Instituto Puig Castellar
Ciclo formativo: Desarrollo de Aplicaciones Multiplataforma (DAM)
Grado: CFGS
Crédito de Síntesis / Proyecto intermodular

Outer Rim Smugglers

Videojuego 2D de acción ambientado en Star Wars con portal web de descarga

Autor/a/es: Alfons Figuereo González
Tutor: Daniel Colomer

Curso: 2025-2026
Fecha de entrega: 17/05/2026

Resumen del proyecto

Outer Rim Smugglers es un videojuego 2D de acción desarrollado con Godot 4 y programado en GDScript, ambientado en el universo de Star Wars, concretamente en el planeta Tatooine. El jugador encarna a Han Solo y debe abrirse paso por un pueblo del Outer Rim esquivando y eliminando a sus enemigos: Stormtroopers, Sandtroopers, Rodianos, y el jefe final Boba Fett. El combate es en tiempo real, con disparos que se desplazan en línea recta por los ejes X o Y. El juego incorpora un sistema de inventario, cofres repartidos por el mapa con objetos coleccionables y un mercader al que venderlos. Todos los sprites del juego, salvo los tiles del mapa (obtenidos de forma gratuita en internet), fueron diseñados a mano con Aseprite. El proyecto incluye también una página web de una sola página (SPA), desarrollada con HTML y CSS puro y alojada en Vercel, que actúa como portal oficial de descarga del juego. El proyecto ha supuesto aproximadamente 150 horas de trabajo repartidas entre el diseño gráfico, el desarrollo del videojuego y la creación de la web.

Palabras clave

Godot 4, GDScript, videojuego 2D, Star Wars, Tatooine, tilemap, inventario, SPA, Vercel, Aseprite

---

Abstract

Outer Rim Smugglers is a 2D action video game developed with Godot 4 and GDScript, set in the Star Wars universe on the planet Tatooine. The player takes on the role of Han Solo and must fight through a desert town against enemies such as Stormtroopers, Sandtroopers, Rodians, and the boss Boba Fett. The combat system is real-time, with projectiles that travel in straight lines along the X or Y axis. The game features an inventory system, treasure chests scattered across the map containing collectible items, and a merchant NPC where items can be sold. All in-game sprites, except the tilemap assets (obtained for free online), were hand-drawn using Aseprite. The project also includes a single-page application (SPA) built with plain HTML and CSS and hosted on Vercel, which serves as the official download portal for the game. The total development time was approximately 150 hours, covering graphic design, game development, and web creation.

Keywords

Godot 4, GDScript, 2D game, Star Wars, Tatooine, tilemap, inventory system, SPA, Vercel, Aseprite

Índice

1. Presentación del proyecto

1.1 Introducción

Outer Rim Smugglers es un videojuego 2D de acción desarrollado con el motor Godot 4, utilizando GDScript como lenguaje de programación. El juego está ambientado en el universo de Star Wars, concretamente en las calles polvorientas de Mos Eisley en Tatooine, y sigue la historia de Han Solo atrapado entre esbirros de Jabba el Hutt, soldados imperiales y peligrosos cazarrecompensas. El jugador explora un mapa único construido con tilemap, como en los juegos de Pokémon de hace unos años, combate en tiempo real mediante disparos direccionales, recoge objetos de cofres repartidos por el escenario y los vende a un mercader de la ciudad. No hay un objetivo final específico, el jugador es libre para hacer lo que le apetezca ya sea combatir con enemigos, explorar Mos Eisley, saquear cofres para hacerte rico o acabar con el famoso cazarrecompensas Boba Fett. Junto al videojuego, el proyecto incluye una página web de una sola página (SPA) desarrollada con HTML y CSS puro, alojada en Vercel, que sirve como portal oficial donde cualquier usuario puede descargar el juego en formato .zip.

1.2 Contexto

Este proyecto es el Proyecto de Fin de Curso del CFGS de Desarrollo de Aplicaciones Multiplataforma (DAM) del Instituto Puig Castellar, curso 2025-2026.

La idea surgió de mis dos grandes pasiones: los videojuegos y Star Wars. Siempre he querido desarrollar mi propio juego, y este proyecto fue la oportunidad perfecta para hacerlo. Elegí Star Wars como temática porque es un universo que conozco bien y que me gusta mucho, y Tatooine como escenario porque encaja perfectamente con la estética de un juego de acción: un planeta marginal, lleno de personajes peligrosos y con una atmósfera única. Escogí Godot 4 como motor porque es gratuito, de código abierto y tiene muy buen soporte para juegos 2D. Además, GDScript es un lenguaje cómodo y rápido de aprender, lo que me permitió centrarme en desarrollar el juego sin perder demasiado tiempo con la curva de aprendizaje del motor.

1.3 Justificación

El desarrollo de videojuegos es una de las aplicaciones prácticas más completas de las habilidades de un programador multiplataforma: requiere diseñar arquitecturas de código limpias, gestionar estados de la aplicación, implementar lógica de juego compleja, diseñar interfaces y trabajar con recursos multimedia. Por ello, este proyecto permite demostrar de forma integral los conocimientos adquiridos durante el CFGS DAM.Además, el proyecto no se limita al videojuego: la creación de una web de descarga añade una dimensión de despliegue y distribución de software que es propia del perfil profesional del desarrollador de aplicaciones. La web complementa el juego y cierra el ciclo de producto: desde el desarrollo hasta la publicación y acceso por parte del usuario final.Por último, el hecho de diseñar todos los sprites del juego con Aseprite añade valor al proyecto como trabajo autónomo e integral.

1.4 Objetivos

El objetivo general del proyecto es desarrollar un videojuego 2D funcional y jugable, con todos sus sistemas integrados, y publicarlo a través de una página web propia.

Los objetivos específicos son los siguientes:

- Desarrollar un videojuego 2D con Godot 4 y GDScript, con mecánicas de combate, exploración e inventario.

- Implementar un sistema de combate en tiempo real con disparos direccionales para el jugador y los enemigos.

- Crear enemigos con inteligencia artificial básica: Stormtroopers, Sandtroopers y Rodianos como enemigos comunes, y Boba Fett como jefe.

- Implementar un sistema de cofres interactivos y un inventario que permite llevar hasta 5 tipos de objetos diferentes, con posibilidad de apilar unidades del mismo tipo.

- Implementar un NPC mercader con el que el jugador pueda vender los objetos recogidos mediante una ventana de inventario interactiva.

- Diseñar todos los sprites del juego con Aseprite, a excepción de los tiles del mapa.

- Construir el mapa del juego con tilemaps en Godot 4, representando Mos Eisley de Tatooine.

- Desarrollar una SPA con HTML y CSS puro que sirva de portal de descarga del juego.

- Desplegar la web en Vercel, con el archivo .zip del juego accesible para su descarga.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

El proyecto es técnicamente viable dado que todas las herramientas utilizadas son de acceso gratuito: Godot 4 es un motor de código abierto, Aseprite fue pirateado, y Vercel ofrece alojamiento gratuito para proyectos estáticos. Desde el punto de vista temporal, el proyecto se ha desarrollado a lo largo de varias semanas con una dedicación total aproximada de 150 horas, lo que lo sitúa dentro de los márgenes habituales de un proyecto de fin de ciclo. La estrategia de desarrollo fue lineal y secuencial: primero se diseñaron todos los recursos gráficos (sprites de personajes, enemigos, objetos y elementos de la UI) con Aseprite, a continuación se desarrolló el videojuego en su totalidad con Godot 4, y por último se creó la web de descarga. Esta estrategia permitió tener todos los assets listos antes de programar el juego, evitando tener que interrumpir el desarrollo para diseñar gráficos.Los recursos necesarios (ordenador personal, software gratuito o con licencia propia, conexión a internet) estuvieron disponibles desde el inicio, lo que garantizó la viabilidad del proyecto en todo momento.

2.2 Metodología de trabajo

La metodología de trabajo utilizada ha sido un enfoque secuencial por fases, adaptado a las características de un proyecto individual. No se ha aplicado una metodología ágil formal como Scrum, pero sí se ha trabajado de forma iterativa dentro de cada fase: una vez completada una funcionalidad, se probaba manualmente antes de avanzar a la siguiente.

El proceso de desarrollo se organizó en las siguientes fases:

1. Fase de diseño gráfico: creación de todos los sprites del juego con Aseprite (personaje jugador, enemigos, objetos de inventario, elementos de interfaz).

2. Fase de desarrollo del videojuego: implementación de todas las mecánicas en Godot 4 con GDScript (movimiento, combate, IA de enemigos, sistema de inventario, cofres, mercader, mapa).

3. Fase de desarrollo web: creación de la SPA con HTML y CSS puro, con ayuda de Claude AI para el diseño y maquetación.

4. Fase de despliegue: publicación de la web en Vercel con el archivo .zip del juego.Las pruebas se realizaron de forma manual durante todo el desarrollo, jugando al juego para detectar errores de comportamiento, colisiones incorrectas o problemas de lógica.

2.3 Planificación

El proyecto se desarrolló a lo largo de varias semanas con una dedicación total estimada de 150 horas. A continuación se detalla la distribución aproximada del tiempo por fases:

Fase 1 — Diseño gráfico con Aseprite (aprox. 5 h): Diseño de los sprites del jugador (Han Solo) y sus animaciones, diseño de los sprites de los enemigos (Stormtroopers, Sandtroopers, Rodianos, Boba Fett, Jabba el Hutt), diseño de elementos varios como cofres, el icono de los créditos…

Fase 2 — Desarrollo del videojuego en Godot 4 (aprox. 135 h): Configuración del proyecto y estructura de escenas, construcción del mapa con tilemap, implementación del movimiento del jugador, sistema de disparos direccionales, lógica de daño y muerte del jugador y los enemigos, implementación de la IA de los enemigos, sistema de cofres e inventario, NPC mercader con ventana de venta, pruebas manuales continuas.

Fase 3 — Desarrollo de la web y despliegue (aprox. 10 h): Maquetación de la SPA con HTML y CSS, con apoyo de Claude AI, preparación del archivo .zip del juego, despliegue en Vercel y comprobación del enlace de descarga.

3. Análisis

3.1 Casos de uso

A continuación se describen los principales casos de uso del sistema, diferenciando entre el videojuego y la web de descarga.

Videojuego

CU-01 Mover al jugador: El jugador utiliza las teclas de dirección o WASD para mover a Han Solo por el mapa del pueblo de Tatooine.

CU-02 Disparar: El jugador pulsa una tecla de disparo para lanzar un proyectil en línea recta a lo largo del eje X o Y. El proyectil impacta al primer enemigo que encuentre y le inflige daño.

CU-03 Recibir daño: Los enemigos pueden disparar al jugador. Si un proyectil enemigo impacta a Han Solo, este pierde puntos de vida.

CU-04 Abrir un cofre: Al acercarse a un cofre del mapa e interactuar con él, el jugador obtiene uno o más objetos que se añaden automáticamente a su inventario.

CU-05 Consultar el inventario: El jugador puede abrir la ventana de inventario en cualquier momento para ver los objetos que lleva. El inventario admite hasta 5 tipos de objetos diferentes, con apilamiento ilimitado por tipo.

CU-06 Vender objetos al mercader: Al interactuar con el NPC mercader del pueblo, se abre la ventana de inventario con un puntero de selección. El jugador elige qué objetos vender y recibe a cambio monedas.

CU-07 Combatir contra enemigos comunes: Los Stormtroopers, Sandtroopers y Rodianos patrullan el mapa y atacan al jugador cuando lo detectan. El jugador debe eliminarlos disparando.

CU-08 Combatir contra jefes: Boba Fett es un enemigo con muchos más puntos de vida que los demás enemigos comunes

Web de descarga

CU-09 Visitar la web: El usuario accede a la SPA desde cualquier navegador y encuentra información sobre el juego.

CU-10 Descargar el juego: El usuario hace clic en el botón de descarga y obtiene el archivo .zip del juego desde Vercel.

3.2 Requisitos funcionales

RF-01: El jugador debe poder moverse en las cuatro direcciones por el mapa mediante teclado.

RF-02: El jugador debe poder disparar proyectiles en línea recta a lo largo del eje X o del eje Y.

RF-03: Los proyectiles del jugador deben infligir daño al primer enemigo con el que colisionen y desaparecer tras el impacto.

RF-04: Los enemigos deben poder disparar proyectiles al jugador que le inflijan daño al impactar.

RF-05: Cada enemigo debe tener puntos de vida y morir cuando llegan a cero.

RF-06: El jugador debe tener una barra de vida visible. Si llega a cero, la partida termina.

RF-07: El mapa debe contener cofres interactivos que al abrirse añadan objetos al inventario del jugador.

RF-08: El inventario del jugador debe soportar hasta 5 tipos de objetos distintos. Los objetos del mismo tipo se apilan sin límite.

RF-09: Debe existir un NPC mercader con el que el jugador pueda interactuar para vender objetos del inventario.

RF-10: Al interactuar con el mercader, debe abrirse la ventana de inventario con un puntero de selección para elegir qué vender.

RF-11: El archivo .zip del juego debe estar alojado en Vercel y ser descargable desde la web.

3.3 Requisitos no funcionales

RNF-01 Rendimiento: El juego debe funcionar de forma fluida a 60 fotogramas por segundo en un ordenador de gama media con sistema operativo Windows.

RNF-02 Usabilidad: Los controles del juego deben ser intuitivos y no requerir tutorial extenso. El sistema de inventario y la interacción con el mercader deben poder utilizarse sin instrucciones previas.

RNF-03 Portabilidad: El juego se distribuye en formato ejecutable para Windows (.exe) dentro de un archivo .zip, sin necesidad de instalación adicional.

RNF-04 Accesibilidad web: La web de descarga debe ser accesible desde cualquier navegador moderno (Chrome, Firefox, Edge) sin necesidad de instalar extensiones ni plugins.

RNF-05 Disponibilidad: La web estará alojada en Vercel con alta disponibilidad y accesible en todo momento mientras el plan gratuito esté activo.

RNF-06 Mantenibilidad: El código del juego está organizado en escenas y scripts independientes en Godot 4, lo que facilita la modificación de funcionalidades de forma aislada.

RNF-07 Coherencia visual: Todos los sprites diseñados con Aseprite siguen un estilo pixel art uniforme y coherente con la estética retro del juego.

3.4 Análisis de alternativas tecnológicas

Para el desarrollo del videojuego se evaluaron los siguientes motores:

Unity: motor ampliamente utilizado en la industria, con gran cantidad de documentación y recursos. Sin embargo, requiere el uso de C# como lenguaje principal y ha experimentado cambios recientes en su modelo de licencias que generan incertidumbre. Además, para proyectos 2D sencillos puede resultar excesivamente pesado.

Godot 4: motor de código abierto, ligero y con soporte nativo para proyectos 2D. Permite trabajar con GDScript, un lenguaje de sintaxis similar a Python, fácil de aprender y muy cómodo para proyectos pequeños y medianos. Fue la opción elegida por su gratuidad, su filosofía open source y su excelente soporte para tilemaps y físicas 2D.

Pygame (Python): biblioteca de desarrollo de juegos sobre Python. Adecuada para proyectos muy simples, pero carece de editor visual, lo que obliga a gestionarlo todo por código y dificulta el trabajo con tilemaps y escenas complejas.

Para el diseño de sprites se evaluó Piskel (gratuito, online) frente a Aseprite (de pago, con licencia propia). Se eligió Aseprite por su mayor potencia, soporte para animaciones por fotogramas, paletas de color avanzadas y exportación optimizada para motores de juego.Para la web se descartó el uso de frameworks como React o Vue, ya que la página es una SPA estática y sencilla que no requiere gestión de estado ni componentes dinámicos. HTML y CSS puro fue la elección correcta por su simplicidad y compatibilidad universal.Para el alojamiento web se consideró GitHub Pages y Vercel. Se eligió Vercel por su mayor facilidad de despliegue, soporte automático de HTTPS y posibilidad de alojar archivos estáticos (como el .zip del juego) directamente en el repositorio.

4. Diseño

4.1 Arquitectura del sistema

El sistema está compuesto por dos partes independientes: el videojuego y la web de descarga.El videojuego está desarrollado íntegramente con Godot 4 y sigue la arquitectura de escenas propia del motor. Cada elemento del juego (jugador, enemigos, cofres, mercader, proyectiles, HUD) es una escena independiente con su propio GDScript. Las escenas se instancian dinámicamente en la escena principal del mundo (el mapa del pueblo de Tatooine). El mapa se construye con el sistema de TileMap de Godot 4, usando tiles estáticos para el suelo y las paredes, con capas de colisión definidas para que el jugador y los enemigos no puedan atravesar ciertos elementos del escenario.La comunicación entre elementos del juego se realiza mediante el sistema de señales (signals) de Godot, lo que desacopla los componentes y facilita la reactividad del sistema (por ejemplo, cuando un proyectil impacta a un enemigo, emite una señal que el enemigo recibe para restar vida).La web de descarga es una página estática de una sola página (SPA) compuesta por un único archivo HTML con su CSS asociado. Se aloja en Vercel junto con el archivo .zip del juego. No hay lógica de servidor ni base de datos, todo el contenido es estático.

4.2 Modelo de datos

El juego no utiliza ninguna base de datos ni sistema de persistencia externo. Todos los datos se gestionan en memoria durante la ejecución de la partida.

Los principales modelos de datos del juego son los siguientes:

Jugador (Han Solo): atributos de vida actual, vida máxima, velocidad de movimiento, daño por disparo y referencia al inventario.

Inventario: estructura de datos que almacena hasta 5 tipos de objetos distintos. Cada entrada contiene el tipo de objeto y la cantidad (sin límite de apilamiento). Se implementa como un diccionario en GDScript.

Objeto: cada objeto tiene un nombre, un identificador de tipo y un valor de venta (precio que el mercader paga por él).

Enemigo (común): atributos de vida actual, velocidad de movimiento, daño por disparo y comportamiento de IA (detección de coordenadas del jugador).

Jefe (Boba Fett): extiende el modelo de enemigo común pero con más vida.

Proyectil: dirección de desplazamiento (eje X o Y) y velocidad. Se destruye al impactar con un enemigo, el jugador o una pared.

Cofre: estado (abierto/cerrado) y lista de objetos que contiene.

La web no tiene modelo de datos propio: el único recurso es el archivo .zip del juego, servido estáticamente desde Vercel.

4.3 Diseño de interfaz

El juego usa una vista top-down con estética pixel art, inspirada en los RPG clásicos de la Game Boy Advance y consolas DS como Pokémon. Los tiles del mapa representan el suelo arenoso y los edificios de Mos Eisley. La interfaz de usuario del juego muestra la cantidad de créditos que tiene el jugador en la esquina superior derecha, un pequeño menú de pausa cuando se pulsa la tecla “esc”, las barras de vida de los personajes una vez han sido heridos y un texto en medio de la pantalla cuando el jugador muere. Cuando el jugador interactúa con el mercader o pulsa la tecla “B”, se abre una ventana de inventario en el centro de la pantalla. Esta ventana muestra los objetos disponibles y, en el caso de la interacción con el mercader, incluye un puntero de selección que el jugador puede mover para elegir qué objeto vender. Todos los sprites de personajes, enemigos y objetos fueron diseñados a mano con Aseprite, con un estilo pixel art coherente que mantiene la estética de la franquicia Star Wars adaptada al formato 2D retro. La web de descarga tiene un diseño temático con colores cálidos que representan el planeta de Tatooine de Star Wars. Consta de una única página con el título del juego, un poco de información como los géneros del videojuego, el espacio que ocupa en disco y un botón destacado de descarga. La maquetación se realizó con HTML y CSS puro, con ayuda de Claude AI para el diseño visual.

5. Desarrollo

5.1 Estructura del proyecto

El proyecto en Godot 4 está organizado siguiendo la estructura de directorios propia del motor:

Estructura del proyecto Godot 4

El proyecto está organizado siguiendo la estructura de directorios propia del motor:

`Outer-Rim-Smugglers/:` Esta es la carpeta raíz del proyecto que contiene los archivos de todas las escenas y scripts del videojuego. También dos carpetas con las listas de los objetos que contienen los dos tipos de cofre en formato .tres(text resource). Por último contiene dos carpetas con los sprites y efectos de sonido utilizados.

`Outer-Rim-Smugglers/gearChestItems:` Archivos .tres del cofre de equipo.

`Outer-Rim-Smugglers/junkChestItems:` Archivos .tres del cofre de chatarra.

`Outer-Rim-Smugglers/soundeffects:` Efectos de sonido.

`Outer-Rim-Smugglers/sprites:` Sprites del videojuego.

5.2 Implementación de funcionalidades

A continuación se describen las principales funcionalidades implementadas en el videojuego.

Movimiento del jugador: implementado en el script del jugador mediante la función _physics_process() de Godot. El movimiento es tile a tile: se lee la dirección de entrada con Input.is_action_pressed() y se interpola la posición del CharacterBody2D entre la casilla origen y la casilla destino usando un porcentaje de avance (percentMoved) escalado por la velocidad y el delta. Antes de iniciar el desplazamiento, un RayCast2D comprueba si la casilla destino está libre de colisiones. El jugador puede encontrarse en tres estados (IDLE, TURNING, WALKING): si cambia de dirección, primero ejecuta una animación de giro mediante el AnimationTree antes de empezar a caminar.

Sistema de disparos: el jugador puede disparar en la dirección hacia la que está mirando. Al pulsar la acción de disparo, se instancia la escena del proyectil (playerLaser.tscn) y se añade como hijo de la escena padre, posicionándolo con un offset respecto al jugador según la dirección (facing_direction). Un temporizador (Timer) impide el disparo continuo. Los proyectiles del enemigo funcionan de forma distinta: al instanciarse, calculan en _ready() la dirección normalizada hacia el jugador y se desplazan a velocidad fija. La detección de impactos se gestiona mediante Area2D: cuando el láser del jugador colisiona con el hitbox de un enemigo, este recibe daño y el proyectil se destruye con queue_free(), y viceversa para los láseres enemigos.

Sistema de looteo de cofres: existen dos tipos de cofres (GearChest y JunkChest), ambos implementados como StaticBody2D. Cada cofre tiene exportada una lista de ítems posibles (lootPool: Array[Item]), y en el momento de inicializarse selecciona aleatoriamente uno con lootPool.pick_random(). El cofre detecta la presencia del jugador mediante señales de Area2D (body_entered / body_exited). Al pulsar la acción de interactuar estando dentro del rango, se llama al método pickObject() del jugador, que delega en el inventario. Si el objeto se añade con éxito, el cofre reproduce su animación de apertura y sonido, y queda marcado como abierto (isOpen = true) para no poder interactuarse de nuevo. Si el inventario está lleno, se muestra un mensaje de aviso mediante un autoload (InventoryMessage).

Sistema de inventario: el inventario está implementado en inventory_ui.gd como un diccionario GDScript (inventory := {}), donde la clave es el recurso Item y el valor es la cantidad acumulada. La capacidad máxima es de 5 tipos distintos de objeto (invCapacity). El método addObject() comprueba si el ítem ya existe en el diccionario para sumar cantidad, o si hay hueco para añadirlo como nueva entrada; en caso contrario retorna false. El método takeoutObject() reduce la cantidad y elimina la entrada del diccionario si llega a cero. La UI se regenera completamente en cada cambio mediante el método update(), que borra y recrea las etiquetas del VBoxContainer.

Mercader y sistema de venta: al acercarse e interactuar con el NPC mercader, su script (Trader.gd) desactiva el movimiento del jugador con set_physics_process(false) y llama a enableTradingMode() en la UI de inventario. En este modo, un cursor (selection) navega por los ítems del inventario con las teclas de dirección. Al confirmar la venta con la tecla de aceptar, se llama a vender_item(), que extrae el objeto del inventario mediante takeoutObject() y llama a add_money() del nodo MoneyHUD, el cual actualiza el contador de créditos visible en pantalla. Al cerrar la ventana, se reactiva el movimiento del jugador y se desactiva el modo trading.

Web de descarga: página estática con un enlace de descarga que apunta al archivo .zip alojado en el mismo repositorio de Vercel. El enlace usa el atributo download de HTML para forzar la descarga del archivo.

5.3 Pruebas

Las pruebas realizadas en el proyecto han sido exclusivamente manuales. No se han implementado pruebas automatizadas, pruebas unitarias ni pruebas de integración formales.El proceso de pruebas consistió en jugar al videojuego de forma continua durante el desarrollo, comprobando el correcto funcionamiento de cada funcionalidad en cuanto era implementada.

6. Conclusiones

6.1 Conclusiones generales

El resultado del proyecto es un videojuego 2D funcional y jugable ambientado en el universo de Star Wars, con un sistema de combate, exploración, inventario y venta de objetos implementados correctamente. Además, el juego cuenta con un portal web de descarga pública alojado en Vercel, lo que lo convierte en un producto completo y distribuible, a pesar de la gran cantidad de desarrollo que le faltaria para poder competir mínimamente con los juegos de hoy en dia.El proyecto ha permitido aplicar de forma práctica los conocimientos adquiridos durante el ciclo: programación orientada a objetos, diseño de arquitecturas de software, gestión de recursos, desarrollo de interfaces y despliegue de aplicaciones.Además, el desarrollo del proyecto ha supuesto el aprendizaje autónomo de nuevas herramientas no vistas directamente en el ciclo, como el motor Godot 4, el lenguaje GDScript y la herramienta de diseño gráfico Aseprite. Este aprendizaje autónomo ha sido uno de los aspectos más valiosos del proyecto, demostrando mi capacidad de adquirir competencias nuevas de forma independiente.

6.2 Consecución de objetivos

A continuación se analiza el grado de cumplimiento de cada uno de los objetivos específicos definidos en el apartado 1.4.

El desarrollo de un videojuego 2D con Godot 4 y GDScript quedó cumplido, ya que el juego es funcional y jugable. El sistema de combate en tiempo real con disparos direccionales también se completó: los proyectiles se desplazan en línea recta por el eje X o Y y dañan tanto a enemigos como al jugador. Los enemigos con IA básica fueron implementados, con Stormtroopers, Sandtroopers y Rodianos. El jefes Boba Fett se incorporó con más vida respecto a los enemigos comunes. El sistema de cofres e inventario funciona correctamente, entregando objetos al jugador con apilamiento y límite de 5 tipos. El NPC mercader dispone de ventana de venta con puntero de selección. Todos los sprites de personajes, enemigos y objetos fueron diseñados a mano con Aseprite, y el mapa de “Mos Eisley” en Tatooine se

construyó con el sistema TileMap de Godot 4. Finalmente, la SPA está operativa y accesible, y el juego es descargable desde la web desplegada en Vercel.

6.3 Valoración de la metodología y planificación

La metodología secuencial por fases utilizada ha funcionado correctamente para este tipo de proyecto individual. Comenzar con el diseño gráfico antes de programar fue una decisión acertada, ya que disponer de todos los assets desde el inicio evitó interrupciones durante el desarrollo del código.La estimación de 150 horas totales fue razonablemente ajustada, aunque la fase de desarrollo del videojuego resultó ser la más extensa y la que presentó más imprevistos.La decisión de apoyarse en Claude AI para la creación de la web fue adecuada y permitió resolver rápidamente la parte de diseño web, dedicándole el tiempo estrictamente necesario. Todo el contenido generado fue revisado, comprendido y adaptado antes de su uso.

6.4 Visión de futuro

El proyecto tiene un buen margen de mejora y expansión. Algunas de las posibles líneas de desarrollo futuro pasan por añadir utilidad a los objetos coleccionables de los cofres y al dinero, añadir más mecánicas de combate como habilidades o la capacidad de curarse, añadir un poco de historia... También sería posible ampliar el mapa con nuevas zonas del pueblo o escenarios adicionales, como el interior del Palacio de Jabba o el desierto abierto. Otra mejora relevante sería implementar un sistema de guardado y carga de partida. En cuanto a la experiencia de usuario, se podría añadir un menú principal con opciones ajustables. Por último, la web podría enriquecerse con capturas de pantalla, un tráiler en vídeo y un contador de descargas.

7. Glosario

Aseprite: software de edición gráfica especializado en pixel art y animaciones por fotogramas. Utilizado para diseñar todos los sprites del juego.

GDScript: lenguaje de programación propio de Godot Engine, con sintaxis similar a Python. Utilizado para programar toda la lógica del juego.

Godot 4: motor de desarrollo de videojuegos de código abierto. Versión 4 del motor Godot, con soporte mejorado para 2D y 3D.

NPC: personaje del juego no controlado por el jugador.

Pixel art: estilo gráfico digital que trabaja a nivel de píxel individual, produciendo imágenes de baja resolución con estética retro.

SPA: aplicación web de una sola página que no requiere recarga del navegador para mostrar su contenido. En este proyecto, la web de descarga es una SPA estática.

TileMap: sistema de Godot para construir mapas a partir de mosaicos (tiles) repetibles. Permite definir capas de colisión y organización visual del escenario.

Vercel: plataforma de despliegue de aplicaciones web estáticas y serverless. Utilizada para alojar la web de descarga del juego.

8. Bibliografía

Godot Engine: https://godotengine.org/es/

Aseprite: https://www.aseprite.org/

Vercel: https://vercel.com/docs

Tiles del mapa: https://beyonderboy.itch.io/desert-map-tileset-16x16

9. Anexos

Capturas de pantalla del juego:

Enlace al portal de descarga: https://outer-rim-smugglers.vercel.app/

Nota sobre el uso de inteligencia artificial

Durante la realización de este proyecto se ha utilizado la herramienta de inteligencia artificial Claude AI: para el diseño y maquetación de la página web de descarga, para la redacción y revisión del presente documento de memoria, y para el facilitar el desarrollo del código del videojuego. En todos los casos, el contenido generado ha sido leído, comprendido, revisado y validado por el autor antes de su uso final. La IA ha actuado como herramienta de apoyo, pero las decisiones de diseño, la lógica del juego y el contenido de la memoria reflejan mi trabajo y mi criterio.