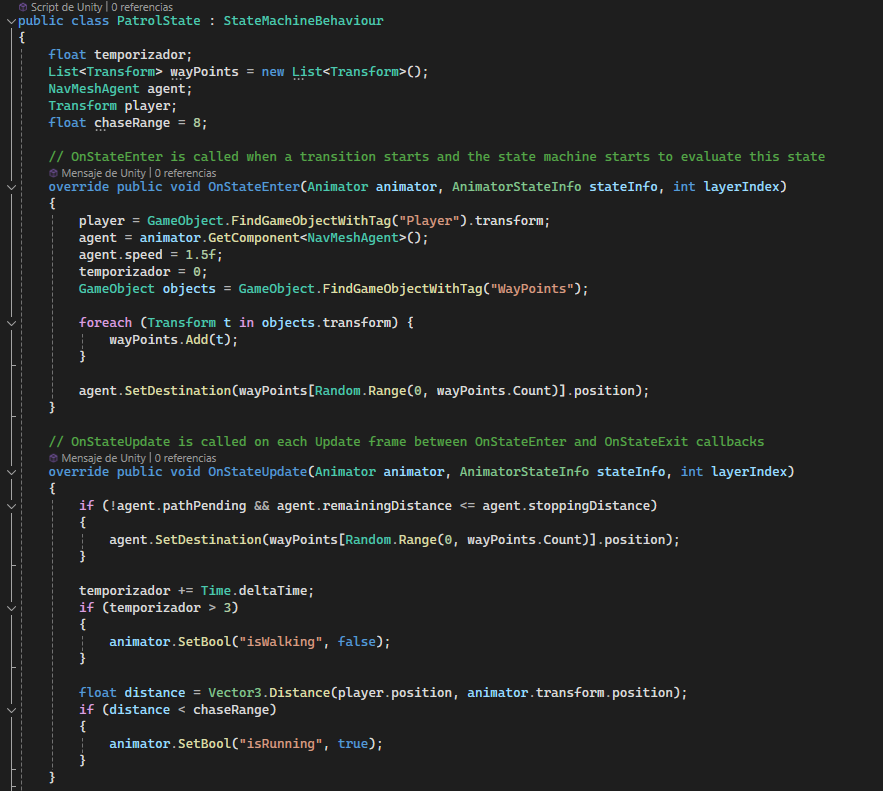
Si nos centramos en el estado de idle se ha desarrollado el siguiente script para modelar su comportamiento:

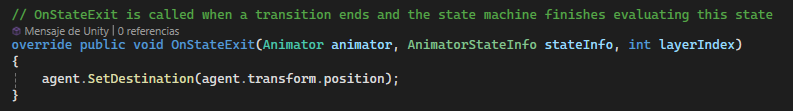


Vemos que se inicia un temporizador para cuando pasen mas de 5 segundos, el enemigo comenzara a caminar, pasando por tanto al estado walk y si la distancia con el jugador es menor a 8 unidades, comenzará a correr hacia él pasando al estado run.

**Estado WALK:**

Pasemos al estado walk que tiene definido un script llamado PatrolState que modela su comportamiento de la siguiente manera:





#### Funcionalidad Principal

* **Patrullaje Aleatorio:** El agente selecciona aleatoriamente puntos de referencia (wayPoints) y se mueve hacia ellos.
* **Detección del Jugador:** Si el jugador se encuentra dentro de un rango definido (chaseRange), el agente cambia su comportamiento para perseguir al jugador.
* **Actualización del Animador:** Dependiendo del estado del agente (patrullando o persiguiendo), se actualizan los parámetros del animador (isWalking, isRunning) para reflejar visualmente estos estados.

#### Detalles de Implementación

* **Variables:**
  + temporizador: Mide el tiempo para controlar las transiciones de comportamiento.
  + wayPoints: Lista de puntos de referencia para el patrullaje.
  + agent: Componente NavMeshAgent que maneja el movimiento del agente.
  + player: Transform del jugador para calcular la distancia.
  + chaseRange: Distancia a la cual el agente comienza a perseguir al jugador.
* **Métodos Sobrecargados:**
  + OnStateEnter: Inicializa las variables y configura el destino inicial del agente.
  + OnStateUpdate: Actualiza el destino del agente, controla el temporizador y cambia los parámetros del animador basado en la distancia al jugador.
  + OnStateExit: Detiene el movimiento del agente cuando el estado se desactiva.

**Estado RUN:**

Si vemos el estado run, veremos que tiene añadido el script ChaseState es un comportamiento personalizado para agentes de IA en Unity que define cómo el agente persigue al jugador y reacciona cuando está lo suficientemente cerca para atacar. Este script se integra con el sistema de animaciones de Unity mediante la herencia de StateMachineBehaviour, permitiendo transiciones suaves entre estados.

#### Funcionalidad Principal

* **Persecución del Jugador:** El agente persigue continuamente al jugador.
* **Transición a Estado de Ataque:** Si el agente se acerca lo suficiente al jugador, cambia su comportamiento para iniciar un ataque.
* **Actualización del Animador:** Dependiendo de la distancia al jugador, se actualizan los parámetros del animador (isRunning, isAttacking) para reflejar visualmente estos estados.

#### Detalles de Implementación

* **Variables:**
  + NavMeshAgent agent: Componente de Unity que maneja el movimiento del agente en una malla de navegación.
  + Transform player: Transform del jugador para calcular la distancia y establecer el destino.
* **Métodos Sobrecargados:**
  + OnStateEnter:
    - Se obtiene el componente NavMeshAgent del agente.
    - Se encuentra la posición del jugador.
    - Se ajusta la velocidad del agente para la persecución (3.5f).
  + OnStateUpdate:
    - Se establece la posición del jugador como el destino del agente.
    - Se calcula la distancia entre el jugador y el agente.
    - Si la distancia es mayor a 15 unidades, se desactiva el parámetro isRunning del animador.
    - Si la distancia es menor a 2.5 unidades, se activa el parámetro isAttacking del animador.
  + OnStateExit:
    - Se detiene el movimiento del agente estableciendo su posición actual como destino.

**Estado ATTACK:**

#### Funcionalidad Principal

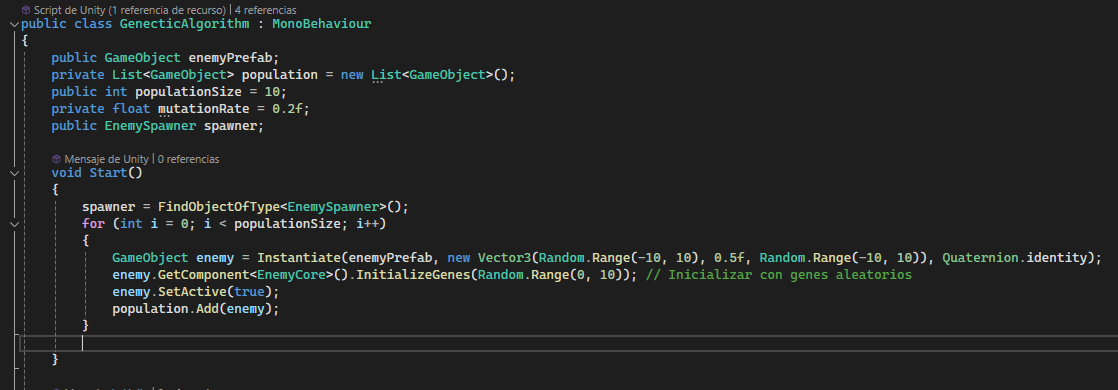
* **Orientación hacia el Jugador:** El agente se orienta continuamente hacia el jugador.
* **Detección de Proximidad:** Si el jugador está lo suficientemente cerca, el agente intentará atacar.
* **Ataque con Retraso:** El ataque del agente tiene un retraso configurado para no ocurrir continuamente.
* **Actualización del Animador:** Dependiendo de la distancia al jugador, se actualiza el parámetro del animador (isAttacking) para reflejar visualmente estos estados.

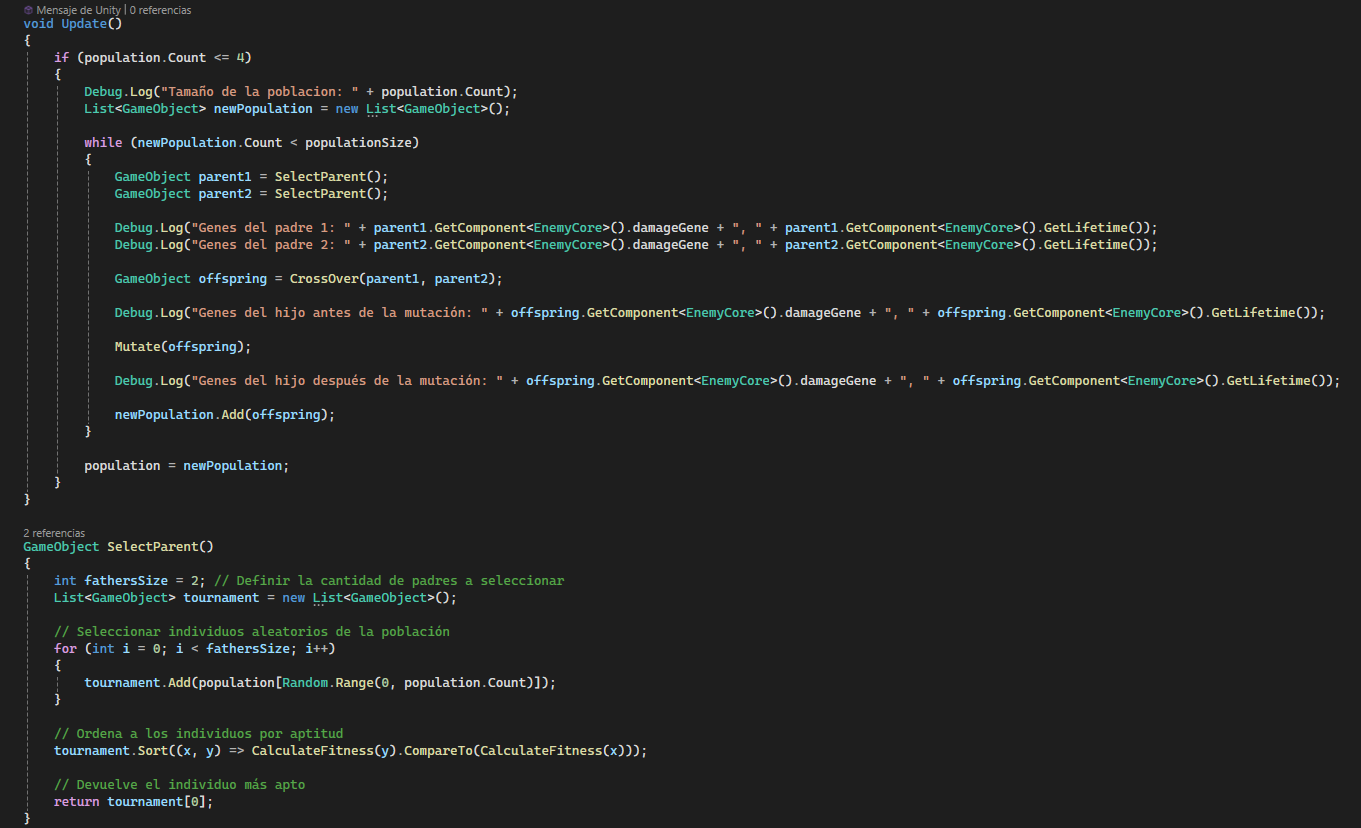
#### Detalles de Implementación

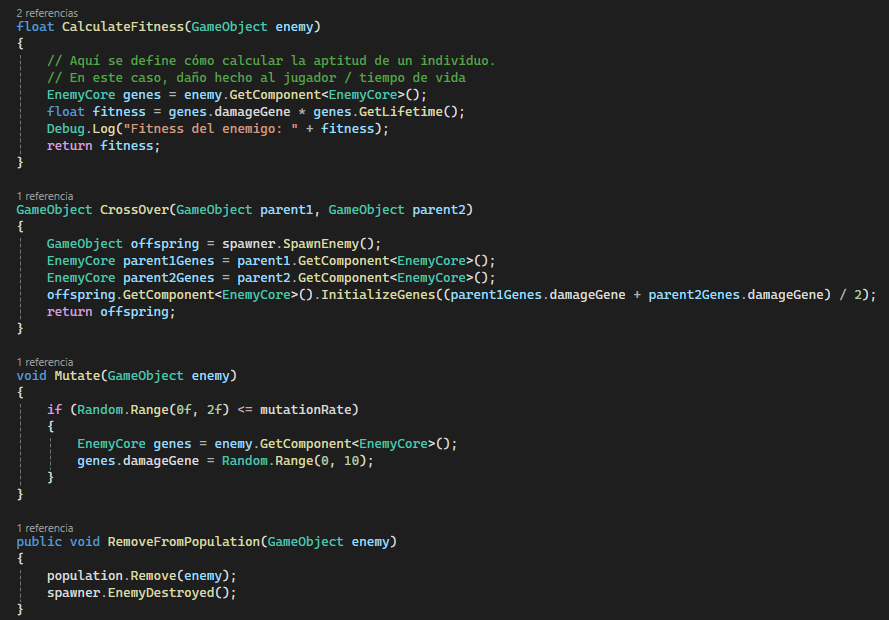
* **Variables:**
  + Transform player: Transform del jugador para calcular la distancia y orientar al agente.
  + DateTime lastTimePlayerHit: Almacena la última vez que el jugador fue golpeado.
  + int playerHitDelayInSeconds: Intervalo de tiempo en segundos entre ataques consecutivos.
* **Métodos Sobrecargados:**
  + OnStateEnter:
    - Se obtiene la posición del jugador al iniciar el estado.
  + OnStateUpdate:
    - El agente se orienta hacia el jugador.
    - Se calcula la distancia entre el jugador y el agente.
    - Si la distancia es mayor a 3.5 unidades, se desactiva el parámetro isAttacking del animador.
    - Si la distancia es menor o igual a 3.5 unidades, se realiza un Raycast para detectar si el jugador está dentro del rango de ataque (2 unidades).
    - Si el Raycast golpea al jugador y ha pasado el tiempo suficiente desde el último ataque, se aplica daño al jugador utilizando el componente PlayerController.
  + OnStateExit:
    - Actualmente vacío, pero puede ser utilizado para limpiar o resetear variables al salir del estado.

Por último, los estados Die y Hurt, definen respectivamente el estado de muerte del enemigo y el estado de dañado cuando recibe daño y se activan sus triggers, controlados ambos en el script EnemyCore.

Por otro lado, también se ha diseñado un algoritmo genético cuyo objetivo es mejorar las características de los enemigos del juego, veámoslo poco a poco:







El script GeneticAlgorithm implementa un algoritmo genético para la evolución de una población de enemigos en Unity. Utiliza técnicas de selección, cruce y mutación para mejorar las características de los enemigos a lo largo del tiempo. Este script se adjunta a un objeto de Unity y gestiona la creación y evolución de la población de enemigos.

#### Funcionalidad Principal

* **Inicialización de la Población:** Crea una población inicial de enemigos con genes aleatorios.
* **Evolución de la Población:** Cuando la población disminuye por debajo de un umbral, genera una nueva generación de enemigos mediante selección, cruce y mutación.
* **Selección de Padres:** Selecciona padres basados en su aptitud para generar descendencia.
* **Cruce y Mutación:** Combina los genes de dos padres para crear descendencia y aplica mutaciones aleatorias.

#### Detalles de Implementación

* **Variables:**
  + GameObject enemyPrefab: Prefab del enemigo utilizado para instanciar nuevos enemigos.
  + List<GameObject> population: Lista que almacena la población actual de enemigos.
  + int populationSize: Tamaño de la población.
  + float mutationRate: Tasa de mutación aplicada a los genes de la descendencia.
  + EnemySpawner spawner: Referencia al spawner de enemigos para instanciar nuevos enemigos.
* **Métodos:**
  + Start(): Inicializa la población con enemigos instanciados con genes aleatorios.
  + Update(): Supervisa el tamaño de la población y genera una nueva generación si es necesario.
  + SelectParent(): Selecciona los padres mediante un torneo de selección basado en la aptitud.
  + CalculateFitness(GameObject enemy): Calcula la aptitud de un enemigo basada en su daño y tiempo de vida.
  + CrossOver(GameObject parent1, GameObject parent2): Crea un nuevo enemigo combinando los genes de dos padres.
  + Mutate(GameObject enemy): Aplica una mutación aleatoria a los genes de un enemigo.
  + RemoveFromPopulation(GameObject enemy): Elimina un enemigo de la población y actualiza el spawner.

Para finalizar este apartado hay que explicar como son spawneados los enemigos generados por el algoritmo genético y para ello usamos el script EnemySpawner:

El script EnemySpawner gestiona la aparición de enemigos en Unity, asegurando que se generen de manera periódica y controlando el número total de enemigos activos en el juego. Se integra con el algoritmo genético (GeneticAlgorithm) para mantener la población de enemigos.

#### Funcionalidad Principal

* **Generación Periódica de Enemigos:** Genera enemigos a intervalos regulares.
* **Control del Número de Enemigos:** Asegura que el número de enemigos no exceda el tamaño de la población definida.
* **Interacción con Algoritmo Genético:** Utiliza la clase GeneticAlgorithm para conocer el tamaño de la población y ajustarse a ella.

#### Detalles de Implementación

* **Variables:**
  + GameObject enemyPrefab: Prefab del enemigo utilizado para instanciar nuevos enemigos.
  + Transform[] spawnPoints: Puntos de aparición predefinidos para los enemigos.
  + float spawnInterval: Intervalo de tiempo entre cada aparición de enemigo.
  + GeneticAlgorithm ga: Referencia al algoritmo genético para obtener el tamaño de la población.
  + float timer: Temporizador para controlar el intervalo de aparición.
  + int currentEnemyCount: Contador del número actual de enemigos en el juego.
* **Métodos:**
  + Start(): Inicializa el temporizador y obtiene la referencia al algoritmo genético.
  + Update(): Actualiza el temporizador y genera un enemigo si se ha alcanzado el intervalo de tiempo y el número de enemigos es menor al tamaño de la población.
  + SpawnEnemy():
    - Selecciona un punto de aparición aleatorio.
    - Instancia un nuevo enemigo en el punto seleccionado.
    - Activa el collider del enemigo y lo marca como activo.
    - Incrementa el contador de enemigos actuales.
  + EnemyDestroyed(): Decrementa el contador de enemigos cuando un enemigo es destruido.

# PANTALLA DE PUNTUACIÓN

En la pantalla de puntuación vemos una tabla con las puntuaciones de los 10 primeros jugadores de la siguiente forma:



Como vemos se permite introducir el nombre del jugador para, si supera las puntuaciones por defecto que se cargan a través de un .txt, aparecerá en dicha tabla y luego será redireccionado a la pantalla de inicio.

# RECURSOS EXTERNOS

* **Imagen de fondo en pantalla de título y puntuación:** <https://create.microsoft.com/es-es/features/ai-image-generator>
* **Asset para configuración gráfica:** <https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/Setup.html>
* **Fuente de texto para los títulos:** <https://www.1001fonts.com/ethnocentric-font.html>
* **Fuente de texto para el resto de elementos:** <https://www.1001fonts.com/ubuntu-font.html>
* **Mapa Nivel 1:** <https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-styled-modular-pack-82913>
* **Animaciones nivel 1:** <https://www.mixamo.com/#/?page=2&query=&type=Character>
* **Música nivel 1:** <https://pixabay.com/es/music/construir-escenas-dark-sci-fi-cyberpunk-112399/>
* **Tema principal:** <https://pixabay.com/es/music/optimista-cyborg-in-me-background-music-for-video-blog-promo-stories-188531/>
* **Pantalla de Game Over (Exclusiva para este proyecto por Álvaro Javier Rodríguez Naranjo):** <https://www.linkedin.com/in/%C3%A1lvaro-javier-rodriguez-naranjo-a68bb513b/?originalSubdomain=es>