

Raúl Toledo Procopio  
Desarrollo de Aplicaciones Multiplataforma  
TFG

Índice

[Índice de tablas e imágenes 2](#_Toc200582926)

[Introducción 3](#_Toc200582927)

[Descripción 3](#_Toc200582928)

[Instalación 3](#_Toc200582929)

[Documentación de código 4](#_Toc200582930)

[Diario de desarrollo 18](#_Toc200582931)

[Pruebas 19](#_Toc200582932)

[Distribución 19](#_Toc200582933)

[Conclusiones 19](#_Toc200582934)

[Bibliografía 20](#_Toc200582935)

# Índice de tablas e imágenes

Imágenes:

[Ilustración 1 Parámetros de Player 4](#_Toc200581994)

[Ilustración 2 Ready() Player 4](#_Toc200581995)

[Ilustración 3 PhysicsProcess Player 5](#_Toc200581996)

[Ilustración 4 PhysicsProcess1 Player 5](#_Toc200581997)

[Ilustración 5 PhysicsProcess2 Player 5](#_Toc200581998)

[Ilustración 6 Attack() Player 6](#_Toc200581999)

[Ilustración 7 UpdateAnimation(direction) Player 6](#_Toc200582000)

[Ilustración 8 Variables Enemies Tier1 7](#_Toc200582001)

[Ilustración 9 Ready() Enemies Tier1 7](#_Toc200582002)

[Ilustración 10 \_PhysicsProcess Enemies Tier1 8](#_Toc200582003)

[Ilustración 11 TakeDamage(damage) Enemies Tier1 8](#_Toc200582004)

[Ilustración 12 FireBallScene Enemies Tier2 9](#_Toc200582005)

[Ilustración 13 ShootFireBall() Enemies Tier2 9](#_Toc200582006)

[Ilustración 14 State enum {Patrol, Attack, Death} Enemies Tier3 9](#_Toc200582007)

[Ilustración 15 DetectionArea Enemies Tier3 10](#_Toc200582008)

[Ilustración 16 PatrolBehavior() Enemies Tier3 10](#_Toc200582009)

[Ilustración 17 AttackBehavior() Enemies Tier3 10](#_Toc200582010)

[Ilustración 18 State enum Boss 11](#_Toc200582011)

[Ilustración 19 Lógica de patrulla vertical Boss 11](#_Toc200582012)

[Ilustración 20 Track AudioManager 12](#_Toc200582013)

[Ilustración 21 Ready() AudioManager 12](#_Toc200582014)

[Ilustración 22 PlayForLevel() AudioManager 12](#_Toc200582015)

[Ilustración 23 GetGroupForLevel() AudioManager 13](#_Toc200582016)

[Ilustración 24 Variables GameState 13](#_Toc200582017)

[Ilustración 25 ChangeHp(amount) GameState 14](#_Toc200582018)

[Ilustración 26 ChangeAmmo(amount) GameState 14](#_Toc200582019)

[Ilustración 27 Reset() GameState 14](#_Toc200582020)

[Ilustración 28 SaveGame(username, levelPath) SaveManager 15](#_Toc200582021)

[Ilustración 29 LoadGame(username) SaveManager 15](#_Toc200582022)

[Ilustración 30 DeleteGame(username) SaveManager 16](#_Toc200582023)

[Ilustración 31 SessionManager 16](#_Toc200582024)

[Ilustración 32 Variables Crono 16](#_Toc200582025)

[Ilustración 33 \_Process(delta) Crono 17](#_Toc200582026)

[Ilustración 34 StopTimer(submitScore) Crono 17](#_Toc200582027)

[Ilustración 35 SubmitScoreAsync(timeSeconds) Crono 18](#_Toc200582028)

Tablas:

[Tabla 1 Diario de desarrollo 18](#_Toc200582095)

# Introducción

La introducción del juego se encuentra en el documento de diseño del juego, adjuntado al proyecto como GDD.pdf o pulsando este [enlace](GDD.docx).

# Descripción

Al ejecutar el juego, pasaremos a una pantalla de login, en la que tendemos que logearnos o registrarnos en caso de que no tengamos cuenta. Una vez hecho esto, pasaremos al menú principal para decidir si queremos jugar una nueva partida o continuar con la que ya tenemos.  
  
Además de los menús, tenemos 10 escenas principales que conforman los niveles jugables.  
  
El juego cuenta con una estructura de carpetas limpia y clara, para poder movernos por ella de una manera simple y ordenada. Entre ellas se distinguen carpetas para los niveles, los personajes, los enemigos, los proyectiles…

# Instalación

Para instalar el juego, solo necesitamos descargarlo de itch.io en el siguiente enlace:  
  
<https://github.com/RaulToledoProcopio/JuegoPSP>  
  
o descargarnos Godot, .NET y clonar el siguiente repositorio a nuestro pc:  
  
<https://github.com/RaulToledoProcopio/JuegoPSP>

# Documentación de código

Vamos a realizar la documentación de código, para ello vamos a centrarnos en primer lugar, en la parte más importante de los scripts que sean relevantes.

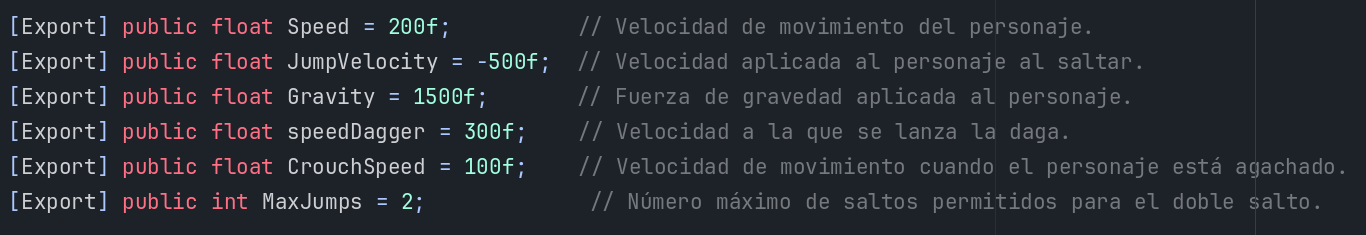
Player.cs – Este es el script del personaje principal  
  
Puntos clave:  
  
**[Export] Speed, JumpVelocity, Gravity, CrouchSpeed, MaxJumps** - Parámetros ajustables que controlan movimiento, salto, agachado…  
  


Ilustración 1 Parámetros de Player

**\_Ready() – Cacheo de nodos** - Se obtienen referencias a nodos de animación, UI y sonidos para no buscarlos en cada frame.  
  


Ilustración 2 Ready() Player

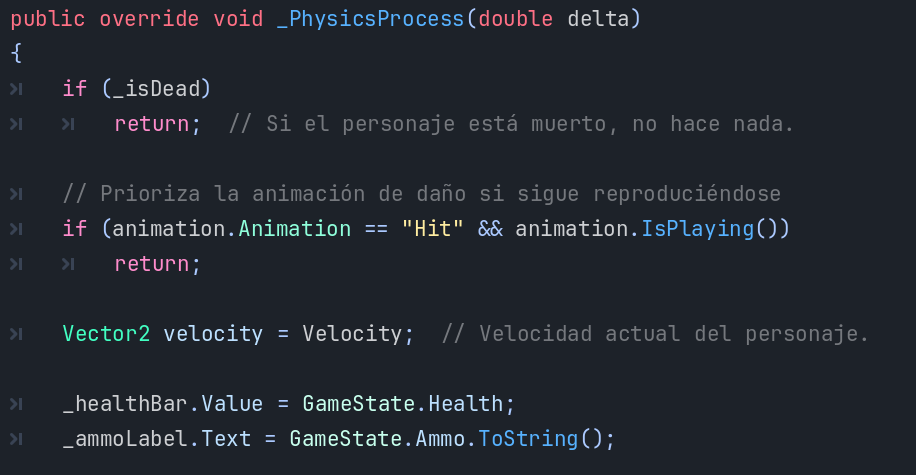
**\_PhysicsProcess(delta) – Movimiento y lógica principal** - Aplica gravedad, lee input (mover, saltar, agacharse, atacar), lanza proyectiles y actualiza la animación.  
  


Ilustración 3 PhysicsProcess Player



Ilustración 4 PhysicsProcess1 Player

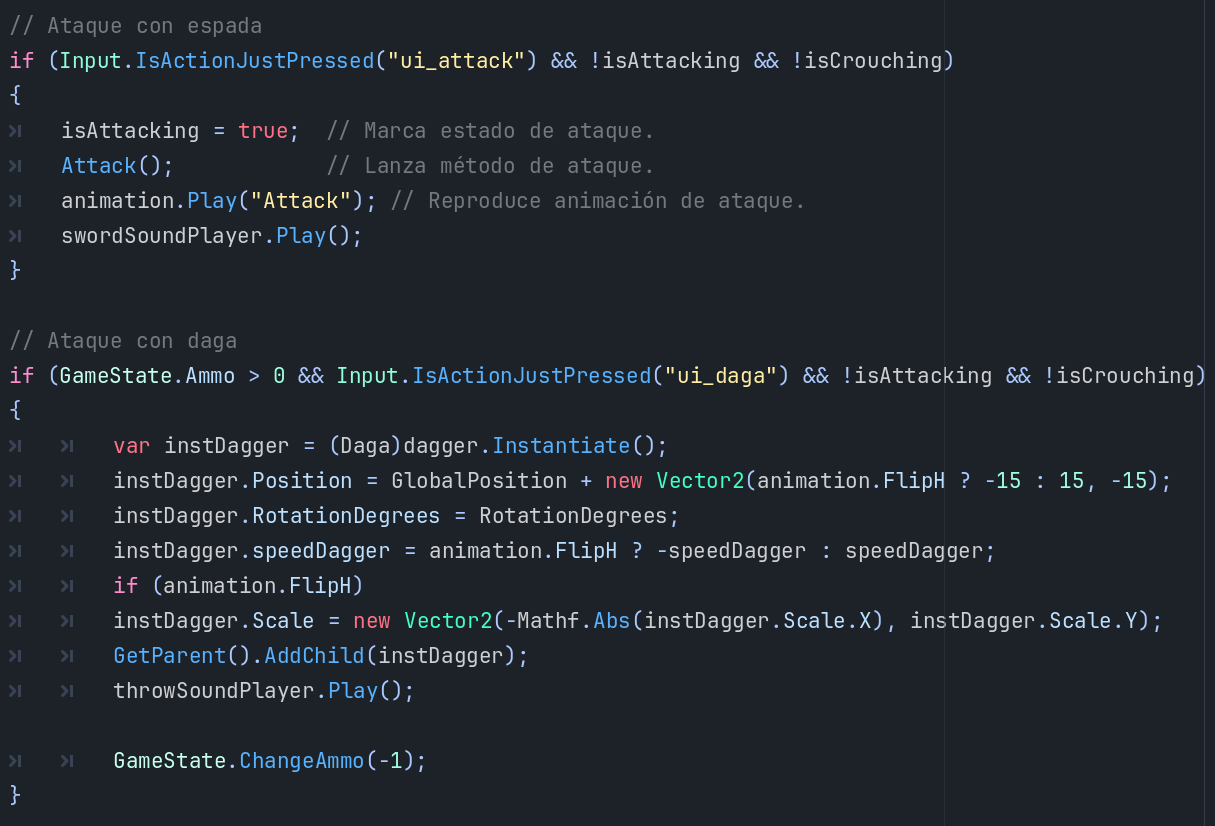


Ilustración 5 PhysicsProcess2 Player

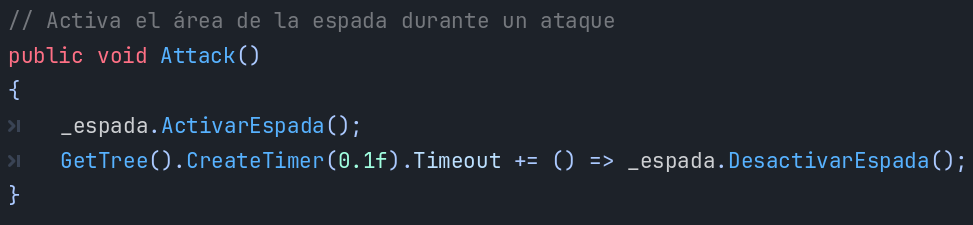
**Attack()** – Activación temporal del collider de la espada, permite detectar colisiones de ataque durante un breve instante. ****

Ilustración 6 Attack() Player

**UpdateAnimation(direction) –** Cambia el estado y reproduce la animación adecuada según salud, ataque, salto, etc.  
  
****

Ilustración 7 UpdateAnimation(direction) Player

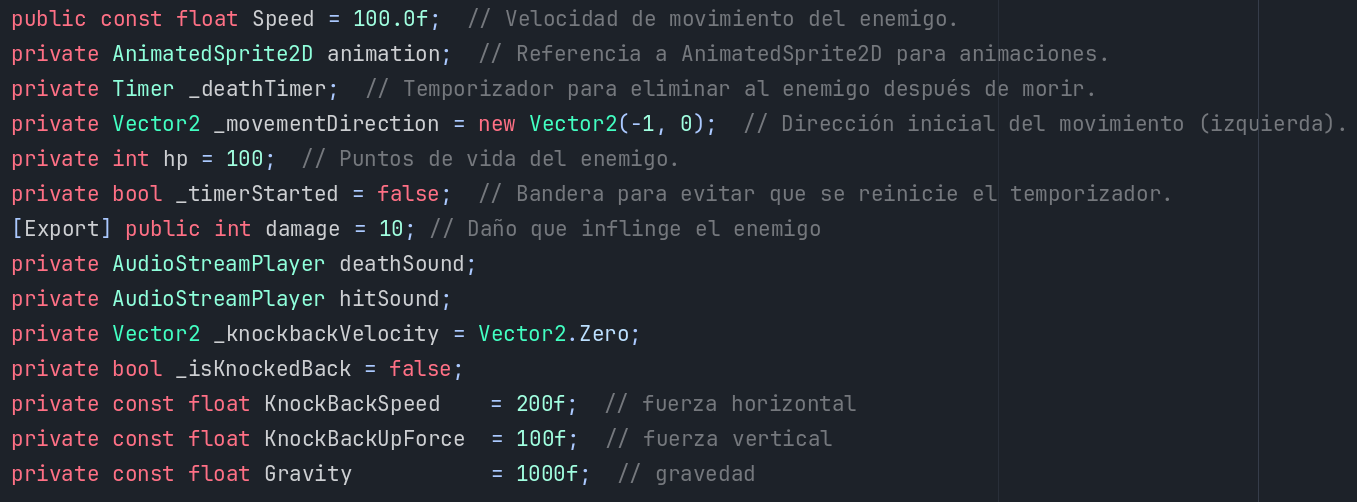
**Enemy1** – Ejemplo Script de enemigos de tier1, aplicable a Enemy3 y 7.  
Puntos clave:  
  
**Variables** - Velocidad, vida y daño base del enemigo, así como todas las variables exportables o constantes.  
  


Ilustración 8 Variables Enemies Tier1

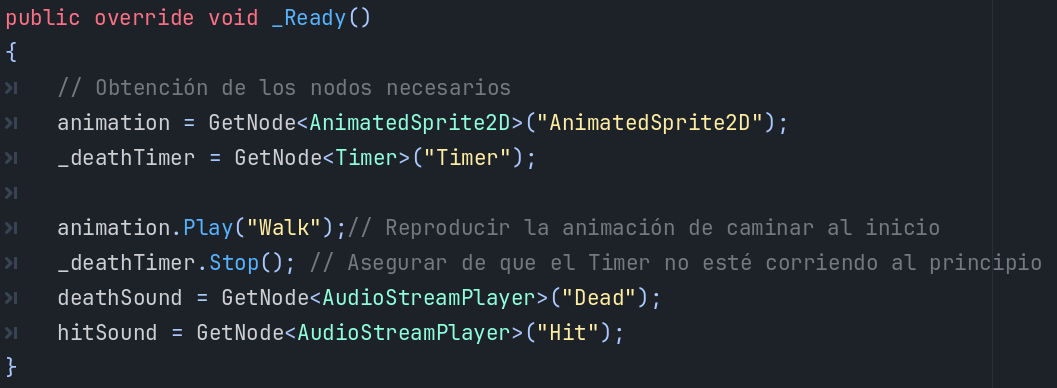
**\_Ready() –** Iniciar animación y sonidos, arranca el “Walk” y cachea AnimatedSprite2D, Timer, AudioStreamPlayer.  
  


Ilustración 9 Ready() Enemies Tier1

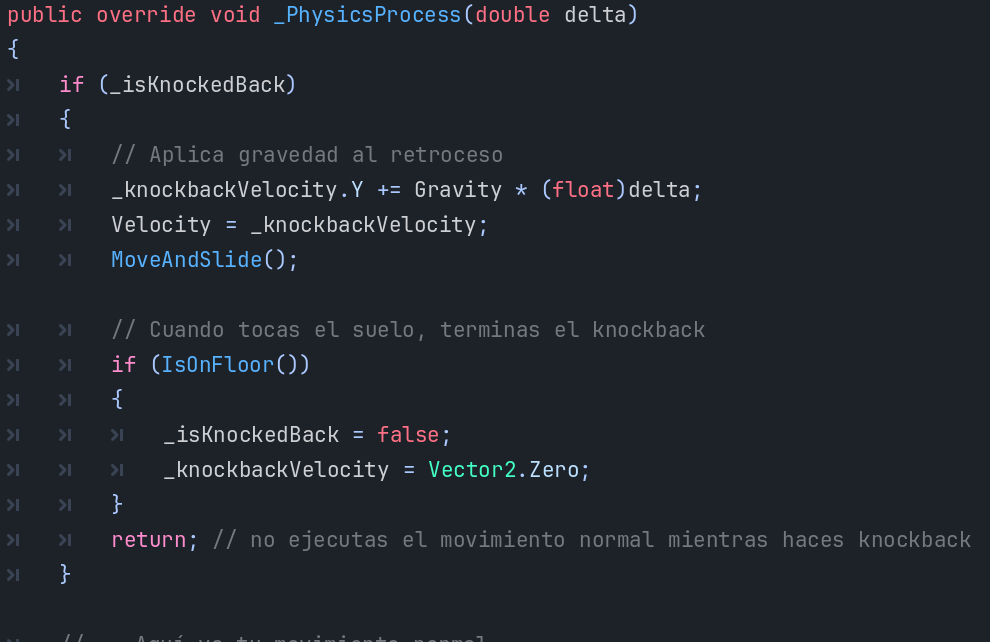
**\_PhysicsProcess(delta) – Movimiento con rebote** - Se desplaza en línea recta, invierte dirección al chocar con muros y aplica knockback cuando recibe daño.  
  


Ilustración 10 \_PhysicsProcess Enemies Tier1

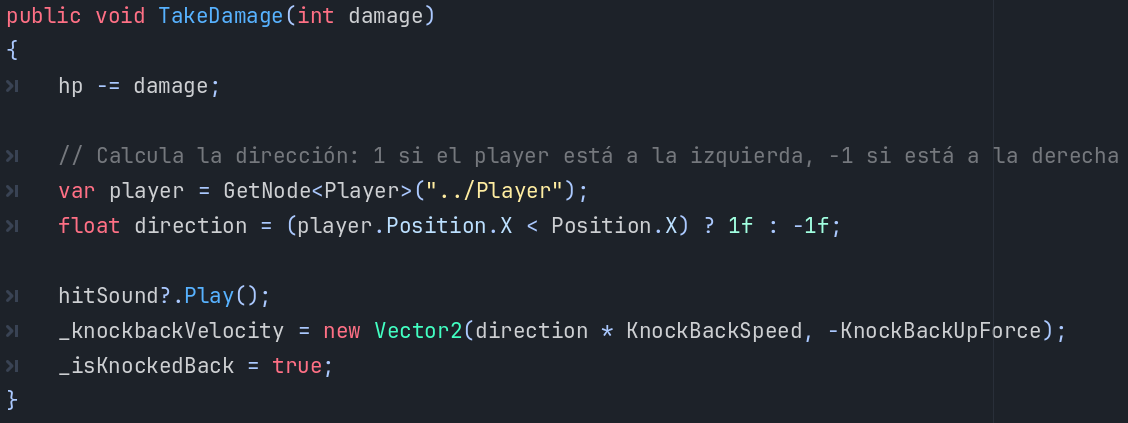
**TakeDamage(damage) – Retroceso y detección de muerte** - Reduce hp, reproduce hitSound, aplica fuerza de retroceso y marca \_isKnockedBack.  
  


Ilustración 11 TakeDamage(damage) Enemies Tier1

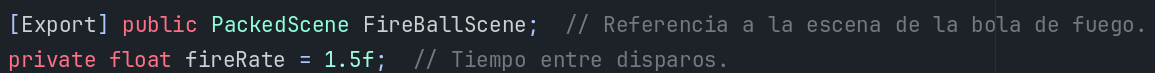
**Enemy2** – Ejemplo Script de enemigos de tier2, aplicable a Enemy5  
Puntos clave:  
  
Además de tener algunos métodos en común con el anterior añadimos la opción de lanzar un proyectil basado en una escena que se lanza desde una posición relativa al propio enemigo.  
  
**FireBallScene + fireRate** - Referencia a escena de proyectil y cadencia de disparo entre fases de “Idle” y “Attack”.  
  


Ilustración 12 FireBallScene Enemies Tier2

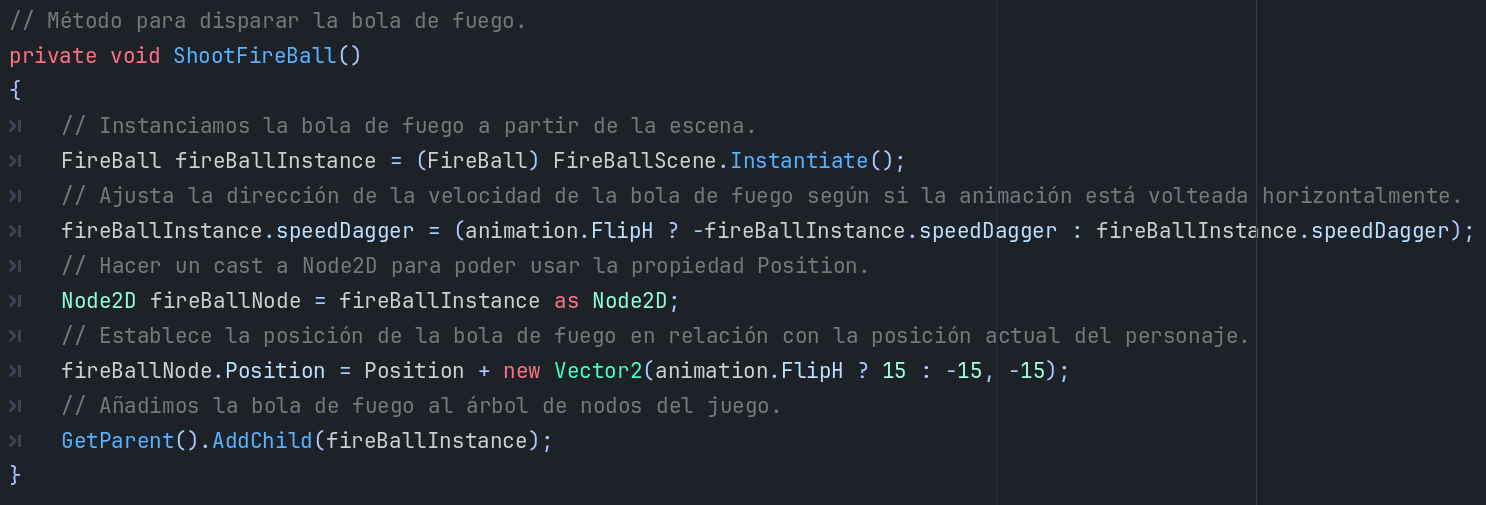
**ShootFireBall() – Instanciación de proyectil** - Ajusta posición y velocidad del FireBallScene según orientación.  
  


Ilustración 13 ShootFireBall() Enemies Tier2

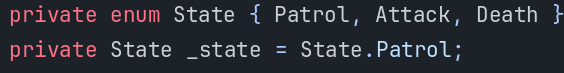
**Enemy4** – Ejemplo Script de enemigos de tier3, aplicable a Enemy6  
  
Puntos clave:  
  
En estos enemigos, la mecánica que añadimos es que tienen un área de detección que detecta al player, cuando esto sucede dejan de patrullar y hacen una animación de ataque al mismo.  
  
**State enum {Patrol, Attack, Death}** - Define el comportamiento: patrullar, atacar al detectar jugador, o morir.  
  


Ilustración 14 State enum {Patrol, Attack, Death} Enemies Tier3

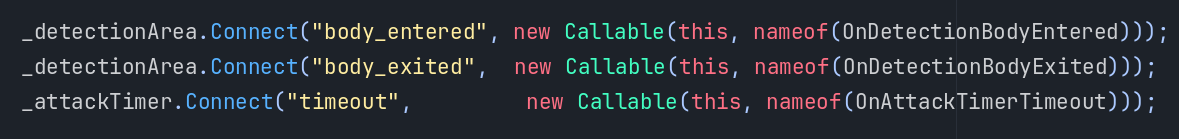
**DetectionArea + Timer de ataque** - Usa un Area2D para entrar/salir de rango y un Timer para cadencia de golpes.  
  


Ilustración 15 DetectionArea Enemies Tier3

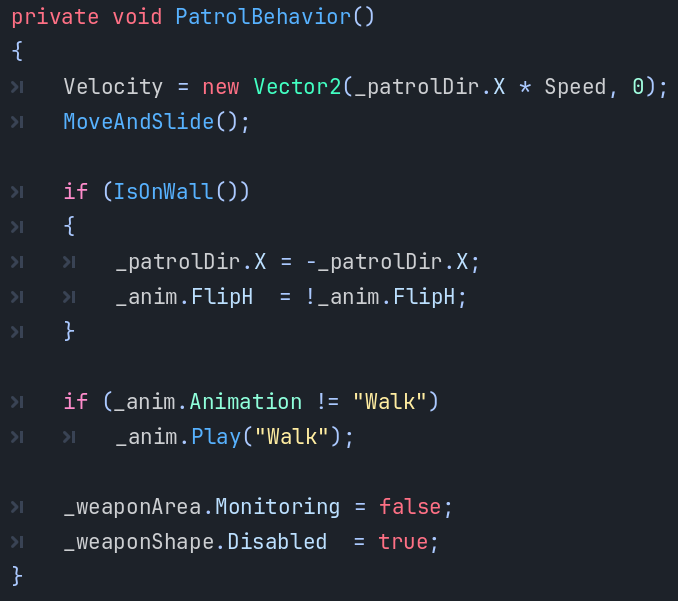
**PatrolBehavior()** - mueve y rebota en muros.  
  


Ilustración 16 PatrolBehavior() Enemies Tier3

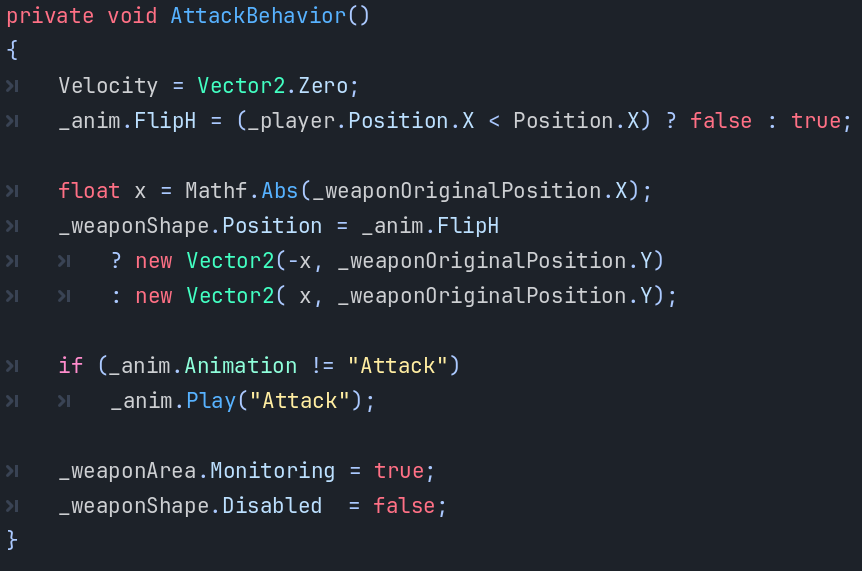
**AttackBehavior()** - detiene, invierte sprite, activa el collider del ataque.  
  


Ilustración 17 AttackBehavior() Enemies Tier3

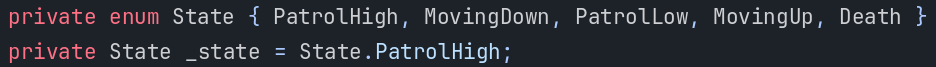
**Boss** – Script de enemigos de tier4, aplicable parcialmente a Miniboss ya que es una mezcla de mecánicas de todos los enemigos anteriores.  
  
En este caso, hemos copiado las mecánicas del resto de enemigo, con la novedad de que cada ciertos segundos el boss baja y hace un barrido a diferente altura.  
  
**State enum {PatrolHigh, MovingDown, PatrolLow, MovingUp, Death}** - Controla patrones de patrulla vertical y ataques de rayo.  
  


Ilustración 18 State enum Boss

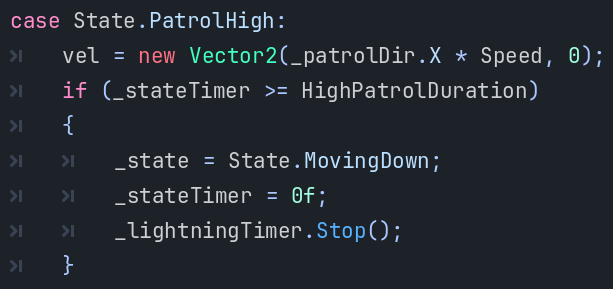
**Lógica de patrulla vertical** - Cambia State tras HighPatrolDuration y mueve hacia LowY, regresando luego arriba.  
  


Ilustración 19 Lógica de patrulla vertical Boss

Ahora vamos a proceder a la documentación de otra parte importante del código, como son los singleton. Los singleton son scripts que son cargados automáticamente y son accesible globalmente desde cualquier parte del juego.  
  
**AudioManager.cs** - Controlar la música de fondo del juego en función del nivel.

Puntos clave:

**\_tracks** : Guarda todas las pistas en memoria para evitar cargas repetidas.

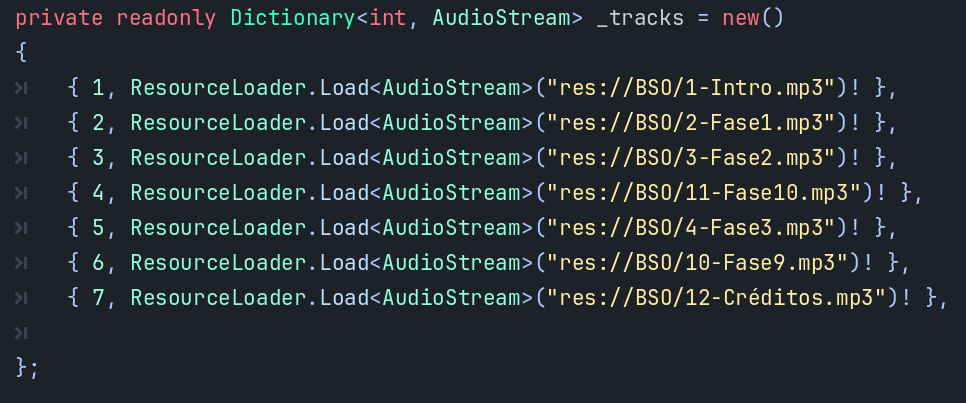


Ilustración 20 Track AudioManager

**\_Ready() y \_player** : Se cachea la referencia a AudioStreamPlayer para optimizar llamadas.

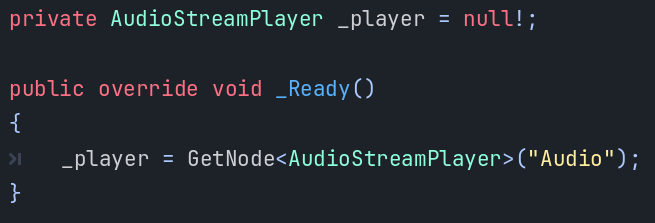


Ilustración 21 Ready() AudioManager

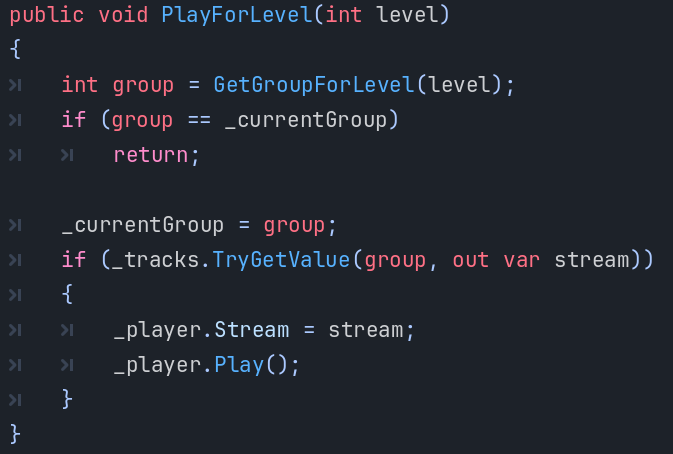
**PlayForLevel()** : Llama a GetGroupForLevel para saber qué pista corresponde al nivel y solo cambia la música si el grupo es distinto al actual.  
  


Ilustración 22 PlayForLevel() AudioManager

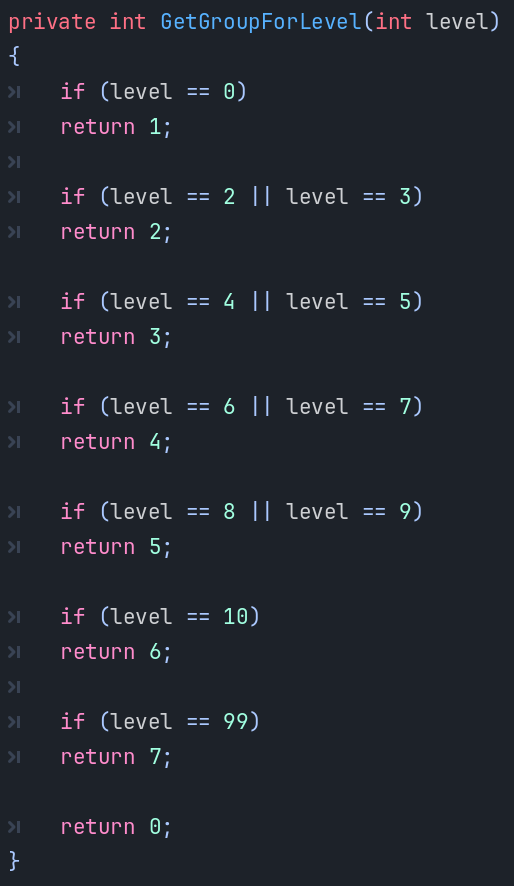
**GetGroupForLevel()** : Centraliza la lógica de asignación nivel→grupo, fácil de ajustar.  
  


Ilustración 23 GetGroupForLevel() AudioManager

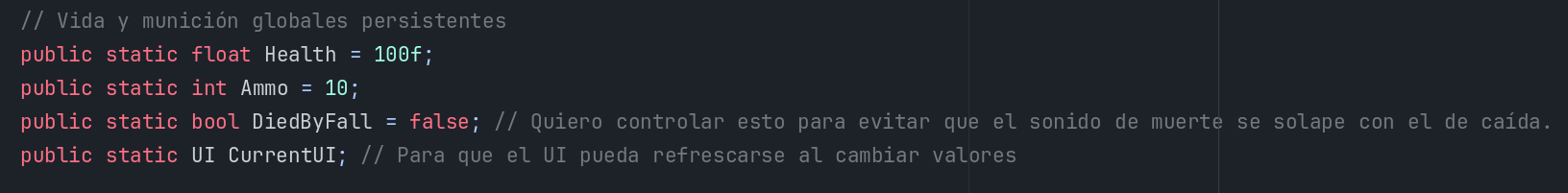
**GameState.cs** - Mantener el estado global del jugador entre escenas.  
Puntos clave:  
  
Variables:  
  
**Health** - Vida global del jugador, persistente entre escenas.  
**Ammo** - Munición global del jugador, persistente.  
**DiedByFall** - Indica si el jugador murió por caída, para gestionar efectos de sonido.  
**CurrentUI** - Referencia al controlador de la interfaz para actualizar valores.  
  


Ilustración 24 Variables GameState

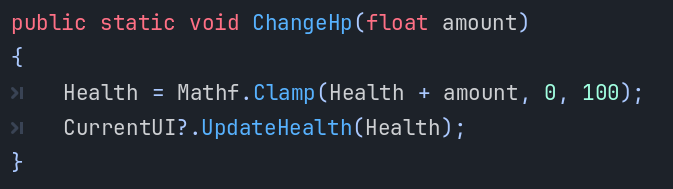
**ChangeHp(amount)** - Ajusta Health con clamp entre 0–100 y notifica a la UI.  
  


Ilustración 25 ChangeHp(amount) GameState

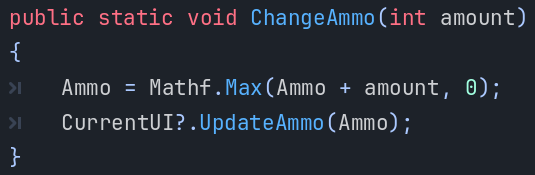
**ChangeAmmo(amount)** - Ajusta Ammo con mínimo cero y actualiza la UI.  
  


Ilustración 26 ChangeAmmo(amount) GameState

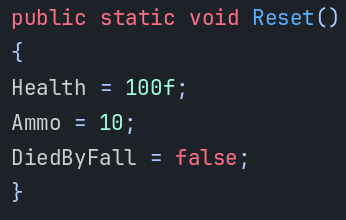
**Reset()** - Revierte Health, Ammo y DiedByFall a valores iniciales.  
  


Ilustración 27 Reset() GameState

**SaveManager.cs** - Gestionar el guardado y carga de partidas en disco.  
Puntos Clave:  
  
**SaveGame(username, levelPath)** - Serializa SaveData a JSON y lo escribe en user://save\_{username}.json.  
  


Ilustración 28 SaveGame(username, levelPath) SaveManager

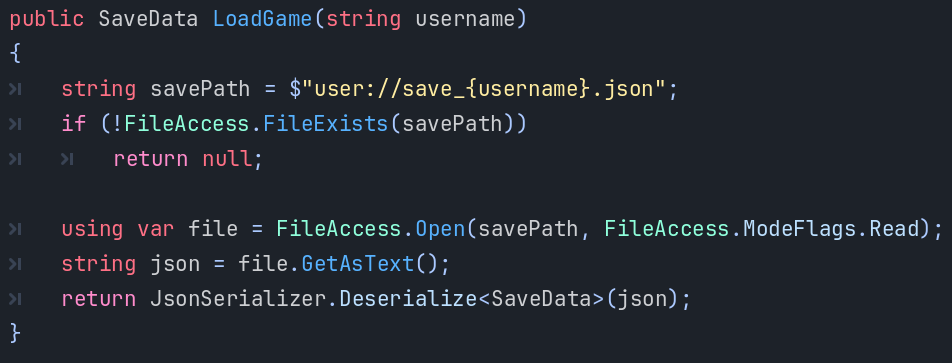
**LoadGame(username)** - Lee el JSON de user://save\_{username}.json y deserializa a SaveData.  
  


Ilustración 29 LoadGame(username) SaveManager

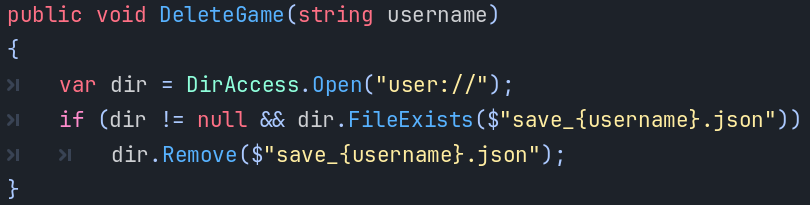
**DeleteGame(username)** - Elimina el archivo de guardado si existe.  
  


Ilustración 30 DeleteGame(username) SaveManager

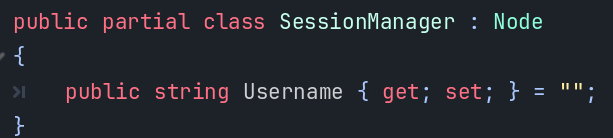
**SessionManager.cs** - Almacenar el nombre de usuario durante la sesión.  
Puntos clave:  
  
**Username** - Almacena el nombre de usuario de la sesión actual.  
**Acceso global** - Permite leer el usuario desde cualquier nodo.  
  


Ilustración 31 SessionManager

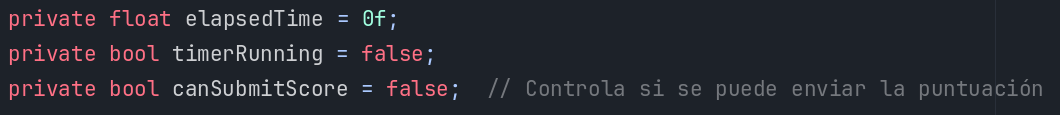
**Crono.cs** - Gestionar el cronómetro de juego y enviar puntuaciones online.  
  
**elapsedTime** - Acumula el tiempo transcurrido mientras el cronómetro corre.  
**timerRunning** - Controla si el cronómetro debe actualizarse en \_Process.  
**canSubmitScore** - Indica si está permitido enviar la puntuación al detener el cronómetro.  
  


Ilustración 32 Variables Crono

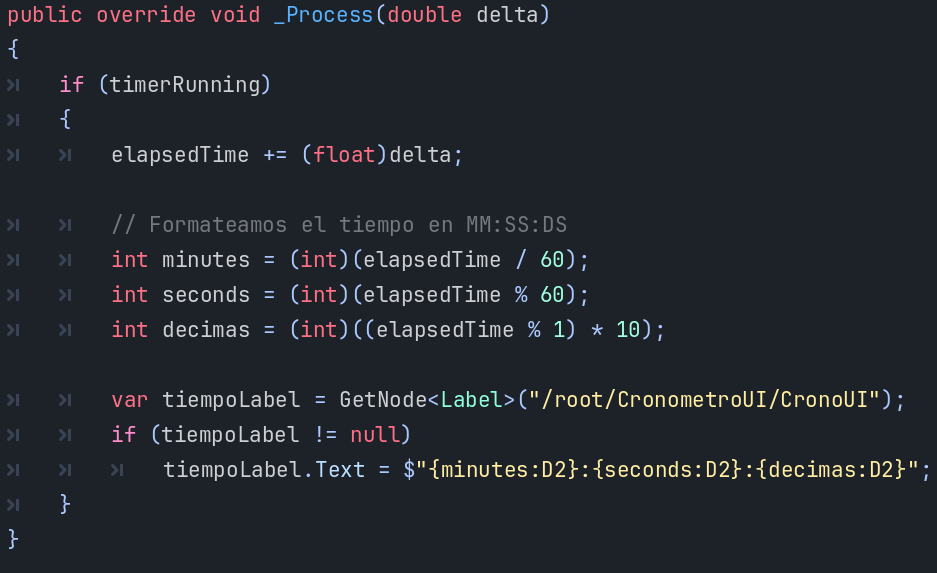
**\_Process(delta)** - Formatea elapsedTime a MM:SS:DS y actualiza el Label.  
  


Ilustración 33 \_Process(delta) Crono

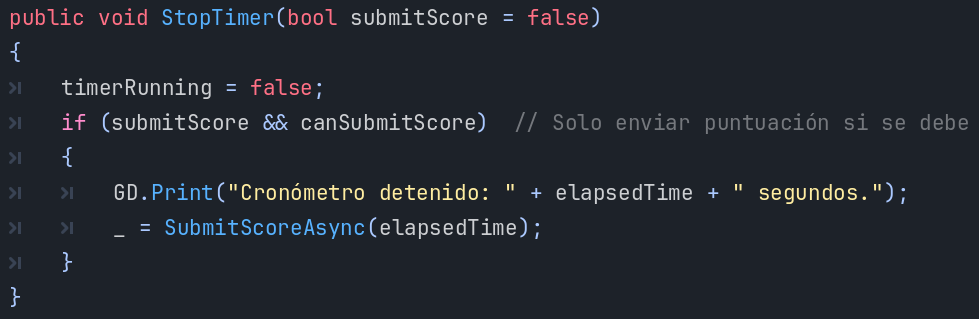
**StopTimer(submitScore)** - Detiene el cronómetro y, si procede, envía la puntuación.  
  


Ilustración 34 StopTimer(submitScore) Crono

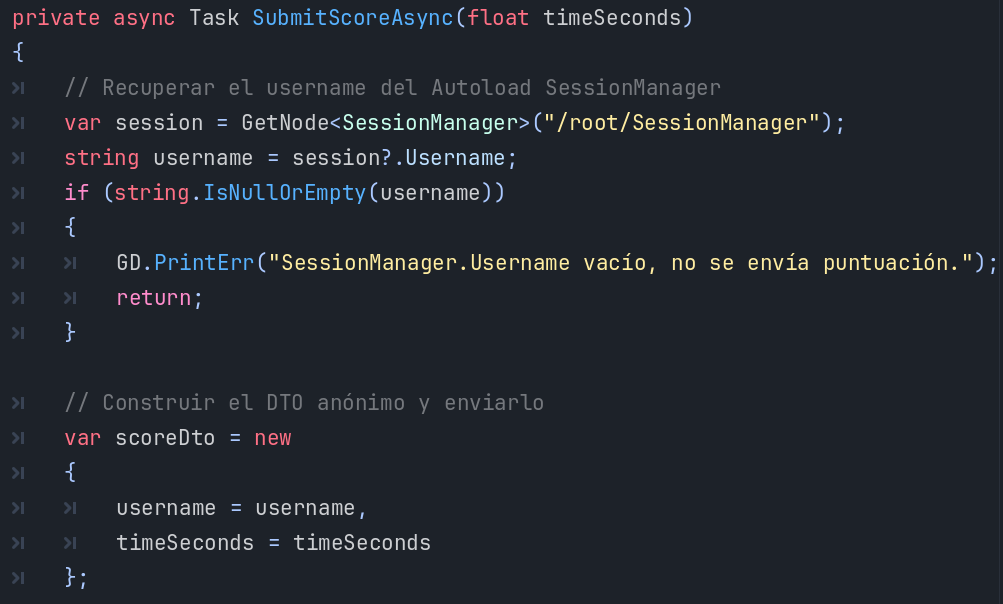
**SubmitScoreAsync(timeSeconds)** - Obtiene Username de SessionManager, construye DTO y usa HttpClient para POST.  
  


Ilustración 35 SubmitScoreAsync(timeSeconds) Crono

# Diario de desarrollo

Tabla 1 Diario de desarrollo

|  |  |  |
| --- | --- | --- |
|  | Tarea | Fecha |
| 1 | Diseño del documento | 20/27 marzo |
| 2 | Creación de personaje | 20/27 marzo |
| 3 | Creación de los primeros niveles | 27 marzo/3 abril |
| 4 | Demo viable | 3/10 abril |
| 5 | Persistencia de datos para el login y el registro | 10/24 abril |
| 6 | Persistencia de datos para las puntuaciones | 10/24 abril |
| 7 | Diseño de la BSO y efectos de sonido | 24 abril/8 mayo |
| 8 | Inclusión de nuevos niveles | 8/22 mayo |
| 9 | Depuración de bugs | 22/29 mayo |
| 10 | Inclusión de nuevos enemigos | 29 mayo/10 junio |
| 11 | Documentación | 5/12 junio |

# Pruebas

Al ser un videojuego, he realizado las pruebas manuales sobre el login y el registro, de las cuales adjuntos capturas y están documentadas en la documentación de la API, incluida en esta misma carpeta en el documento API.PDF o bien en el repositorio de la misma como Readme.md y he distribuido el juego a varias personas de confianza para que hagan de betatesters.

# Distribución

Juego distribuido de manera gratuita mediante Itch.io

# Conclusiones

**Comparativa:**

La jugabilidad final del producto ha resultado ser exactamente la esperada; el juego se ha desarrollado tal y como lo había concebido desde un principio.

Durante el proceso, he aprendido nuevas herramientas y técnicas, como el uso de TileMapLayers y TileSets para pintar niveles, algo que no dominaba en la primera entrega.

También he logrado implementar nuevas mecánicas en los enemigos, algunas con mayor complejidad técnica, como activar un Area2D que haga daño solo durante una animación específica, y que además se oriente hacia la posición del jugador.

**Mejoras futuras:**

Dentro del propio juego, en el menú final, hay una sección dedicada a las mejoras futuras previstas. Entre ellas, me gustaría destacar la intención de añadir una historia más desarrollada con un lore definido, así como cinemáticas más elaboradas que complementen la experiencia actual.

# Bibliografía

<https://docs.godotengine.org>  
<https://docs.godotengine.org/es/4.x/classes/index.html#csharp-api>  
<https://learn.microsoft.com/es-es/dotnet/csharp/>  
<https://learn.microsoft.com/es-es/dotnet/api/>  
<https://chatgpt.com/>  
<https://itch.io/>  
<https://suno.com/home>  
<https://www.bing.com/images/create?setlang=es>