

|  |  |
| --- | --- |
| Programación de Juegos 2D en Unity  STEEL JUSTICE |  |
| Juan Alberto Domínguez Vázquez  Saúl Rodríguez Naranjo  Universidad de Huelva | Ingeniería Informática | Programación de Juegos (2024) |  |

**ÍNDICE**

[INTRODUCCIÓN 3](#_Toc165123825)

[PANTALLA DE TÍTULO 3](#_Toc165123826)

[PRIMER NIVEL 3](#_Toc165123827)

[SEGUNDO NIVEL 3](#_Toc165123828)

[PANTALLA DE PUNTUACIÓN 3](#_Toc165123829)

[RECURSOS EXTERNOS 3](#_Toc165123830)

# INTRODUCCIÓN

Steel Justice es un videojuego en 2D basado en la jugabilidad de juegos retro como pueden ser Megaman, Metal Slug o Contra. El contexto en el que se desarrolla es aquel del de una sociedad futurista con tintes cyberpunk, en el que encarnamos a un robot de combate llamado Legión que ha de luchar contra los criminales de la ciudad.

# PANTALLA DE TÍTULO

ASDASDASD

# PRIMER NIVEL

En este primer nivel nos encontramos en un laboratorio secreto subterráneo donde deberemos matar a todos los enemigos presentes.

Primero nos centraremos en el diseño del personaje principal y es que nuestro robot Legión, es representado dentro de Unity con el objeto Jugador, el cual usa para poder moverse, el script MovimientoJugador, que cuando pulsamos las teclas A o D, se moverá a izquierda o derecha respectivamente, así como cuando pulsemos la barra espaciadora, se producirá la acción de salto y cuando pulsemos el click izquierdo del ratón comenzará a disparar bolas de plasma. Cabe añadir que, para dotar de física al personaje, se le han añadido los componentes de Unity BoxCollider2D y RigidBody2D y que también el script posee varias variables serializadas como el poder de salto la velocidad y la capa en la que se muestra al personaje, que podrán ser modificadas en caso de necesidad.

Si volvemos a la acción de disparo, esta es gestionada por el script AtaqueJugador, el cual tiene varias variables serializadas como el tiempo que debe de pasar entre cada disparo, el punto desde el cual se produce el disparo, un array de objetos de tipo Proyectil, que veremos más adelante, y por último, el sonido de disparo. Este script para la creación de los disparos hace uso de la técnica denominada Object Pooling, la cual consiste en crear un array de objetos para nuestros disparos, en este caso del tipo Proyectil, a continuación, se crea un punto desde el cuál se va a disparar, que ha de coincidir con la animación de disparo de nuestro personaje, y vamos recorriendo dicho array para en cada ejecución “devolver” un proyectil que se encuentre disponible dentro de los tiempos de enfriamiento entre cada acción de disparo. Con esta técnica nos evitamos tener que estar creando y destruyendo instancias del objeto proyectil cada vez que el jugador quiera disparar y, por tanto, disminuimos significativamente el uso de recursos del juego y aumentamos su eficiencia al ejecutarse.

El script Proyectil, se encarga de definir el comportamiento de cada disparo cuando colisiona con otro objeto gracias al método, OnCollisionEnter2D que se llama cuando el proyectil entra en contacto con otro Collider2D. Establece la variable golpeado en true para evitar que el proyectil cause más daño, desactiva el collider del proyectil y reproduce una animación de explosión a través del Animator. Si el objeto con el que colisiona tiene la etiqueta "Enemy", reduce su salud llamando al método TakeDamage() del componente Health del objeto colisionado. También dentro del método update, si el proyectil ha golpeado algo, simplemente retorna y no hace nada más. Si no ha golpeado nada, mueve el proyectil en la dirección establecida multiplicando la velocidad por el tiempo transcurrido desde el último fotograma. Por último, lleva un seguimiento del tiempo de vida del proyectil y si supera cierto límite (5 segundos en este caso), desactiva el GameObject del proyectil.

En cuanto a su implementación, se ha optado por diseñar a mano la estructura del mismo, partimos de una primera habitación en la que se nos presentan trampas que deberemos de sortear y el primer enemigo a batir, el cual tiene implementado un algoritmo A\* para el seguimiento de caminos, que en este caso su objetivo es perseguir al jugador hasta acabar con él. Dicho algoritmo se ha utilizado con la ayuda de la librería astarpathfindingproyect, la cuál nos ofrece la posibilidad de dar esta cualidad a nuestros objetos del juego de manera sencilla, simplemente añadiendo varios scripts al mismo y configurando el comportamiento, alcance y velocidad de seguimiento.

# SEGUNDO NIVEL

SADASDASDASD

# PANTALLA DE PUNTUACIÓN

ADASDASDASD

# RECURSOS EXTERNOS

* **Imagen de fondo en pantalla de título y puntuación:** <https://wall.alphacoders.com/big.php?i=1283852>
* **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>
* **Sprites Nivel 1:** <https://foozlecc.itch.io/sci-fi-lab-tileset-decor-traps/download/eyJleHBpcmVzIjoxNzEyODM2MDU2LCJpZCI6OTE1MzA5fQ%3D%3D.FB1dXAzmMkvJm1hZsBClPMp7v8M%3D>
* **Animaciones nivel 1:** <https://creativekind.itch.io/nightborne-warrior>, <https://clembod.itch.io/bringer-of-death-free>, <https://wuhuli.itch.io/robot-sprite-brawlbot>
* **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/>
* **Sprites, música y animaciones Nivel 2:** <https://assetstore.unity.com/packages/2d/environments/warped-city-2-200208>
* **Efectos de sonido nivel 1 y 2:** <https://pixabay.com/es/sound-effects/>, <https://www.mediafire.com/file/wszdta62jwdgyru/Audio.zip/file>