Institut Puig Castellar
Santa Coloma de Gramenet

COLLECT THE COINS

Proyecto de desarrollo

CFGM Administración de Sistemas Microinformáticos i Redes

Miembros: Shaim Howlader, Jansen Ureña Oporto

Grupo: A

Curso académico: 2SMX

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Resumen del proyecto:

Nuestro proyecto consiste en desarrollar un videojuego propio utilizando el motor gráfico Godot, concretamente la versión 4.6, y el lenguaje de programación GDScript.

El objetivo principal es diseñar y desarrollar un videojuego funcional que integre mecánicas básicas. Cabe mencionar que el proyecto se inició en Godot 4.5 y posteriormente se migró a la versión 4.6. Además, se buscó dominar las herramientas principales del motor y crear un producto final jugable que refleje el proceso creativo y técnico del equipo.

Para alcanzar los objetivos planteados, se siguió una metodología basada en el aprendizaje autónomo, el uso de tutoriales y la práctica progresiva dentro del motor. El proceso se estructuró en tres fases: investigación mediante vídeos y guías, modificación del código para adaptarlo a las necesidades del juego, y consulta de recursos sobre creación de escenas, personajes, niveles y sistemas básicos.

Este trabajo ha demostrado la importancia de avanzar de forma progresiva, experimentando y adaptando los ejemplos aprendidos a las necesidades del proyecto. También ha quedado clara la relevancia de la resolución de problemas y la organización como elementos clave para mejorar el resultado final.

Palabras claves

#Videojuego #Godot #Mundos #Monedas #Enemigos #Carrera

Abstract:

Our project consists of developing an original video game using the Godot engine, specifically version 4.6, and the GDScript programming language.

The primary objective is to design and develop a functional video game that integrates core mechanics. It is worth noting that the project began in Godot 4.5 and was subsequently migrated to version 4.6. Additionally, we aimed to master the engine's main tools and create a playable final product that reflects the team's creative and technical process.

To achieve these goals, we followed a methodology based on autonomous learning, the use of tutorials, and progressive practice within the engine. The process was structured into three phases: research through videos and guides, code modification to adapt it to the game's requirements, and consulting resources on the creation of scenes, characters, levels, and basic systems.

This work has demonstrated the importance of advancing progressively, experimenting, and adapting learned examples to the specific needs of the project. It also highlighted the relevance of problem-solving and organization as key elements for improving the final result.

Keywords (entre 4 i 8):

#Videogame #Godot #Worlds #Coins #Enemies #Speedrun

Índice

Lista de figuras

1 Introducción

El juego se centra en recoger monedas mientras el jugador intenta sobrevivir a diferentes trampas y obstáculos que aparecen a lo largo de los niveles. Nuestro objetivo es crear un juego entretenido, con dificultad progresiva y fácil de entender, pero difícil de completar.

Hemos querido diseñar un juego accesible para cualquier tipo de jugador, sin necesidad de experiencia previa, pero que aun así requiera esfuerzo y precisión para superar todos los niveles.

También trabajamos para que el juego tenga una estructura clara, con niveles bien diferenciados y una dificultad que aumente de forma gradual. De esta manera, el jugador puede mejorar mientras avanza y evitar que la experiencia se vuelva repetitiva.

Otro objetivo importante del proyecto ha sido aprender a trabajar en equipo y organizar mejor el desarrollo. Tuvimos que repartir tareas, resolver errores y aprender a utilizar herramientas nuevas, lo que nos permitió comprender mejor cómo se crea un videojuego desde cero.

En general, este proyecto no solo consiste en hacer un juego, sino también en aprender sobre programación, diseño y trabajo en grupo, con la intención de crear un resultado final funcional y divertido para otras personas.

Mecánica principal

El juego se fundamenta en un sistema de recolección de monedas en entornos de plataformas 2D. El jugador debe recoger la totalidad de las monedas de cada nivel dentro de un tiempo límite, evitando el contacto con enemigos y obstáculos que reducen su barra de vida.

La vida no se regenera entre niveles, lo que introduce una gestión de recursos continúa a lo largo de toda la partida. Si desciende al 0%, el juego reinicia desde el nivel 1.

La dificultad aumenta progresivamente en cada uno de los seis niveles, incrementando la densidad de obstáculos y la complejidad del escenario.

Personaje principal

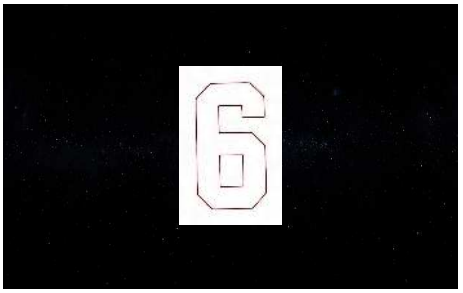
Personaje con la barra de vida

- Estilo visual

Al iniciar el juego, el jugador accede al menú principal, donde dispone de tres opciones: ver desafío, jugar y salir. El juego cuenta con seis mapas, cada uno representando un mundo distinto con su propio fondo y paleta de colores característica. El sexto mapa actúa como nivel final.

Cada mundo está diseñado para ofrecer una experiencia visual única, desde cielos nocturnos con luna llena hasta campos brillantes y coloridos. Algunos enemigos presentan colores similares al entorno para aumentar la dificultad visual y obligar al jugador a prestar más atención.

Imágenes de fondo:



- Organización de trabajo

El equipo trabajó de forma individual pero coordinada mediante **GitHub**, un sistema de control de versiones que permite hacer un seguimiento detallado de los cambios en el código y mantener copias de seguridad del proyecto.

Gracias a esta herramienta, cada miembro pudo trabajar de manera independiente sin riesgo de perder el trabajo realizado, facilitando la colaboración y la integración de las distintas partes del juego.

1.1 Contexto

- Jugabilidad en los juegos 2D

La jugabilidad en los videojuegos de plataformas 2D se basa en controles simples pero precisos, donde cada movimiento del personaje influye directamente en el ritmo del nivel. Saltar, esquivar o recoger objetos son acciones básicas que deben funcionar correctamente para que la experiencia de juego resulte satisfactoria.

En este tipo de juegos es fundamental que los controles respondan con fluidez, ya que un movimiento impreciso puede hacer que el jugador pierda el control y, con ello, el interés por el juego. Por este motivo, se ha trabajado para que el movimiento del personaje sea fluido y que las acciones se comprendan fácilmente desde el primer momento.

Los juegos 2D suelen centrarse en la habilidad del jugador: no se trata solo de avanzar, sino de saber cuándo saltar, cuándo detenerse y cómo esquivar los obstáculos sin perder vida.

- Influencia de los juegos en ya sea 2D o 3D

Los juegos de plataformas en 2D han sido una parte esencial de la historia de los videojuegos. Desde las primeras consolas hasta la actualidad, este estilo ha marcado a generaciones enteras gracias a su forma directa y visual de presentar la acción. Títulos como Super Mario Bros., Celeste o Hollow Knight han demostrado que no es necesaria una cámara compleja para crear mundos memorables y llenos de personalidad. Su fortaleza reside en la claridad del diseño, la fluidez del movimiento y la creatividad en el diseño de niveles, cualidades que los mantienen vigentes incluso hoy en día.

Imagen de Hollow Knight

Imagen de Mario Bros

1.2 Justificación

El equipo llevaba tiempo con la idea de crear un videojuego, pero siempre se había pospuesto por falta de tiempo o experiencia. Finalmente, se decidió comenzar el proyecto, siendo conscientes de que no sería un proceso sencillo y que habría que aprender, cometer errores y rehacer partes del desarrollo.

El objetivo principal no era obtener un producto perfecto, sino adquirir experiencia real, mejorar como desarrolladores y ver cómo una idea se convierte progresivamente en algo tangible. Este proyecto representa la forma del equipo de crear algo propio y descubrir hasta dónde se puede llegar con esfuerzo y constancia.

1.3 Objetivos

1.3.1 Objetivos generales

El objetivo del proyecto es crear un videojuego 2D utilizando **Godot Engine** y **GDScript**. El juego propone al jugador avanzar por niveles llenos de trampas, enemigos y obstáculos que exigen precisión y rapidez.

Durante el desarrollo también se trabajó en el diseño de cada nivel, en el comportamiento de los enemigos y en que las trampas tuvieran coherencia dentro del juego. Asimismo, se buscó mantener un estilo visual sencillo pero agradable, logrando una estética uniforme durante toda la partida.

En resumen, el objetivo fue construir un juego 2D funcional, entretenido y con una estética definida, mientras se adquirían y mejoraban conocimientos en el desarrollo con Godot.

1.3.2 Objetivos específicos

1. Diseñar una idea original para el videojuego, incluyendo personajes, trampas y enemigos.
2. Programar todas las mecánicas del juego.
3. Desarrollar una interfaz de usuario clara y atractiva.
4. Crear plataformas y niveles con diseño propio.
5. Añadir efectos visuales básicos para enriquecer la experiencia, como partículas y animaciones.
6. Implementar efectos de sonido para acciones como saltos, golpes y trampas, así como música de fondo.

1.4 Estrategia y planificación del proyecto

Para la planificación del proyecto se utilizó un **diagrama de Gantt** como herramienta de organización. Este permitió obtener una visión general de las tareas previstas y establecer una distribución inicial del trabajo.

Sin embargo, a medida que avanzó el desarrollo, el diagrama fue perdiendo protagonismo, ya que surgieron tareas imprevistas según las necesidades del juego. Esto obligó a adaptar la planificación sobre la marcha, ajustando tiempos y prioridades para mantener el progreso del proyecto.

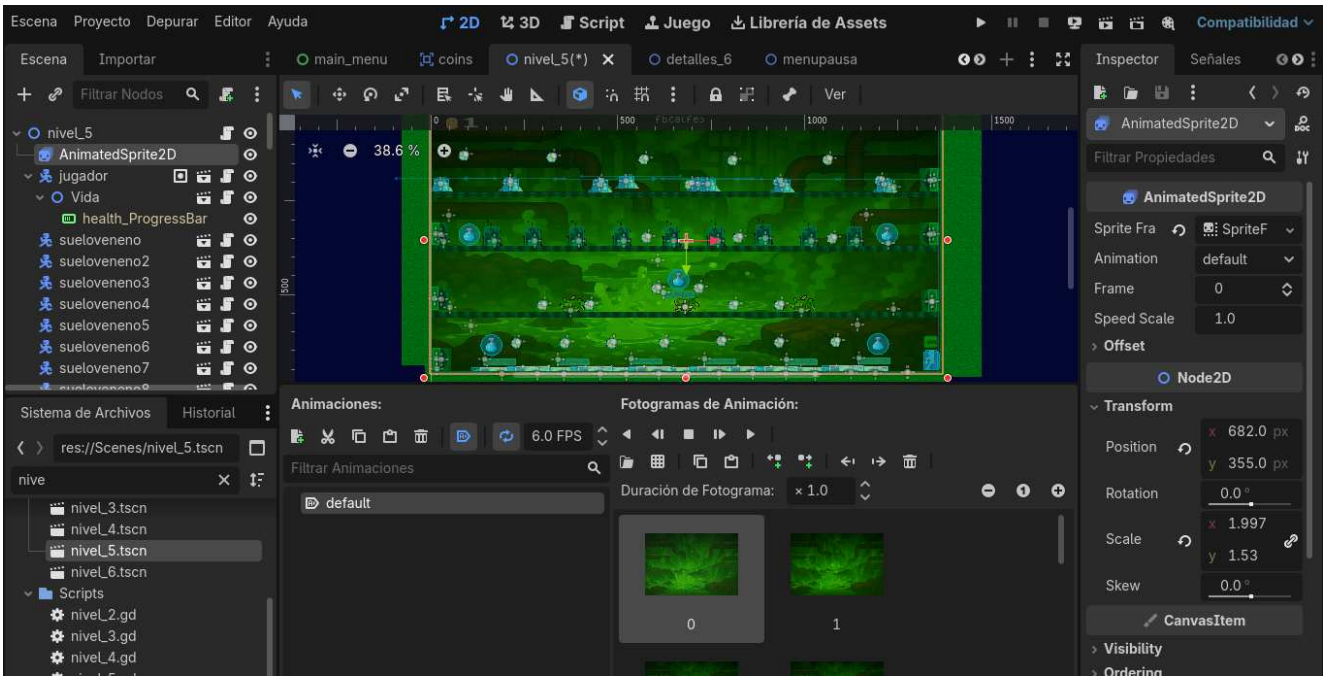
El uso del Gantt fue útil como punto de partida, pero el desarrollo real requirió flexibilidad y capacidad de reacción ante problemas técnicos, cambios en el diseño y nuevas ideas que aparecieron durante el proceso.

1.5 Metodología de trabajo

Para llevar a cabo este proyecto se siguió una metodología basada en el aprendizaje autónomo. Al no contar con experiencia previa en Godot Engine, el proceso comenzó investigando tutoriales en vídeo de YouTube y consultando la documentación oficial del motor, adaptando los ejemplos encontrados a las necesidades concretas del juego.

El trabajo se organizó de forma individual pero coordinada: cada miembro trabajaba en su parte y sincronizaba los cambios mediante GitHub, lo que permitía revisar el progreso y evitar la pérdida de código. Cuando surgía algún problema, se buscaban soluciones consultando herramientas de inteligencia artificial como ChatGPT o Copilot.

El desarrollo se estructuró en fases: primero el movimiento del jugador, después los enemigos y obstáculos, a continuación la interfaz y, finalmente, el ajuste de los niveles. Cada fase incluía pruebas constantes para detectar errores antes de avanzar a la siguiente.



1.6 Estudio económico y presupuestario

En nuestro proyecto no vamos a gastar dinero real porque utilizaremos programas gratuitos, pero igualmente hemos hecho un cálculo aproximado como si fuera un proyecto real.

Tareas principales del proyecto

1. Ordenador de cualquier sistema operativo
2. El ordenador tiene que tener el motor Godot Engine 4 para ejecutar el juego
3. Mantener una tasa estable de al menos 30 FPS
4. Ratón (Opcional)
5. Conexión a internet para usar GitHub y descargar recursos.
6. Programas gratuitos para crear o editar sprites e imágenes.
7. Espacio suficiente en el disco duro para guardar el proyecto.
8. Cuenta de GitHub para guardar las versiones del juego.
9. Tiempo de trabajo durante el curso para desarrollar el videojuego.

Recursos	Descripción	Coste aproximado
Ordenador	Ordenador para programar y ejecutar Godot	**400 €**
GitHub Pages	Hosting de la página web del juego	**0 €**
Software Godot	Motor de videojuegos	**0 €**
GitHub	Control de versiones	**0 €**
Recursos gráficos	Sprites e imágenes gratuitas	**0 €**
Recursos de sonido	Efectos de sonido gratuitos	**0 €**

Coste total: 400 €

2 Descripción del proyecto

2.1 Análisis de requisitos

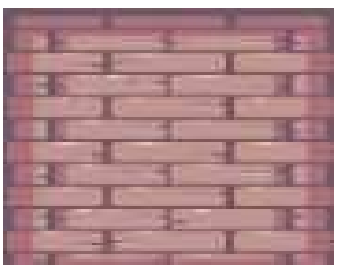
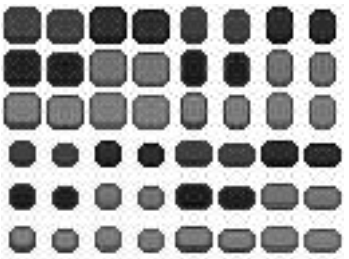
- El jugador debería moverse hacia la derecha y izquierda con las flechas


- Necesita saltar con el espacio y correr con la tecla shift


- Hay una barra de vida encima del jugador, contador de monedas y la cuenta regresiva

- Estos son los tilemaps que hemos utilizado para crear el mapa



- No hay errores graves que rompan la estructura del juego

2.1.1 Requisitos funcionales

El jugador puede moverse libremente, saltar y correr. También puede recoger objetos como pociones de cura y está expuesto a una poción de veneno invisible que reduce su vida progresivamente. Los enemigos no se dirigen directamente hacia el jugador, sino que patrullan de un lado a otro, por lo que el jugador debe anticiparse a sus movimientos para evitarlos. Las trampas se encuentran en el suelo y existen varios tipos. Además, hay plataformas móviles que el jugador debe utilizar para progresar por los niveles.

En todo momento se muestra en pantalla la vida del jugador, el número de monedas recogidas y el contador de monedas totales. El juego dispone de varios menús: menú principal, menú de pausa, pantalla de misión completada y pantalla de misión fallida, todos ellos intuitivos y fáciles de usar.

Menús:

Menú principal:

Menú de pausa:

Pantalla feedback:

Pantalla de victoria:

Pantalla de derrota:

2.1.2 Requisitos no funcionales

Rendimiento: El juego debe ejecutarse de forma fluida, manteniendo un mínimo de 30 FPS en condiciones normales.

Usabilidad: La interfaz y los controles deben ser intuitivos, permitiendo que cualquier usuario pueda jugar sin curva de aprendizaje significativa.

Estabilidad: El sistema debe operar sin errores críticos, bloqueos ni cierres inesperados.

Compatibilidad: El juego debe funcionar correctamente en equipos con sistema operativo Windows.

Tiempo de carga: Las transiciones entre niveles deben completarse en un tiempo reducido para mantener la atención del jugador.

Interfaz de usuario: La información relevante (vida, monedas, estado del jugador) debe mostrarse de forma clara y legible.

Tiempo de respuesta: Los controles deben reaccionar de forma inmediata a las acciones del usuario.

Claridad visual: Los elementos del juego (enemigos, monedas, obstáculos) deben ser fácilmente distinguibles para evitar confusiones pero depende del nivel.

2.2 Tecnologías

2.2.1 Comparativa de las tecnologías valoradas

Godot

Godot es un motor de desarrollo de videojuegos que permite crear juegos en 2D y 3D. En nuestro proyecto lo utilizamos para crear los niveles, programar el movimiento del personaje y añadir los enemigos y monedas.

Pros:

Es gratuito y de código abierto.

Es fácil de aprender para proyectos pequeños.

Está muy bien optimizado para juegos en 2D.

Tiene su propio lenguaje de programación (GDScript), que es sencillo.

Contras:

Cuenta con una comunidad y ecosistema más reducidos en comparación con otros motores como Unity o Unreal Engine.

Para proyectos muy grandes puede quedarse más limitado.

A veces pueden aparecer errores (bugs) que no son fáciles de solucionar.

GitHub

GitHub es una plataforma que permite guardar proyectos y controlar los cambios del código. Nosotros lo usamos para no perder el proyecto y poder trabajar en equipo.

Pros:

Permite guardar copias del proyecto en la nube, evitando pérdidas si falla el ordenador.

Facilita trabajar en equipo porque varias personas pueden subir cambios.

Permite volver a versiones anteriores si algo falla o si cometemos un error.

Guarda el historial de todo lo que se ha cambiado en el proyecto.

Contras:

Si no se organiza bien, pueden aparecer conflictos entre archivos.

Si se suben archivos mal, hay que saber cómo corregirlos.

Si no haces commits seguidos puedes perder cambios sin darte cuenta.

GDScript

GDScript es el lenguaje de programación que usa Godot y es el que utilizamos para programar el juego.

Pros:

Está muy bien integrado con Godot y funciona rápido dentro del motor.

Permite programar juegos sencillos sin mucho código.

Permite hacer pruebas rápidas durante el desarrollo.

Contras:

. Solo se usa dentro de Godot, por lo que no es tan útil fuera del programa.

. Tiene menos funciones avanzadas que otros lenguajes más complejos.

. Para proyectos muy grandes puede quedarse limitado.

Git Bash

Git Bash es una herramienta que permite usar comandos de Git a través de una consola. Nosotros la utilizamos para subir cambios al repositorio y gestionar el proyecto desde el ordenador.

Pros:

Permite usar comandos de Git de forma directa y rápida.

Funciona bien junto con GitHub para subir y descargar cambios.

Permite tener más control sobre el proyecto.

Es útil para aprender cómo funciona Git realmente.

No ocupa mucho espacio y es fácil de instalar.

Contras:

Al principio es difícil porque hay que aprender comandos para gestionar archivos del proyecto de Godot .

Si escribes mal un comando puedes hacer cambios sin querer.

No tiene interfaz gráfica, todo se hace por texto.

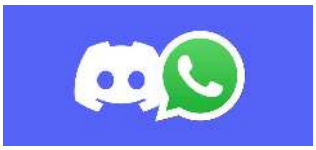
2.2.2 Tecnologías elegidas

Godot: Lo utilizamos como motor principal para desarrollar el videojuego. Con Godot creamos los niveles, añadimos los personajes, enemigos, monedas y toda la parte visual del juego. Lo hemos elegido porque es gratuito, no es muy difícil de usar y está muy bien para juegos en 2D como el nuestro.

GDScript: Es el lenguaje de programación que usa Godot. Lo utilizamos para hacer que el juego funcione, por ejemplo para el movimiento del personaje, las colisiones o el sistema de vida. Nos ha resultado bastante fácil porque es parecido a Python y se entiende bien.

Google Drive: Lo usamos para compartir archivos importantes como documentos del proyecto, imágenes o ideas. Así los dos podemos acceder a todo desde cualquier sitio y tener una copia guardada por si pasa algo.

WhatsApp o Discord: Los utilizamos para comunicarnos rápidamente. Nos sirve para hablar sobre el proyecto, avisarnos de cambios o resolver dudas sin tener que estar en el mismo sitio. Esto nos ayuda a trabajar más rápido y coordinados.



Formspree: Esta herramienta principalmente la hemos utilizado para nuestra página web, dentro de la página web hay un formulario para que nos puedas enviar un mensaje por si has encontrado algún bug, alguna sugerencia de mejora o simplemente dejar una reseña de tu experiencia con el juego.

Git bash: Lo usamos para gestionar y subir nuestro proyecto a GitHub escribiendo comandos directamente. Es una terminal de Windows que es igual que Linux.

Github para añadir la nueva versión del juego, subir los últimos cambios en el repositorio para no perderlo.

git add .: Prepara todos los archivos modificados o nuevos para ser guardados. El punto . significa "todo lo que hay en esta carpeta".

```
shaim@LAPTOP-1V7GLCM9 MINGW64 ~/Pictures/STEAL-A-COINS-1 (main)
$ git add .
```

git commit -m “Hecho”: Guarda los cambios preparados con un mensaje descriptivo.

```
shaim@LAPTOP-1V7GLCM9 MINGW64 ~/Pictures/STEAL-A-COINS-1 (main)
$ git commit -m "Hecho"
```

git push: Subir los cambios al github

```
shaim@LAPTOP-1V7GLCM9 MINGW64 ~/Pictures/STEAL-A-COINS-1 (main)
$ git push
```

GitHub Pages: Servicio gratuito de GitHub que permite publicar páginas web directamente desde un repositorio. Se usó para crear la página oficial del juego, donde cualquier persona puede descargarlo, ver los controles y contactar con los desarrolladores. Los pasos son simplemente en el apartado de pages, cambiar el none por main y lo guardamos. En 1 minuto o 2, github ya nos había generado el enlace.

2.3 Estructura del proyecto

El proyecto del videojuego Collect The Coins está hecho en diferentes partes dentro de Godot Engine. Cada parte del juego se encarga de una función concreta para garantizar el correcto funcionamiento del proyecto

Jugador:
Aquí se encuentra el personaje principal del juego. Esta parte incluye el sprite del personaje, las animaciones y el script que controla su movimiento: caminar, saltar y detectar colisiones con los objetos de cada nivel.

Monedas:
Las monedas son los objetos que el jugador debe recoger para completar cada nivel. Cada moneda tiene un pequeño script que detecta cuando el jugador la toca para sumarla al contador y hacer que desaparezca.

Enemigos y obstáculos:
En esta parte se encuentran los diferentes peligros del juego, como pinchos, sierras, fuego o enemigos que se mueven.

Estos objetos tienen scripts que detectan cuando el jugador los toca y hacen que pierda vida hasta llegar al 0%. Cuando esto ocurre, el jugador debe volver a empezar desde el nivel 1.

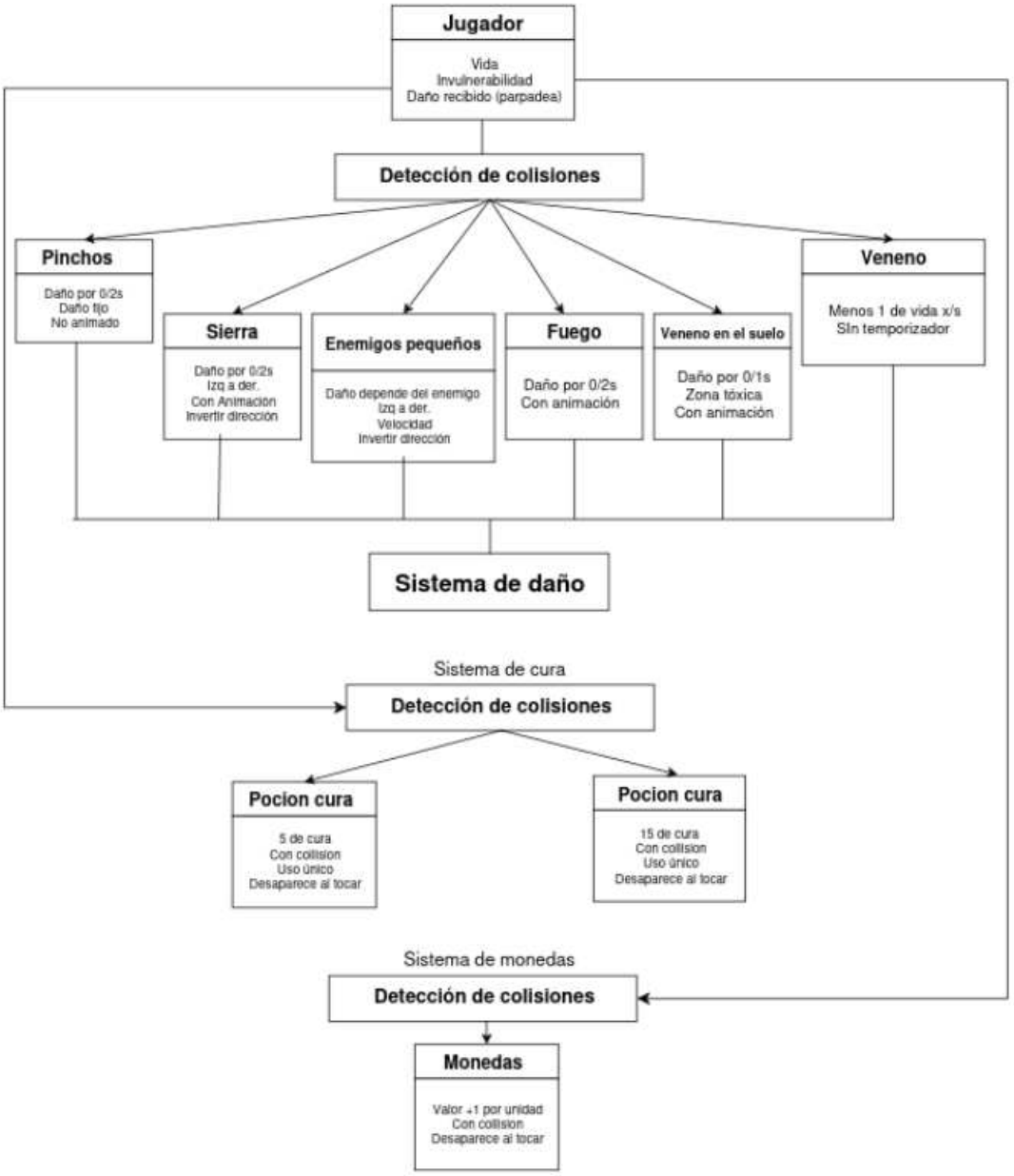
Niveles:
El juego tiene 6 niveles diferentes. Cada nivel tendrá plataformas, obstáculos, enemigos y monedas colocadas en distintas posiciones. A medida que el jugador avanza , los niveles se vuelven más difíciles.

Sistema de progreso:
Esta parte controla cuántas monedas ha recogido el jugador y si el nivel se ha completado. Si el jugador pierde, el juego vuelve al nivel 1. También controla la vida en el script GameData (autoload).

Menús del juego o feedback:
El proyecto también tiene pantallas como el menú principal, la pantalla de inicio del juego y la pantalla que aparece cuando el jugador pierde o gana el nivel y tiene un menú pausa que lo que hace es que puedes pausar el juego en cualquier momento y seguir cuando quieras, en cambio si te sales del juego, el progreso no se guardará.

2.4 Descripción de los componentes

Para hacer el videojuego Collect The Coins se utilizan diferentes componentes dentro de Godot Engine. Cada componente tiene una función concreta dentro del juego y todos trabajan juntos para que el juego funcione correctamente. Aquí se explican los componentes principales y cómo funciona cada uno.



2.4.1 Componente 1 (Jugador)

El componente del jugador es el personaje principal que controla la persona que está jugando. Este componente incluye el sprite del personaje, sus animaciones y el script que gestiona sus acciones.

El script del jugador se encarga de controlar el movimiento (caminar a izquierda o derecha, saltar), así como detectar colisiones con plataformas, enemigos o monedas.

También administra la vida del jugador y detecta cuándo recibe daño por parte de un enemigo u obstáculo.

Este componente es uno de los más importantes del juego, ya que permite la interacción entre el jugador y el resto de elementos del nivel.

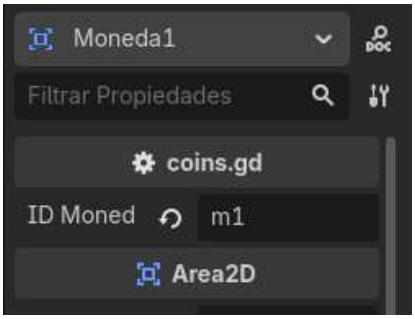
2.4.2 Componente 2 (Monedas)

Las monedas son los objetos que el jugador debe recoger durante cada nivel. Cada moneda tiene un pequeño script que detecta cuándo el jugador la toca.

Cuando el jugador entra en contacto con una moneda, esta desaparece y se suma al contador de monedas recogidas.

Este sistema permite comprobar si el jugador ha conseguido todas las monedas necesarias para completar el nivel.

A continuación se muestran los frames de animación de la moneda, que representan su movimiento giratorio para hacerla más visible dentro del nivel:



2.4.3 Componente 3 (enemigos y obstáculos)

Este componente incluye todos los elementos peligrosos del juego, como pinchos, sierras, fuego o enemigos que se mueven por el mapa.

Cada uno de estos objetos tiene un sistema de colisiones que detecta cuando el jugador los toca. Cuando esto ocurre, el jugador pierde vida.

Si la vida del jugador llega al 0%, el juego se reinicia desde el primer nivel.

Este componente sirve para aumentar la dificultad del juego y hacer que el jugador tenga que evitar los peligros mientras intenta recoger las monedas.

Ejemplos de enemigos y trampas:



2.5 Definición de las funcionalidades

El videojuego Collect the Coins tiene varias funcionalidades que hacen que el jugador pueda jugar y avanzar por los diferentes niveles. La idea del juego es bastante simple: controlar al personaje, recoger monedas y evitar obstáculos o enemigos. Esto se ha desarrollado usando Godot Engine. Os explicamos las funcionalidades principales del proyecto y qué hace cada una dentro del juego.

2.5.1 Funcionalidad del movimiento del jugador

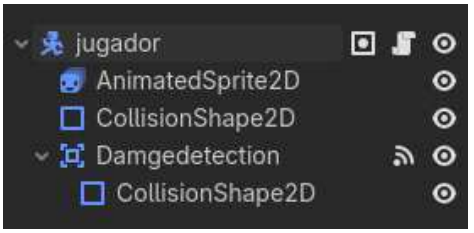
Una de las funcionalidades muy importantes es el movimiento del jugador. El jugador puede controlar al personaje para moverse por el mapa, ir hacia la izquierda o hacia la derecha, también saltar para pasar plataformas o enemigos y correr. Esto funciona gracias a un script que detecta las teclas que pulsa el jugador. Cuando se pulsa una tecla, el personaje se mueve dentro del nivel. De esta forma el jugador puede explorar el mapa y seguir avanzando.

```gdscript
extends CharacterBody2D

# Movilidad

@export var speed := 150
@export var run_speed := 260
@export var jump_force := -360
var gravity := 940
```

Velocidad normal: 150 px/s
Velocidad rápida 260
El fuerza de salto está a -360
Gravedad a 940 para simular una caída más natural



Estas son las únicas opciones de movimientos que tiene el jugador.
Idle: Personaje en sí
Caída, salto, correr y caminar

Movimientos:

Idle *Walk* *Run* *Jump* *Fall*

2.5.2 Funcionalidad de recoger monedas

Otra funcionalidad importante del juego es poder recoger monedas. En cada nivel hay varias monedas repartidas por el mapa y el jugador tiene que ir recogiéndolas. Cuando el personaje toca una moneda, el juego detecta el contacto y la moneda desaparece. Al mismo tiempo se suma al contador de monedas recogidas. Esta funcionalidad está implementada completamente.

```
func _ready():
    if GameData.monedas == GameData.monedas_totales:
        titulo_completo = "MISIÓN COMPLETADA"
        sonido.play()
    else:
        titulo_completo = "MISIÓN FALLIDA"

    texto_completo = "Has cogido %d de %d monedas" % [
        GameData.monedas,
        GameData.monedas_totales
    ]
```

GameData es un script global (autoload) que tenemos para guardar información entre escenas, como las monedas recogidas o la vida del jugador.

Script claro del GameData.gd.

Como vemos en la siguiente imagen, el script **GameData** es un tipo **Node** para poder funcionar como Autoload. En él guardamos todas las variables globales del juego:

```
1  extends Node
2
3  var max_health = 100
4  var health = 100
5  var monedas: int = 0
6  var monedas_recogidas: Array = []
7
8  var monedas_totales: int = 0
9
```

2.5.3 Funcionalidad del sistema de vida

El juego también tiene un sistema de vida para el jugador.

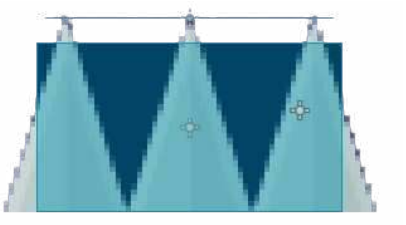
Si el personaje toca un enemigo o un obstáculo del mapa, pierde una parte de su vida.

Si la vida llega a 0, el jugador pierde la partida y debe empezar desde el principio.

Esto hace que el juego sea más desafiante y obliga al jugador a tener cuidado.

Esta funcionalidad está completamente implementada.

En la captura anterior de GameData se pueden ver las variables *health* y *max_health*.



Cuando el jugador está encima de una trampa, recibe daño y el personaje parpadea en rojo.

Esto sirve para que, incluso sin sonido, el jugador pueda identificar visualmente que ha recibido daño.

2.5.4 Funcionalidad del menú del juego

El juego cuenta con un menú inicial desde el cual el jugador puede comenzar la partida y entrar al primer nivel.

Esta funcionalidad está completamente implementada y incluye:

Menú principal

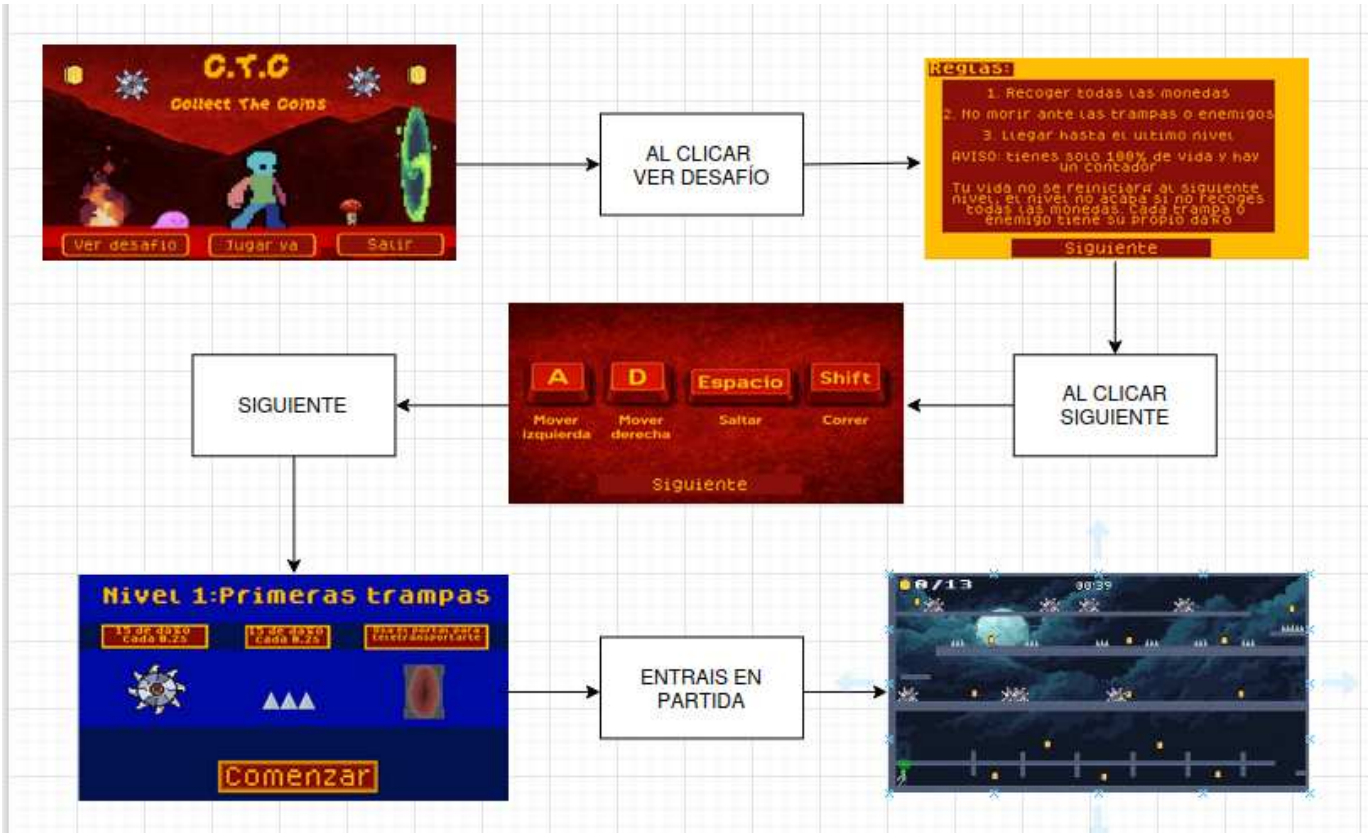
Reglas del desafío

Menú de pausa

Pantalla de misión completada

Pantalla de misión fallida

En el futuro se podría ampliar añadiendo opciones como configuración de sonido o selección directa de niveles.



3 Otros capítulos

A lo largo del desarrollo surgieron distintos obstáculos que obligaron al equipo a replantear partes del trabajo. El problema más recurrente fue el sistema de colisiones, especialmente en las plataformas móviles, donde el jugador las atravesaba o quedaba atrapado. La solución fue ajustar las capas de colisión en el inspector de Godot, aunque requirió varios intentos antes de dar con la causa real del error.

Otro problema fue la pérdida de datos entre escenas: al cambiar de nivel, la vida del jugador se reiniciaba a 100 porque las variables de salud se inicializaban localmente en cada escena. Implementar GameData como Autoload resolvió el problema haciendo que los valores persisten globalmente.

La migración de Godot 4.5 a 4.6 también generó imprevistos, ya que algunos scripts relacionados con animaciones y señales dejaron de funcionar y tuvieron que adaptarse, consumiendo tiempo no planificado. Finalmente, el trabajo compartido en GitHub generó conflictos de archivos en varias ocasiones, lo que llevó al equipo a establecer la norma de avisar antes de hacer push para evitar sobrescribir el trabajo del otro.

Todos estos problemas, aunque frustrantes, fueron una parte fundamental del aprendizaje y ayudaron al equipo a entender mejor el funcionamiento interno de Godot.

4 Conclusiones

En este proyecto, el trabajo final recibió el nombre de *Collect The Coins*, donde el objetivo principal es recoger monedas mientras se evitan obstáculos y enemigos.

A lo largo del desarrollo hemos tenido que aprender a usar nuevas herramientas, organizar el trabajo y resolver los problemas que han ido apareciendo.

El proyecto nos ha servido para entender mejor cómo se crea un videojuego desde cero, no solo a nivel de programación, sino también en el diseño de niveles, la jugabilidad y la organización del trabajo. Aunque al principio presentaba una complejidad elevada, poco a poco hemos ido avanzando hasta conseguir un juego funcional.

También hemos aprendido la importancia de probar el juego constantemente, ya que muchos errores solo se detectan cuando se juega. Esto nos ha ayudado a mejorar el resultado final y a hacer el juego más estable.

4.1 Conclusiones generales del proyecto

Desarrollar Collect The Coins ha sido una experiencia que va más allá de la programación. A lo largo del proyecto se aprendió a utilizar un motor de videojuegos real, a organizarse como equipo, a resolver problemas técnicos y a tomar decisiones cuando algo no funcionaba como se esperaba. Ver el juego terminado y funcional, sabiendo que se construyó desde cero, es el resultado del que más se enorgullece el equipo.

4.2 Consecución de los objetivos

Recolección de monedas: Conseguido. El jugador puede recoger todas las monedas del mapa.

Cuenta atrás: Conseguido. Hay un temporizador que añade presión y hace el juego más dinámico.

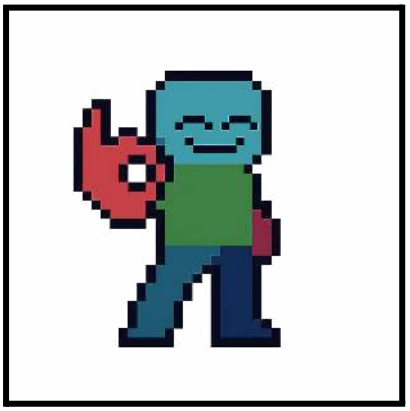
Sistema de vida: Conseguido. El jugador empieza con el 100% de vida y la va perdiendo al contacto con los peligros. No reinicia su vida al siguiente nivel

Progresión sin regenerar vida: Conseguido. La vida no se reinicia al pasar al siguiente nivel

Interfaz por mapa: Conseguido. Cada nivel tiene su propia interfaz y diseño.

Enemigos y peligros variados: Conseguido. Se han implementado sierras, enemigos móviles, pinchos, veneno y fuego.

En general, todos los objetivos planteados al inicio del proyecto se han cumplido satisfactoriamente.



4.3 Valoración de la metodología y planificación

La metodología seguida funcionó bien en términos generales, aunque no siempre se ajustó exactamente a lo planificado al inicio.

El diagrama de Gantt fue útil para obtener una visión global del proyecto en las primeras fases, pero perdió relevancia a medida que el desarrollo avanzaba, ya que las tareas surgían de forma más espontánea según las necesidades del juego.

GitHub fue una herramienta clave para mantener la coordinación entre los miembros del equipo, permitiendo trabajar de forma independiente sin perder el progreso.

Los ajustes realizados sobre la marcha, cómo rehacer el sistema de colisiones o rediseñar algunas trampas, formaron parte natural del proceso y enseñaron al equipo a adaptarse rápidamente ante los imprevistos.

En definitiva, la metodología elegida fue adecuada para un proyecto de este tamaño, permitiendo avanzar de forma progresiva hasta alcanzar un resultado funcional y completo.

4.4 Visión de futuro

El juego presenta un gran potencial de crecimiento de cara a futuras versiones. Una de las mejoras prioritarias sería implementar un sistema de registro de usuarios que permita guardar el progreso de forma permanente, algo que no pudo desarrollarse durante el proyecto por limitaciones de tiempo y conocimientos técnicos.

También se contemplan mejoras a nivel de contenido y personalización, como un apartado de skins para que el jugador pueda modificar la apariencia de su personaje, así como habilidades especiales que enriquezcan la jugabilidad y la hagan más variada.

A nivel de contenido, se planea añadir más niveles con entornos y diseños diferentes, nuevos tipos de enemigos con comportamientos más avanzados, y mejoras visuales y sonoras que hagan la experiencia más inmersiva.

En conclusión, Collect the Coins es una base sólida sobre la que seguir construyendo, y quien sabe si se seguirá desarrollándose más allá de este proyecto.

5. Glosario

Animated Sprite: Es un nodo de Godot que sirve para poner animaciones, como las del personaje, los enemigos o las trampas sierras, fuego o los pinchos.

Autoload: Sistema de Godot que permite tener un script que funciona en todas las escenas. En Collect the Coins lo usamos para GameData, que guarda cosas como la vida o las monedas.

Colisión: Es cuando el jugador toca una moneda, un enemigo o una trampa. Godot detecta ese choque y hace que pase como sumar monedas, perder vida. Así como las consecuencias correspondientes.

Cuenta atrás (Timer): Un temporizador que marca el tiempo que queda para recoger todas las monedas antes de que se acabe.

GDScript: Lenguaje de programación propio de Godot, similar a Python, utilizado para programar todas las mecánicas del juego como el movimiento, las colisiones o el sistema de vida.

Gameplay: La experiencia de jugar: cómo se controla el personaje, cómo se recogen las monedas, cómo se esquivan los enemigos y lo fluido que va todo.

Git: Sistema de control de versiones que permite registrar los cambios del código y volver a versiones anteriores si algo falla.

GitHub: Página web donde subimos el proyecto para tenerlo en la nube y trabajar los dos sin perder nada.

Hitbox: Área invisible que rodea al personaje, los enemigos y las trampas, y que Godot usa para detectar cuándo se tocan y aplicar el daño correspondiente.

Indie: Término que describe videojuegos desarrollados de forma independiente, sin el respaldo de grandes empresas. Collect the Coins es un proyecto indie.

Godot Engine: El motor que usamos para hacer el juego. Es gratis, de código abierto y muy bueno para juegos en 2D.

Scene (Escena): Cada parte del juego en Godot. Por ejemplo: cada nivel, el menú principal y la interfaz son escenas distintas.

Sprite: Imagen o conjunto de imágenes que representan visualmente a un personaje u objeto dentro del juego, como el jugador, las monedas o los enemigos.

Tilemap: Sistema para construir los niveles usando piezas pequeñas (tiles) que forman el suelo, las plataformas y el escenario.

6. Bibliografía

En este apartado se recogen todas las fuentes que hemos utilizado durante el desarrollo del proyecto **Collect the Coins**. Estas fuentes nos han servido para aprender a usar las herramientas, entender mejor cómo funciona Godot y también para resolver errores que nos iban saliendo mientras hacíamos el juego.

Sobre todo hemos usado vídeos de YouTube, porque explican las cosas paso a paso y es más fácil seguirlos mientras vas probando en el propio programa. Muchas veces nos pasaba que algo no funcionaba y buscando un vídeo parecido encontrábamos la solución o una idea para hacerlo mejor.

También hemos usado la documentación oficial de Godot Engine y páginas como GitHub o itch.io para apoyarnos en el desarrollo del proyecto.

Colisiones al personaje:

Renato Meyer. (2024). Aprende GODOT en 7 minutos

https://www.youtube.com/watch?v=Wa4yO92SXkc

En este vídeo aprendemos cómo hacer que el personaje detecte los objetos del entorno, como el suelo o las plataformas. Gracias a esto pudimos evitar que el personaje atraviese cosas y hacer que el salto funcione bien.

Como poner enemigos en Godot:

Academia de Videojuegos. (2025). Enemigos. Plataformas 2D con Godot.

https://www.youtube.com/watch?v=Z5PmBJG9PRo

Aquí vimos cómo crear enemigos básicos y hacer que se muevan o sigan un comportamiento. Esto nos ayudó a añadir dificultad al juego.

Como poner un contador de monedas dentro del juego:

LuisCanary. (2022). Juego de Plataformas 2D/Godot Tutorial/UI-HUD Monedas/Godot/5-Cap/Programacion videojuegos

https://www.youtube.com/watch?v=o3NCad3mIzY

Este vídeo nos enseñó a mostrar en pantalla cuántas monedas lleva el jugador. Así el jugador sabe cuánto le falta para completar el nivel.

Godot y sus funciones:

Findemor. (2024). Cómo usar Godot y Aprender desde CERO a hacer juegos

https://www.youtube.com/watch?v=-_LiMyZGoXw

Nos sirvió para entender mejor cómo funciona el motor en general, como escenas, nodos y scripts.

Creación de monedas:

LuisCanary. (2022). Juego de Plataformas 2D/Godot Tutorial/Recoger Monedas/Godot/4-Cap/Programacion videojuegos

https://www.youtube.com/watch?v=u99myfBJDlc

En qué consiste el Script:

Findemor. (2024). Cómo Aprender a Programar desde CERO con GDScript y Godot para videojuegos

https://www.youtube.com/watch?v=fQf_ocKIHWM

Aquí aprendimos lo básico del lenguaje que usamos para programar el juego. Nos ayudó a empezar desde cero.

Otras fuentes utilizadas

Godot Engine (2026). Documentación oficial de Godot.

https://docs.godotengine.org/es/stable/

GitHub (2026). Plataforma utilizada para guardar el proyecto y trabajar en equipo.

https://github.com

YouTube (2026). Vídeos y tutoriales sobre desarrollo de videojuegos en Godot.

https://www.youtube.com/results?search_query=crear+videojuegos+en+godot

Itch.io (2026). Plataforma utilizada para descargar objetos y subirlos al juego.

https://itch.io

IAS utilizadas

Chat GPT

Lo hemos usado para resolver dudas rápidas, entender errores del código y ayudarnos a redactar algunos apartados del proyecto cuando no sabíamos cómo explicarlos bien.

Google Gemini

Nos ha servido como apoyo para buscar información y comparar explicaciones sobre algunas partes del desarrollo.

Claude

Lo hemos utilizado para entender mejor algunos conceptos y tener otra forma de explicación cuando algo no nos quedaba claro.

Grok

Se ha usado como apoyo para resolver dudas puntuales y ver diferentes formas de hacer algunas cosas.

Microsoft Copilot

Nos ha ayudado sobre todo en la parte de programación, sugiriendo código y ayudando a detectar errores.

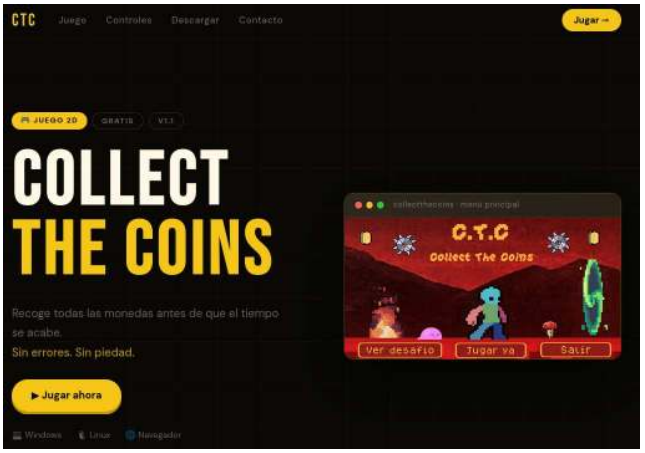
En general, todas estas fuentes nos han ayudado mucho en la resolución de dudas y en el aprendizaje durante el desarrollo del juego. Muchas cosas las hemos aprendido viendo vídeos y probando por nuestra cuenta hasta que funcionaba.

7 Anexos

Explicación de la página web:

1. Inicio

Al acceder a la página, se presenta el juego con su nombre completo y una breve descripción: "Recoge todas las monedas antes de que el tiempo se acabe. Sin errores. Sin piedad." Es un juego 2D gratuito. Desde esta sección ya se puede acceder directamente al juego mediante el botón "Jugar ahora", con disponibilidad para Windows, Linux y navegador.



2. El juego

Se muestran las características principales del juego de forma visual:

- Plataformas: Windows + Linux
- Precio: 100% gratis
- Año: 2026
- Desarrolladores: 2

3. Controles

Se listan todos los controles necesarios para jugar:

4. Descarga

El juego es gratuito en todas las plataformas, sin registro, sin pago y sin trampa. Hay tres opciones disponibles:

5. Comparte

Una vez jugado, la página invita a compartirlo con amigos a través de:

6. Contacto

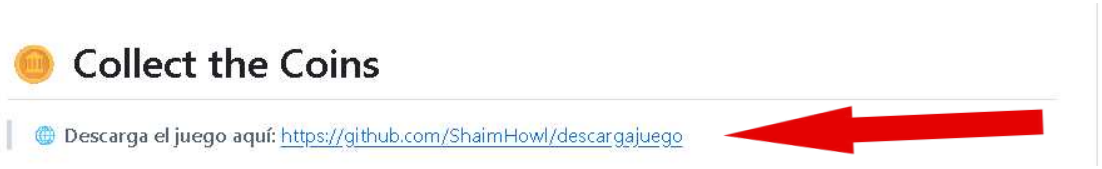
La página dispone de un formulario de contacto para bugs, ideas o cualquier consulta. Los campos disponibles son: Nombre, Correo y Mensaje, con un botón de envío.

Pasos para descargar el juego:

Primera opción:

Paso 1 - Acceder al repositorio principal Dirígete al repositorio del juego en GitHub: 🔗GitHub Repositorio

Paso 2 - Localizar el enlace de descarga Una vez dentro, desplázate hacia abajo por la página. Encontrarás un enlace que te redirigirá al repositorio de descarga: 🔗



Paso 3 - En este repositorio, desplázate un poco hacia abajo para acceder a la página web del juego. Desde allí verás el enlace para ir a la página web:

Podrás:

Jugar directamente desde el navegador, sin necesidad de instalar nada

Descargar la versión para Windows y ejecutarlo directamente

Descargar la versión para Ubuntu:

Atención: No podrás ejecutar el juego desde Linux sin haber completado estos 5 pasos.

1. Extrae el juego. Una vez descargado, debes extraer todo el directorio.
2. Entra dentro del directorio y ábrelo en la terminal desde donde estás.
3. Aplica este permiso:

```
chmod +x CollectTheCoins.x86_64
```

4. Una vez aplicados los permisos, deberás ejecutar el juego desde la terminal con este comando:

```
./CollectTheCoins.x86_64
```

¿Dudas o sugerencias? Desde la misma página también encontrarás un formulario de contacto para enviarnos un correo con cualquier duda o sugerencia de mejora.

Desde la página web también puedes compartir el juego con tus amigos a través de las siguientes plataformas:

- Twitter
- Reddit
- Whatsapp

Segunda opcion:

Paso 1 — Acceder al repositorio principal Dirígete al repositorio del juego en GitHub: Repositorio del GitHub

---

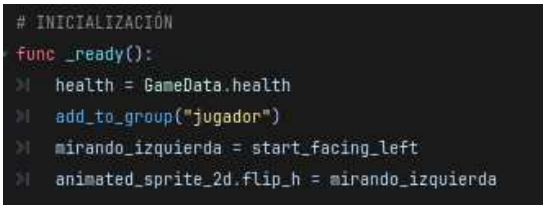
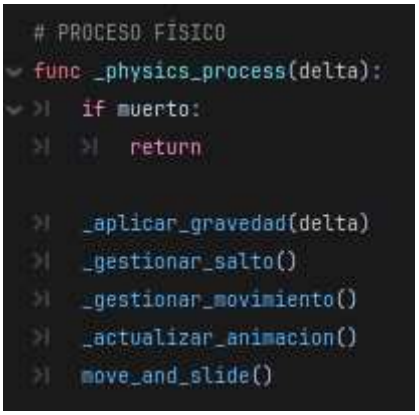
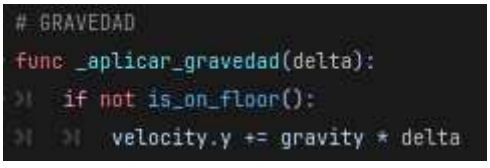
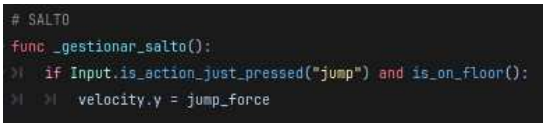
Paso 2 — Obtener el enlace SSH. Una vez dentro, haz clic en el botón verde `Code`. Se abrirá un desplegable, selecciona la pestaña SSH y copia el enlace que aparece.

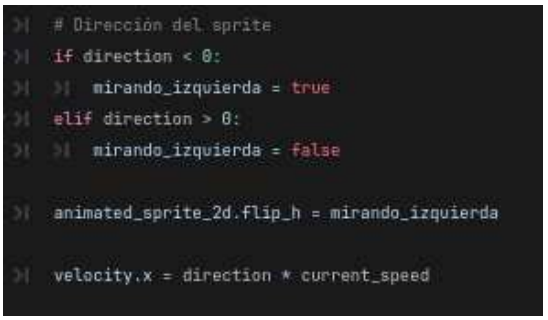
Paso 3 — Clonar el repositorio Abre una terminal y ejecuta el siguiente comando, sustituyendo enlace por el que copiaste:

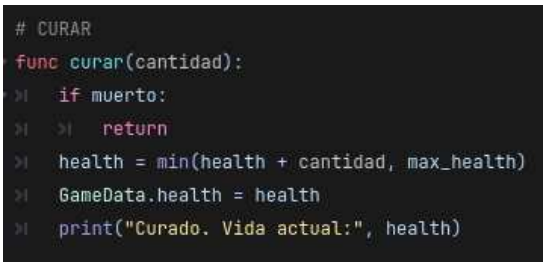
Único requisito es tener el motor de Godot Engine para importar el proyecto y jugarlo

Scripts:

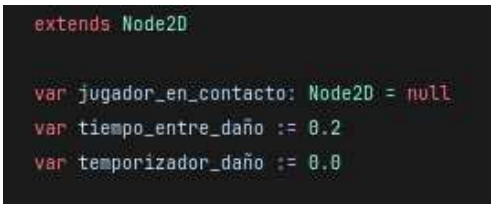
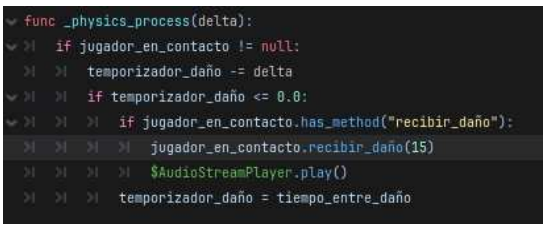
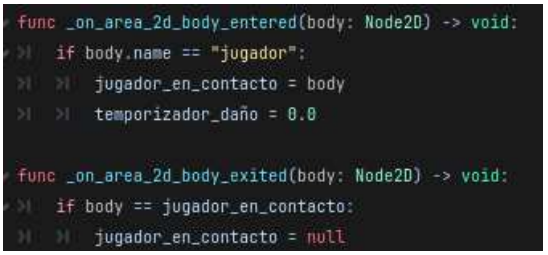
Jugador:

	**speed =** Velocidad de caminar **run speed =** Velocidad máxima al correr **jump force =** Que tan alto salta **gravity =** La fuerza que empuja al jugador hacia abajo cuando está en el aire **max health =** Vida máxima **start facing left =** Decide si el personaje empieza por la izquierda **tiempo invulnerabilidad y flash =** Cuánto dura el tiempo de espera tras ser golpeado y el efecto visual de parpadeo **health: int =** Vida actual el jugador **muerto =** Un valor verdadero/falso para saber si el jugador ha perdido toda su vida. **invulnerabilidad =** Se activa para que no reciba mucho daño al instante y no muera enseguida
	Gamedata es el script global que nos ayuda a que la vida no se reinicie al siguiente nivel. Añade a un grupo jugador para que el resto de objetos también pertenezcan a este grupo para infligir daño. Podemos decidir en el inspector desde que lado empieza a ver el jugador.
	Si el jugador muere para (sí está en true), que no siga haciendo nada, se queda congelado. Aplica todas las funciones
	En general sirve para que el personaje no esté flotando.
	Godot detecta si acabas de pulsar la tecla de salto (por ejemplo, el espacio).

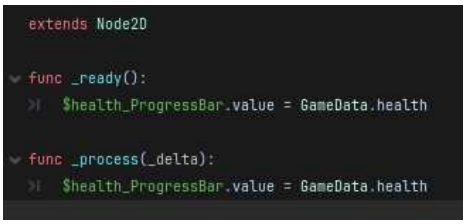
	Si la dirección es menor que uno (-1), el código marca que estamos mirando a la izquierda y si es mayor que 0 está mirando hacia la derecha.
	Si está en el suelo y está en 0, se aplica Idle, sino velocity.x = run:speed se anima run, abs es para que la velocidad se invierta, si ninguno ha funcionado, entonces anima “walk”. Si ninguno de los anteriores ha funcionado y es menor que 0, quiere decir que es un -1 subiendo hacia el 0, anima “jump”, sino anima fall que es la animación de caída.
	Si el personaje ya está muerto: No se le puede hacer más daño, así que ignora el golpe.Si el personaje es invulnerable: Si tiene el "escudo" activado (y el ataque no es especial), el golpe rebota y no pasa nada. Se le resta la fuerza del golpe a la vida actual del personaje. Se actualiza el marcador global (para que el juego recuerde cuánta vida te queda si cambias de pantalla). Se imprime un mensaje interno para que los desarrolladores puedan ver la vida restante. Activa un escudo temporal: Para que el personaje no reciba 50 golpes por segundo, el juego lo vuelve "invulnerable" un breve momento (como cuando el personaje parpadea en los juegos clásicos). El juego mira el marcador. Si la vida llegó a cero o menos, se activa la animación o el menú de muerte.

	Si estás muerto ya no haces nada, max health es la vida máxima. health es la vida actual + cantidad, vida que voy a recibir. El min escoge el valor mínimo
	Vector2 = Detiene cualquier movimiento del jugador, devuelve el color original al jugador si está parpadeando. Animacion vacia. is inside tree = verifica si de verdad está muerto el personaje para no causar errores y cambia la escena a misión fallida

Pinchos:

	La primera guarda si hay un jugador tocando la trampa en ese momento. La segunda define cada cuántos segundos se aplica el daño, en nuestro caso cada 0.2 segundos. Y la tercera funciona como un cronómetro interno que cuenta el tiempo y cuando llega a cero aplica el daño y vuelve a empezar.
	Esta función se ejecuta continuamente mientras el juego está corriendo. Primero comprueba si hay un jugador tocando la trampa. Si es así, va restando tiempo al cronómetro y cuando llega a cero le aplica 15 puntos de daño al jugador, reproduce el sonido de golpe y reinicia el cronómetro para repetir el proceso.
	Estas dos funciones controlan cuando el jugador entra y sale de la trampa. Cuando entra, guarda la referencia al jugador y empieza el cronómetro de daño. Cuando sale, vacía esa referencia y la trampa deja de hacer daño.

Vida que va vinculada con el Script de gamedata:

	Este script conecta la barra de vida visual con el valor real de la vida del jugador. Al inicio la sincroniza, y después se actualiza continuamente cada frame para que cuando el jugador reciba daño, la barra baje en tiempo real.