Institut Puig Castellar
Santa Coloma de Gramenet

Sonic The Hedgehog Remastered
Projecte de desenvolupament
CFGM Sistemes Microinformàtics i Xarxes

Unai Meneses i Angel Perez
Grup: 2B
Curs acadèmic: SMX 2025-2026

Resum del projecte (màxim 250 paraules):

Temàtica del projecte: Aquest projecte se centra en el desenvolupament d'una versió remasteritzada del videojoc clàssic Sonic The Hedgehog, adaptada per a ordinador amb tecnologies modernes.

Objectiu del projecte: L'objectiu principal és crear una experiència nostàlgica per als jugadors originals mentre s'atreu un nou públic interessat en jocs de plataformes clàssics. Es pretén mantenir la mecànica de plataformes 2D característica, incloent-hi la velocitat i col·lecció d'anells, però incorporant gràfics millorats, animacions més fluides i controls optimitzats per a teclat i ratolí.

Metodologia seguida: El desenvolupament es basa en la implementació de programació orientada a objectes per crear diversos nivells jugables, un sistema de puntuacions, efectes sonors i música, juntament amb una interfície d'usuari intuïtiva. S'apliquen principis de disseny d'interfícies interactives per garantir una experiència d'usuari òptima.

Resum de les conclusions: El projecte permet adquirir competències essencials en desenvolupament de videojocs, programació orientada a objectes i disseny d'interfícies. La remasterització exitosa demostra la viabilitat de modernitzar clàssics del gaming mantenint l'essència original mentre s'incorporen millores tècniques contemporànies, oferint així una experiència que combina nostalgia amb qualitat visual i tècnica actual.

Paraules clau (entre 4 i 8):

Sonic The Hedgehog, remasterització, videojocs de plataformes, programació orientada a objectes, desenvolupament de videojocs, interfície d'usuari

Abstract:

Project theme: This project focuses on developing a remastered version of the classic video game Sonic The Hedgehog, adapted for computer with modern technologies.

Project objective: The main objective is to create a nostalgic experience for original players while attracting a new audience interested in classic platform games. The aim is to maintain the characteristic 2D platform mechanics, including speed and ring collection, while incorporating improved graphics, smoother animations, and optimized keyboard and mouse controls.

Methodology: Development is based on object-oriented programming implementation to create various playable levels, a scoring system, sound effects and music, along with an intuitive user interface. Interactive interface design principles are applied to ensure optimal user experience.

Summary of conclusions: The project enables the acquisition of essential skills in video game development, object-oriented programming, and interface design. The successful remastering demonstrates the feasibility of modernizing gaming classics while maintaining the original essence and incorporating contemporary technical improvements, thus offering an experience that combines nostalgia with current visual and technical quality.

Keywords (entre 4 i 8):

Sonic The Hedgehog, remastering, platform games, object-oriented programming, video game development, user interface

Índex

Llista de figures

1 Introducció

Aquest projecte consisteix en desenvolupar una versió remasteritzada de Sonic The Hedgehog per a PC utilitzant Godot Engine 4. L'objectiu és recrear l'experiència clàssica del joc de plataformes 2D original com la velocitat, recollida d'anells i salts, amb gràfics modernitzats i controls optimitzats per a ordinador. El resultat serà un prototip funcional amb diversos nivells jugables, sistema de puntuació i integració d'àudio que serveixi com a projecte d'aprenentatge en desenvolupament de videojocs.

1.1 Context

Sonic The Hedgehog (1991) no només va crear un nou gènere de plataformes 2D incorporant velocitat, sinó que també va canviar la percepció que es podia tenir dels jocs de plataformes 2D. Es van posar a disposició del públic motors potents com el Godot Engine; així, el desenvolupament de videojocs es va democratitzar. Ara, tothom té accés a eines tan poderoses per als seus projectes 2D amb una àmplia documentació.

La indústria està mostrant una tendència cap a la remasterització de títols clàssics (Sonic Mania, Crash Bandicoot N. Sane Trilogi), demostrant la demanda tant dels jugadors nostàlgics com dels nous públics. La recreació de jocs familiars proporciona especificacions clares que permeten centrar-se en l'aprenentatge tècnic.

Cal destacar que la democratització del desenvolupament de videojocs ha tingut un impacte especialment significatiu en el segment dels jocs indie. Estudis petits i fins i tot desenvolupadors individuals han pogut llançar títols de gran qualitat gràcies a motors gratuïts com Godot. Exemples com Celeste, Undertale o Hollow Knight demostren que un equip reduït pot crear un producte competitiu amb eines accessibles, sempre que hi hagi una visió clara del producte i una bona gestió del projecte.

En aquest context, recrear un clàssic com Sonic The Hedgehog no és només un exercici nostàlgic, sinó una oportunitat real d'aprendre les tècniques que utilitzen els estudis professionals actuals, aplicades a un producte amb especificacions conegudes i una comunitat de referència molt activa.

1.2 Justificació

El projecte ofereix aprenentatge integral en programació orientada a objectes, gestió d'esdeveniments, física 2D i disseny d'interfícies. Treballar amb mecàniques ben documentades permet concentrar-se en la implementació tècnica sense perdre temps en el disseny conceptual del joc.

Des del punt de vista econòmic, totes les eines emprades —Godot Engine, GDScript, Aseprite, GIMP, Audacity i Git— són gratuïtes i de codi obert, eliminant qualsevol barrera econòmica. Això garanteix l'accessibilitat del projecte i el seu potencial de reproducció per part d'altres estudiants o desenvolupadors independents.

La dimensió motivacional és igualment rellevant. Treballar amb un personatge i un joc coneguts per ambdós membres de l'equip ha facilitat la presa de decisions de disseny i ha mantingut el nivell d'implicació alt. La motivació intrínseca és un factor determinant en projectes de desenvolupament de programari, especialment quan l'equip és petit i no hi ha un client extern que marqui els terminis.

Finalment, el projecte té un valor de portafoli clar: un joc funcional i jugable és un producte tangible que es pot mostrar en entrevistes de feina o en processos de

selecció del sector tecnològic, demostrant competències pràctiques que van més enllà dels coneixements teòrics.

1.3 Objectius

1.3.1 Objectiu general

L'objectiu principal d'aquest projecte és el disseny i desenvolupament d'un prototip funcional de videojoc de plataformes en 2D, basat en la franquícia "Sonic The Hedgehog", utilitzant el motor Godot Engine 4.

Aquest objectiu no es limita a la creació del programari lúdic, sinó que se centra en l'adquisició i consolidació de competències tècniques pròpies del desenvolupament d'aplicacions:

- Desenvolupament de Programari: Ús de programació orientada a objectes mitjançant la jerarquia de nodes i scripts en GDScript.

- Gestió de Projectes: Planificació de les fases de producció, des del prototipatge de físiques fins a la integració d'assets finals i el desplegament del producte.

- Treball en Equip: Coordinació per a la integració de diferents mòduls de codi, disseny de nivells i recursos multimèdia en un repositori unificat.

El producte final s'ha concebut com una eina de demostració de capacitats davant d'un tribunal avaluador, assegurant que el sistema compleixi amb els estàndards de qualitat en l'àmbit de SMX

1.3.2 Objectius específics

Per tal d'assolir l'objectiu general i garantir la viabilitat del prototip, s'han definit els següents objectius específics, que abasten des de la programació base fins a la gestió del projecte:

- Desenvolupament de les físiques de moviment: Programar un sistema cinemàtic mitjançant el node CharacterBody2D que gestioni paràmetres d'acceleració, fricció i velocitat màxima. L'objectiu és aconseguir una resposta precisa en el salt i una gestió de col·lisions que eviti errors de penetració en superfícies complexes o rampes.

- Disseny i estructuració de nivells: Crear tres entorns jugables diferenciats que permetin una progressió lògica de la dificultat. Això inclou l'ús de `TileMaps` per al disseny del terreni i la configuració de fons amb desplaçament de paral·laxi per dotar el joc de profunditat visual.

- Lògica de col·leccionables i economia del joc: Implementar un sistema de detecció d'objectes per a la recollida d'anells, el qual ha d'actualitzar en temps real un comptador global de puntuació i gestionar la pèrdua d'aquests recursos quan el jugador rep dany.

- Gestió d'assets gràfics i animació: Importar, configurar i optimitzar els fulls de sprites (spritesheets) tant del personatge principal com dels enemics. Es busca una integració fluida de les animacions mitjançant màquines d'estats que responguin als inputs del jugador i a la lògica de la IA.

- Arquitectura de so i ambientació: Integrar un sistema d'àudio dinàmic que gestioni de forma independent la música de fons (BGM) i els efectes de so (SFX). Cada nivell ha de tenir una identitat sonora pròpia que reaccioni a certs esdeveniments, com salts o la derrota d'enemics.

- Disseny de la interfície d'usuari (UI/HUD): Desenvolupar un menú principal interactiu i una interfície en pantalla (Heads-Up Display) que mostri de manera clara les estadístiques vitals (anells, vides i temps) durant la partida, utilitzant nodes de control de Godot.

- Control de versions i flux de treball: Establir un flux de treball professional mitjançant Git i GitHub, permetent un control de versions rigorós que faciliti la recuperació de codi i la coordinació de les actualitzacions de la documentació tècnica.

- Optimització del rendiment: Monitoritzar i ajustar el codi i la càrrega de recursos per garantir una taxa de refresc constant de 60 FPS (fotogrames per segon), assegurant una experiència d'usuari fluida i sense latència en els controls.

1.4 Estratègia i planificació del projecte

L'estratègia principal d'aquest projecte es basa en el desenvolupament integral d'un producte nou partint de les especificacions mecàniques i visuals d'un referent existent, en aquest cas, el joc original de SEGA. Tot el sistema s'ha construït des de zero utilitzant el motor Godot Engine 4, la qual cosa ha requerit una planificació rigorosa per replicar fidelment el comportament del personatge. Aquesta estratègia ha permès un control total sobre el codi, facilitant la personalització d'elements i l'optimització de recursos per a la plataforma PC.

Pel que fa a la viabilitat, el projecte s'ha analitzat des de quatre perspectives clau. Tècnicament és un objectiu factible, ja que Godot Engine 4 integra totes les eines de física, gestió de tilesets i scripting necessàries. Temporalment, es considera un projecte assolible gràcies a una estructura de fases incrementals amb un Producte Mínim Viable (MVP) clarament definit. Econòmicament, el projecte és altament accessible, ja que es basa exclusivament en programari lliure i recursos gratuïts, eliminant qualsevol barrera de llicències. Finalment, és un repte perfectament

realitzable de forma individual o en grup reduït gràcies a la corba d'aprenentatge del llenguatge GDScript i la modularitat del motor.

La planificació temporal s'ha organitzat en tres etapes fonamentals per assegurar la qualitat del lliurament final. Les dues primeres setmanes s'han dedicat a la investigació tècnica, la recerca d'assets i la configuració de l'entorn de treball i el control de versions. El bloc central del projecte, situat entre les setmanes 3 i 10, s'ha centrat en el desenvolupament iteratiu mitjançant sprints. Aquest mètode de treball ha permès disposar d'una versió executable del joc des de les primeres etapes, facilitant la detecció de possibles conflictes en les col·lisions o bugs en la lògica de joc abans d'avançar a la següent fase. Finalment, les setmanes 11 i 12 s'han reservat per a la fase de Quality Assurance (QA), on s'han realitzat les proves de rendiment per garantir els 60 FPS estables, la correcció final d'errors i la redacció de la documentació tècnica que acompania aquest prototip.

1.5 Metodologia de treball

Pel que fa a la metodologia de treball, s'ha adoptat un enfocament àgil basat en Sprints. Aquesta estructura permet una revisió constant dels objectius i una adaptació ràpida davant de qualsevol imprevist tècnic. La fragmentació del desenvolupament s'ha organitzat de la següent manera:

- Sprints 1-2 (Sistemes base): El focus inicial s'ha centrat en el nucli del joc, desenvolupant el sistema de moviment i la gestió de col·lisions. Aquesta és la fase més crítica, ja que de la precisió de les físiques depèn tota l'experiència posterior.
- Sprints 3-4 (Mecàniques d'interacció): Un cop establert el moviment, s'han implementat els elements interactius com la recollida d'anells i la lògica bàsica dels enemics. Alhora, s'ha construït el primer nivell per testejar aquestes mecàniques en un entorn real.
- Sprints 5-6 (Expansió de contingut): Amb la base tecnològica consolidada, el treball s'ha orientat a la creació de nivells addicionals, incrementant la complexitat del disseny d'escenaris i la varietat de reptes.
- Sprints 7-8 (Finalització i poliment): L'última etapa s'ha dedicat a la interfície d'usuari (UI), la integració de l'apartat sonor i el poliment general (game feel), assegurant que el producte final sigui coherent i estable.

Per a la gestió i coordinació d'aquestes tasques, s'ha utilitzat un conjunt d'eines estàndard en la indústria del programari. El seguiment del flux de treball s'ha realitzat mitjançant taulers Kanban (a través de GitHub Projects o Trello), permetent visualitzar en tot moment l'estat de les tasques (pendents, en procés o finalitzades). El control de versions s'ha gestionat de forma rigorosa amb Git i els repositoris de GitHub, garantint la seguretat del codi i la possibilitat de revertir canvis si fos necessari. Finalment, tota la documentació tècnica i el seguiment dels canvis s'han realitzat utilitzant el format Markdown, assegurant una presentació clara, estructurada i fàcilment editable.

1.6 Estudi econòmic i pressupostari

Eina	**Cost**
Godot Engine 4	0€
GDScripts	0€

Aseprite	0€ (Codi obert)
GIMP	0€
Audacity	0€
Git/GitHub	0€
TOTAL	**0€**

Tot i que les eines utilitzades són gratuïtes i de codi obert, el projecte té un cost indirecte relacionat amb el temps de desenvolupament i els recursos de maquinari. El desenvolupament del videojoc ha requerit aproximadament 120 hores de treball repartides entre programació, disseny gràfic, proves i documentació.

En un entorn professional, aquest temps podria representar un cost econòmic considerable si es comptabilitzessin hores de desenvolupador, dissenyador gràfic i tester. També cal tenir en compte el consum energètic dels equips i el manteniment del maquinari utilitzat durant el projecte.

2 Descripció del projecte

Aquest projecte se centra en la reconstrucció tècnica i visual del videojoc "Sonic The Hedgehog". La finalitat és crear un producte que s'executi de manera nativa en plataformes PC, aprofitant les millores de rendiment que ofereixen els motors de jocs actuals. El projecte no busca només replicar l'aparença estètica, sinó aprofundir en la lògica de programació que permet un moviment a alta velocitat en entorns 2D.

El joc es desenvolupa sota una arquitectura basada en el motor Godot Engine 4, la qual cosa permet una gestió eficient de la memòria i una execució fluida. S'ha posat un èmfasi especial en la fidelitat de les mecàniques, com el sistema de física d'inèrcia (on el personatge tarda a agafar velocitat i a frenar) i la interacció amb elements clàssics de l'entorn com les rampes, els anells i els enemics robòtics. Aquest prototip serveix com a demostració de la capacitat de crear sistemes complexos d'interacció en temps real.

2.1 Anàlisi de requisits

L'anàlisi de requisits és una fase fonamental per definir les metes tècniques del projecte i garantir que el producte final compleixi amb les expectatives de rendiment i jugabilitat. Aquests requeriments s'han extret a partir de l'estudi de les mecàniques originals de Sonic, adaptant-les a les possibilitats de Godot Engine 4.

2.1.1 Requisits funcionals

Els requisits funcionals defineixen les accions específiques que el sistema ha de ser capaç d'executar. Aquests s'han definit seguint el model de comportament del joc original de 1991, adaptant-los a les capacitats de l'usuari actual de PC.

- RF01 - Sistema de moviment i físiques del personatge: El nucli del joc es basa en un control dinàmic. El jugador ha de poder moure el personatge horitzontalment amb una acceleració progressiva que simuli la inèrcia. Així mateix, s'ha d'implementar un sistema de salt parabòlic afectat per la gravetat i

el "Spin Dash", una mecànica que permet carregar energia cinètica en repòs per sortir disparat a alta velocitat.

- RF02 - Sistema de col·lisions i interacció amb l'entorn: El motor ha de garantir una detecció precisa amb les superfícies del TileMap. Això inclou el càlcul d'angles en pendents i rampes per ajustar la velocitat de Sonic, així com la detecció de col·lisions amb enemics o trampes que resultin en la pèrdua de recursos o vida.
- RF03 - Mecànica de col·lecció i dispersió d'anells: Els anells actuen com a sistema de salut i puntuació. El sistema ha de permetre la recollida per contacte i, de manera crucial, gestionar la dispersió física dels anells en totes direccions quan el personatge rep dany.
- RF04 - Intel·ligència Artificial d'enemics (Badniks): S'han d'implementar almenys dos tipus d'enemics amb patrons de moviment autònoms (patrulla). El jugador ha de poder interactuar amb ells de forma ofensiva, destruint-los mitjançant un salt o el "Spin Dash", activant una animació d'explosió i l'alliberament de puntuació.
- RF05 - Disseny i progressió de nivells: El joc ha de constar de dos nivells amb estètiques i configuracions de plataformes diferenciades. Cada nivell ha de tenir un punt d'entrada i una meta clarament identificables, amb una corba de dificultat creixent que posi a prova les habilitats apreses pel jugador.
- RF06 - Sistema de puntuació i rànquing: Es calcularà una puntuació basada en tres variables: anells recollits, enemics vençuts i una bonificació de temps per rapidesa. Aquestes dades s'han d'emmagatzemar per mostrar una taula de puntuacions altes al final de la sessió.
- RF07 - Interfície d'usuari (UI/HUD): El sistema ha de disposar d'un menú principal funcional (Jugar, Opcions, Sortir) i un HUD en temps real. El HUD mostrarà de forma constant el recompte d'anells, el temps transcorregut, la puntuació actual i les vides restants.
- RF08 - Gestió d'àudio dinàmic: Integració de música de fons (looping) i efectes sonors (SFX) disparats per esdeveniments concrets (salt, recollida d'anells, explosió d'enemics). S'inclourà un panell de control per ajustar el volum de manera personalitzada.
- RF09 - Lògica de vides i estat de "Game Over": El jugador disposarà d'un comptador de 3 vides inicials. El sistema gestionarà la mort del personatge en cas de contacte amb un perill sense anells al marcador, amb la possibilitat de guaniar una vida extra cada 30 anells. En arribar a zero vides, es disparará la pantalla de Game Over.
- RF10 - Sistema de Checkpoints: Per evitar la frustració de l'usuari, s'implementaran punts de control intermedis. En perdre una vida, si el jugador ha activat un checkpoint, reapareixerà en aquesta posició en lloc de l'inici del nivell.

2.1.2 Requisits no funcionals

Aquests requisits determinen la qualitat de l'experiència tècnica i l'estabilitat de l'aplicació, aspectes crítics en un projecte d'Administració de Sistemes Informàtics.

- RNF01 - Rendiment i optimització: El programari s'ha d'executar a 60 FPS estables en equips de gamma mitjana. Els temps de càrrega entre escenes han de ser mínims (inferiors a 3 segons) per mantenir el ritme de joc.

- RNF02 - Usabilitat i experiència d'usuari:

  Els controls han de ser intuïtius. El sistema oferirà missatges d'ajuda contextuals i la possibilitat de remapejar les tecles per adaptar-se a les preferències de cada usuari.

- RNF03 - Portabilitat i estàndards visuals:

  L'aplicació serà nativa per a Linux (Ubuntu 20.04 o superior) i suportarà resolucions estàndard de 1920x1080 píxels, assegurant que els elements del HUD no es deformin en diferents resolucions de pantalla.

- RNF04 - Mantenibilitat i codi net:

  El desenvolupament seguirà els principis de la Programació Orientada a Objectes (OOP), amb un codi modular i comentat. S'utilitzarà Git com a sistema de control de versions per garantir la traçabilitat de cada canvi.

- RNF05 - Seguretat i integritat:

  El joc no recopilarà dades privades de l'usuari. Les puntuacions locals s'emmagatzemaran en fitxers amb un format que n'asseguri la integritat, evitant modificacions externes trivials.

- RNF06 - Escalabilitat de l'arquitectura:

  L'estructura de nodes de Godot s'ha de dissenyar de manera que sigui senzill afegir nous nivells, enemics o power-ups sense haver de reescriure el nucli del motor del personatge.

- RNF07 - Accessibilitat:

  S'inclouran ajustos visuals com el contrast i la brillantor, així com instruccions clares i llegibles, per permetre que un ventall més ampli d'usuaris pugui gaudir del producte sense barreres.

2.1 Previsió de tasques d'investigació

Tot i que el nucli d'aquest projecte és el desenvolupament tècnic, l'èxit de la implementació depèn directament d'una fase prèvia d'estudi i anàlisi. Donada la complexitat del comportament del personatge original, s'han definit les següents línies d'investigació abans de procedir a la fase de programació:

- 01 - Estudi de mecàniques de moviment clàssiques: Per tal de replicar la jugabilitat original, és imperatiu estudiar conceptes com l'acceleració progressiva i la inèrcia en funció del pendent. La tasca consisteix a desglossar els valors de velocitat màxima, força de salt i fricció horitzontal.
  - Resultat: Un document tècnic amb la matriu de valors i constants que definiran el comportament de Sonic.

- 02 - Anàlisi de nodes de física en Godot 4: S'ha d'investigar la diferència de rendiment i control entre els nodes CharacterBody2D (moviment cinemàtic programat) i RigidBody2D (físiques pures gestionades pel motor). L'objectiu és determinar quin node ofereix millor precisió per a un personatge d'alta velocitat.
    - Resultat: Decisió tècnica justificada sobre l'arquitectura física del personatge.
- 03 - Gestió avançada de col·lisions (Tunneleing): Un problema comú en jocs de gran velocitat és el tunnelling (quan un objecte travessa una paret en un sol frame). Cal investigar tècniques com el Continuous Collision Detection (CCD) o l'ajust dels Safe Margins en Godot per evitar aquests errors.
    - Resultat: Implementació d'un sistema de col·lisions robust que garanteix la integritat del personatge en moviments ràpids.
- 04 - Optimització i gestió de recursos en pantalla: Investigar mètodes per mantenir un rendiment constant de 60 FPS, especialment en nivells amb gran densitat d'objectes. S'estudiaran tècniques de Visibility Enabler per desactivar processos d'objectes que queden fora de la càmera.
    - Resultat: Guia interna d'optimització d'escenes per maximitzar la taxa de fotogrames.
- 05 - Implementació de sistemes d'animació 2D: Estudiar el funcionament del node AnimatedSprite2D en comparació amb Animation Player. Es busca determinar quina eina permet una transició més fluida entre estats (córrer, saltar, frenar) mitjançant màquines d'estats.
    - Resultat: Creació d'un flux de treball eficient per a la integració i sincronització de fulls de sprites.
- 06 - Arquitectura del sistema d'àudio: Investigació sobre la gestió de busos d'àudio en Godot mitjançant AudioStreamPlaier. L'objectiu és aprendre a separar els canals de música de fons i efectes especials per permetre ajustos de volum independents.
    - Resultat: Documentació d'ús del sistema d'àudio i configuració de busos.
- 07 - Disseny de nivells mitjançant TileMaps: Exploració de les noves eines de TileSet i TileMap de Godot 4. S'ha d'estudiar com definir capes de col·lisió i metadades en els mosaics per agilitzar el disseny d'escenaris complexos.
    - Resultat: Establiment d'una metodologia de disseny de nivells ràpida i escalable.
- 08 - Persistència de dades i seguretat: Anàlisi comparativa entre l'ús de fitxers JSON (lectura fàcil) i formats binaris (més segurs) per al guardat de puntuacions i configuracions. S'investigaran mètodes de serialització i xifratge bàsic per protegir els fitxers de guardat.
    - Resultat: Implementació d'un sistema de guardat segur i eficient per a les dades de l'usuari.

2.2 Tecnologies

En aquesta secció es detallen les eines de programari i els motors de desenvolupament que s'han considerat per a la realització del projecte. L'elecció de la tecnologia és un pas crític, ja que determina no només el llenguatge de programació, sinó també la facilitat per gestionar les físiques 2D, que són l'element central d'un joc de Sonic.

2.2.1 Comparativa de les tecnologies valorades

Abans d'iniciar el desenvolupament, s'ha realitzat una anàlisi comparativa entre tres dels motors més rellevants en la indústria del desenvolupament independent: Godot Engine, Uniti i Defold. Aquesta comparativa permet entendre els punts forts i febles de cada plataforma respecte a les necessitats específiques de la remasterització de Sonic.

Característiques	**GodotEngine**	**Uniti**	**Defold**
Tipus	Motor 2D/3D Open-source	Motor 2D/3D - Versió Free	Motor 2D - Gratuït
Llenguatge	GDScript, C#, C++	C#	Lua
Pros	Motor físic 2D natiu i altament personalitzable. Sistema de nodes ideal per a jocs de plataformes ràpids. Excel·lent gestió de col·lisions, rampes i moviments tipus dash.	Motor molt sòlid amb eines de desenvolupament avançades. Disposa de físiques robustes i una gran quantitat de recursos i tutorials disponibles per a la comunitat.	Motor extremadament lleuger i ràpid en l'execució. Eines especialitzades exclusivament en entorns 2D amb un sistema d'exportació molt simple.

Contres	Pot requerir ajustos manuals en determinats càlculs de física. Tot i ser molt complet, la part 3D és menys madura que la 2D.	La configuració del moviment cinemàtic és més complexa. Resulta un motor més pesat per a projectes de mitjana escala.	La comunitat d'usuaris és més petita. Existeixen pocs tutorials específics per a mecàniques de plataformes d'alta velocitat.

2.2.2 Tecnologies escollides

Una vegada analitzades les diferents alternatives, s'ha definit el conjunt d'eines que conformaran l'ecosistema de desenvolupament del projecte. S'ha prioritzat l'ús de programari de codi obert i eines gratuïtes que ofereixin resultats de nivell professional.

Motor de joc i Llenguatge de programació

El nucli tecnològic del projecte serà Godot Engine 4.2, utilitzant GDScript com a llenguatge de programació principal. Les raons d'aquesta elecció són:

- Optimització i Especialització: Godot destaca per tenir un motor de renderitzat 2D independent, la qual cosa permet una gestió de píxels i col·lisions molt més precisa que en motors generalistes.
- Llenguatge de Programació: S'ha escollit GDScript per la seva integració nativa i perfecta amb l'arquitectura de nodes del motor. La seva sintaxi, clara i llegible (molt similar a Python), permet un cicle de desenvolupament ràpid i un rendiment òptim per a les necessitats d'un joc de plataformes d'alta velocitat.
- Accessibilitat Tècnica: En ser un entorn multiplataforma, gratuït i amb un editor integrat molt lleuger, facilita el treball en diferents estacions sense necessitat de grans requeriments de maquinari.

Disseny Gràfic i Multimèdia

Per a la creació i gestió dels recursos visuals i sonors, s'ha optat per una combinació d'eines especialitzades:

- Aseprite: S'utilitzarà com a eina principal per al disseny de sprites i la creació d'animacions frame-by-frame. És l'estàndard de la indústria per al pixel art gràcies a la seva gestió eficient de capes i línies de temps.
- GIMP (GNU Image Manipulation Program): S'emprarà per al disseny d'elements de la interfície d'usuari (UI) i el postprocessament de textures de fons, aprofitant la seva potència en l'edició de mapes de bits.
- Audacity: Aquesta eina de codi obert serà l'encarregada de l'edició, retall i optimització dels efectes sonors i les pistes de música, garantint que el format de l'àudio sigui compatible amb el motor de joc sense sacrificar qualitat.

Gestió de Projecte i Control de Versions

Per garantir la seguretat del codi i una organització eficient de les tasques, s'utilitzaran els següents sistemes:

- Git i GitHub: S'ha establert Git com el sistema estàndard per al control de versions, permetent mantenir un historial de canvis detallat. GitHub actuarà com a repositori remot, assegurant la integritat del codi i la facilitat de desplegament.
- GitHub Projects: Es farà servir com a eina de gestió de projecte. En estar totalment integrat amb el repositori, permet vincular directament les tasques (incidències) amb els canvis de codi, utilitzant un tauler visual que facilita el seguiment del progrés durant els diferents sprints.

2.3 Estructura del projecte

L'arquitectura del projecte s'ha dissenyat seguint les millors pràctiques del desenvolupament de videojocs moderns, aprofitant la modularitat i el sistema basat en escenes que ofereix Godot Engine 4. L'estructura es fonamenta en una organització lògica de fitxers i una jerarquia de nodes que permet mantenir el codi desacoblat,

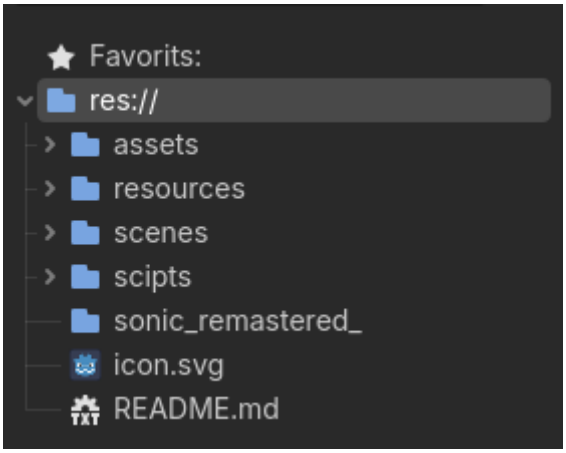
facilitant així l'escalabilitat del producte i un manteniment eficient mitjançant el control de versions amb Git i GitHub.

Pel que fa a l'organització interna, el projecte es divideix en carpetes especialitzades per optimitzar el flux de treball: la carpeta scenes conté els fitxers .tscn que defineixen les entitats (jugador, enemics, nivells, interfície i objectes); la carpeta scripts emmagatzema la lògica de programació en GDScript, separant el control global de la representació visual; la carpeta assets inclou tots els recursos multimèdia processats, com els sprites creats amb Aseprite i l'àudio optimitzat amb Audacity; finalment, la carpeta exports es reserva per als binaris compilats per a Windows i Linux.

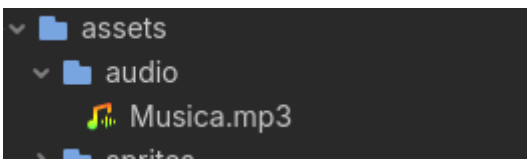
El sistema s'articula mitjançant diversos components principals que interactuen entre si per oferir una experiència de joc completa. El GameManager, implementat com un Singleton, actua com el cervell del joc gestionant l'estat global (vides, puntuació i nivell actual). El Jugador gestiona la recepció d'inputs, les físiques de moviment i les col·lisions, mentre que els Enemics es basen en una classe pare que defineix la IA i els patrons de moviment. Els Nivells funcionen com a contenidors que integren el tilemap i els objectes interactuables, i el HUD mostra la informació en temps real. Tot aquest ecosistema es recolza en un AudioManager que centralitza la reproducció de música i efectes sonors.

Finalment, la comunicació entre aquests components es realitza mitjançant un sistema de senyals basat en el patró de disseny Observer, que permet que les entitats estiguin desacoblades (per exemple, el jugador notifica un canvi i el HUD l'escolta per actualitzar-se). A més, s'han aplicat patrons com la Component-Based Architecture, el State Pattern per gestionar els estats del personatge (córrer, saltar, ferit) i el ja esmentat Singleton per a la gestió de dades globals, assegurant una estructura de programari robusta i professional.

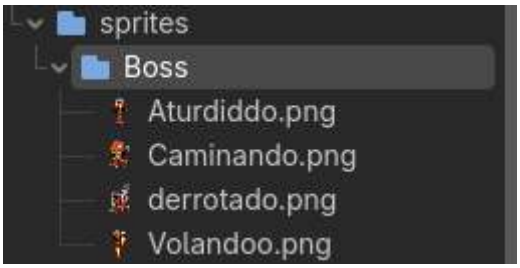
ESTRUCTURA JOC

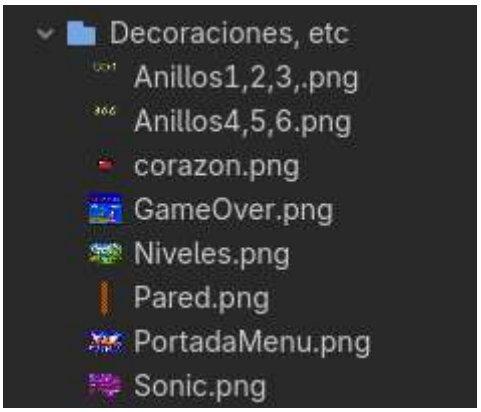


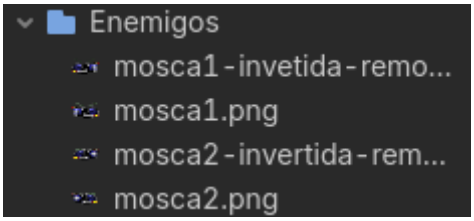
ASSETS



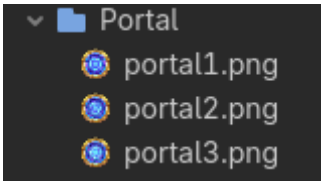
SPRITES

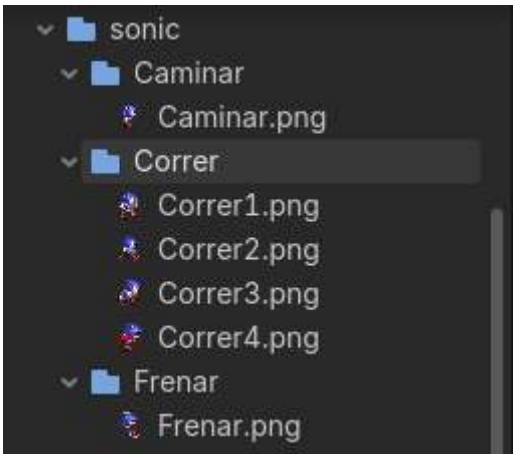

sprites
Boss
Aturdiddo.png
Caminando.png
derrotado.png
Volandoo.png

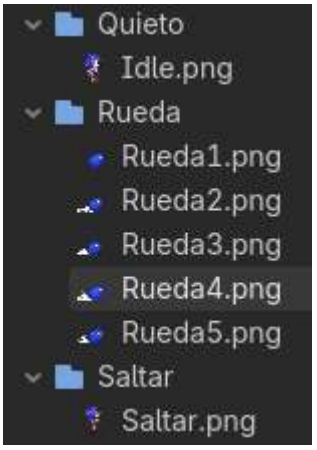

Decoraciones, etc
Anillos1,2,3,.png
Anillos4,5,6.png
corazon.png
GameOver.png
Niveles.png
Pared.png
PortadaMenu.png
Sonic.png

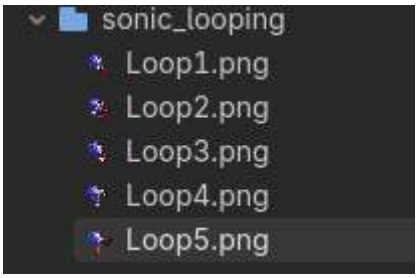

Enemigos
mosca1-invetida-remo...
mosca1.png
mosca2-invertida-rem...
mosca2.png


Niveles
Boss.png
Level1.png
Level2.png

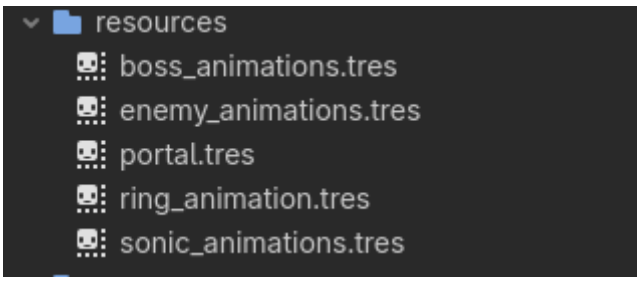

Portal
portal1.png
portal2.png
portal3.png



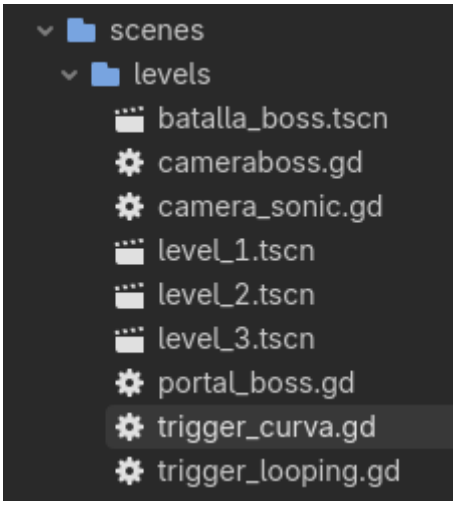


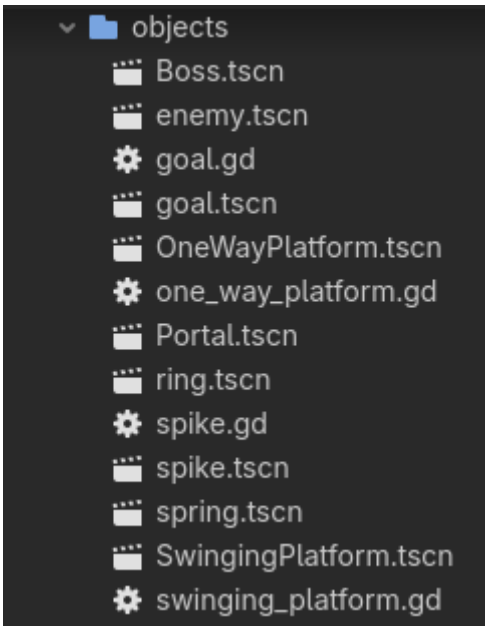


RESOURCES



SCENES




objects
Boss.tscn
enemy.tscn
goal.gd
goal.tscn
OneWayPlatform.tscn
one_way_platform.gd
Portal.tscn
ring.tscn
spike.gd
spike.tscn
spring.tscn
SwingingPlatform.tscn
swinging_platform.gd


player
sonic.tscn


ui
GameOver.tscn
hud.tscn
LevelSelect.tscn
level_complete.gd
level_complete.tscn
main_menu.tscn

SCRIPTS


scipts
boss.gd
enemy.gd
game_manager.gd
game_over.gd
hud.gd
level_select.gd
main_menu.gd
player.gd
ring.gd
spring.gd

2.4 Descripció dels components

En aquest apartat es detallen els elements modulars que formen l'arquitectura del videojoc. Cada component ha estat dissenyat sota el principi de responsabilitat única, assegurant que cada part del codi gestioni una funcionalitat específica del sistema per facilitar la depuració i el manteniment.

2.4.1 Component Player

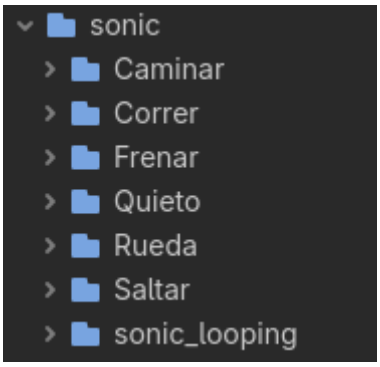
Aquest component és el més complex del projecte, ja que la seva funcionalitat principal és gestionar tot el comportament del personatge protagonista. El seu codi s'encarrega del processament en temps real dels inputs de l'usuari, l'aplicació de les fórmules físiques de moviment (acceleració, fricció i velocitat màxima) i la gestió de les animacions segons l'estat actual. A més, controla la detecció de col·lisions amb el terreny i la lògica de recollida d'anells.

Per a la seva construcció, s'han utilitzat els següents nodes de Godot:

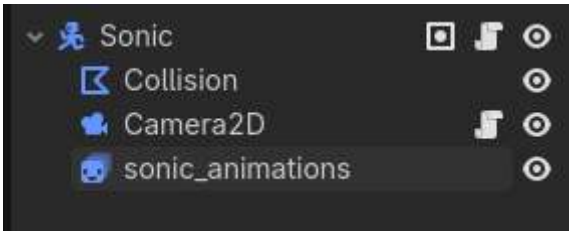
- CharacterBody2D (Sonic): El node base que proporciona l'API per al moviment cinemàtic i la detecció de col·lisions.
- AnimatedSprite2D (sonic_animation): Encarregat de reproduir els diferents frames d'animació sincronitzats amb la lògica del personatge.
- CollisionShape2D (collision): Defineix la forma física (generalment una càpsula o cercle) que interactuarà amb el món.
- AnimationPlayer: Serveix per fer el moviment de sonic més fluid.

El jugador es gestiona mitjançant una màquina d'estats que alterna entre: Repòs (Idle), Corrent (Running), Saltant (Jumping), Danyat (Hurt) i Càrrega de velocitat (Spin Dash).

ESTRUCTURA SONIC



NODES SONIC



ANIMACIONS SONIC

ACCELERAR

RODA

FRE

LOOPING

QUIET

SALT

2.4.2 Component Enemy

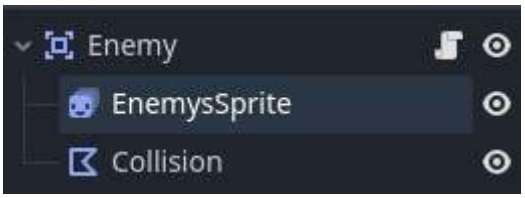
La funcionalitat d'aquest component és gestionar el comportament autònom dels enemics (Badniks). Inclou la lògica de moviment automàtic mitjançant patrons definits, la detecció de proximitat del jugador i la gestió de la seva pròpia destrucció quan reben un impacte.

S'han dissenyat 2 tipus d'enemics amb comportaments diferenciats:

- Buzz Bomber: Enemic volador que patrulla una zona aèria i ataca el jugador des d'una posició elevada.
- Eggman: Enemic robot que llança bombes i envesteix a sonic

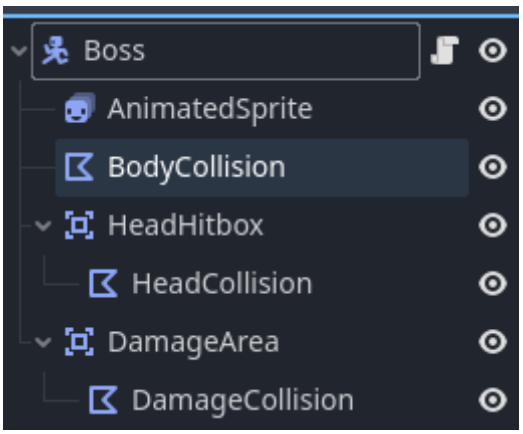
Els nodes clau per a aquest component són el CharacterBody2D per al moviment, RayCast2D per detectar el terra o parets abans de xocar, i Timer per gestionar les pauses entre atacs o canvis de direcció.

ESTRUCTURA ENEMICS



EGGMAN:

ESTRUCTURA EGGMAN



DERROTAT:

CAMINANT:

ATORDIT:

2.4.3 Component Level

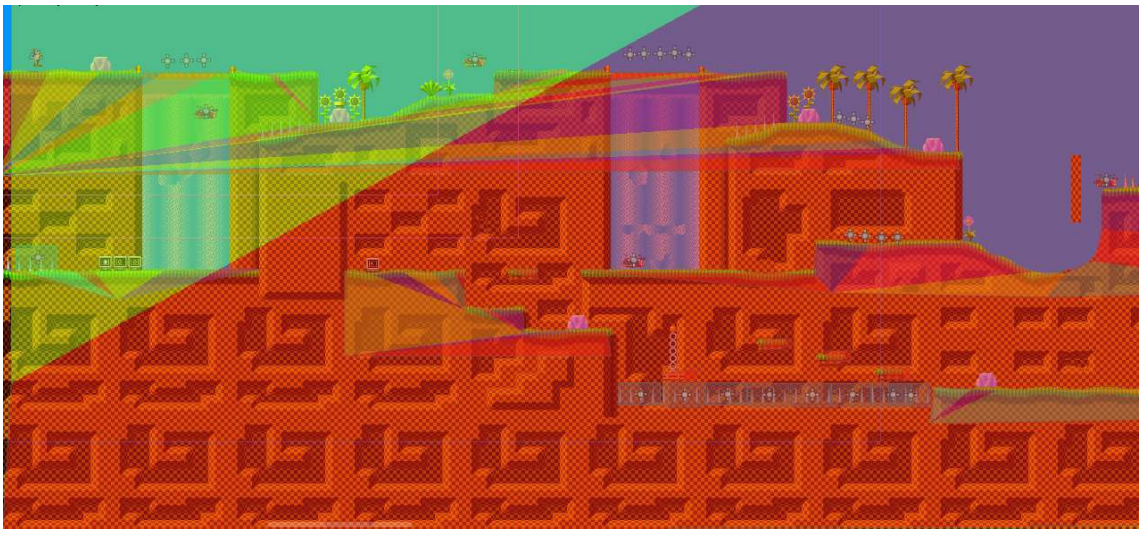
El component de nivell actua com l'organitzador de tot el contingut jugable. La seva funció és estructurar el mapa, posicionar els enemics i objectes interactuables, definir els límits del món i controlar el flux de la partida (punts de spawn i meta).

Per aconseguir una experiència visual rica, s'han organitzat les escenes en 5 capes diferenciades:

1. Background (Fons): Fons de color blau
2. Terreno (Terrain): Capa de col·lisió principal on es mouen les entitats.
3. Objectes: Capa interactuable que conté els anells i els checkpoints.
4. Enemigos: Ubicació inicial dels Badniks.
5. Elements decoratius: Detalls visuals que no tenen col·lisió però aporten ambientació.

Els nodes principals utilitzats són el TileMap per a la construcció del terreny, el ParallaxBackground per crear efecte de profunditat en el fons, el Marker2D per marcar coordenades de reaparició i la Camera2D per al seguiment dinàmic del jugador.

EDICIÓ NIVELL



2.4.4 Component GameManager

El GameManager és el gestor global i centralitzat de l'estat del joc, implementat com un Autoload (Singleton). La seva responsabilitat és mantenir la persistència de les dades durant tota l'execució, independentment de si el jugador canvia de nivell o reinicia una fase.

Les seves responsabilitats principals inclouen:

- Gestió de vides: Controlar el comptador de vides restants i decidir quan s'ha d'activar el reinici total del joc.
- Puntuació global: Acumular els punts obtinguts per anells i enemics destruïts.
- Flux de navegació: Orquestrar el canvi entre els diferents nivells i escenes.
- Estats terminals: Gestionar les condicions de victòria (arribar a la meta) o de derrota (Game Over), disparant les interfícies corresponents.

SCRIPT ANELLS

game_manager.gd

```
extends Node

signal ring_count_changed

var rings = 0
var lives = 3
var current_level = 1

func add_ring():
    rings += 1
    ring_count_changed.emit()

func lose_rings():
    rings = 0
    ring_count_changed.emit()

func reset_game():
    rings = 0
    lives = 3
    current_level = 1
    ring_count_changed.emit()

func next_level():
    current_level += 1
```

2.5 Definició de les tasques

2.5.1 Prova 1

Sistema de moviment del personatge amb velocitat progressiva, amb salt i spin dash com al videojoc original. Utilitzarem CharacterBody2D amb velocitat progressiva. El spindash es recarrega de manera automàtica.

2.5.2 Prova 2

Detecció de col·lisions amb terreny, rampes i enemics. Utilitzarem CollisionShape2D per al personatge, també un sistema de RaiCast2D per detectar pendents també tindrà col·lisions amb enemics mitjançant Area2D.

2.5.3 Prova 3

Sistema de col·lecció de anells per puntuació. Cada anell es un Area2D que detecta contacte amb el jugador, quan sonic rep dany els anells s'expulsen en múltiples direccions i després de 3 segons desapareixen

2.6 Definició de les funcionalitats

2.6.1 Funcionalitat 1- Moviment del jugador

Aquesta tasca se centra en la programació del moviment de l'avatar. A diferència d'un joc de plataformes estàndard, aquest sistema requereix una gestió d'acceleració progressiva per simular la inèrcia característica de la franquícia.

- Detalls tècnics: S'utilitzarà el node CharacterBody2D per gestionar el moviment. El codi aplicarà vectors de força horitzontal que augmentaran gradualment fins a arribar a un topall de velocitat màxima.

- Accions especials: S'inclourà el salt parabòlic i la mecànica de Spin Dash. Aquesta darrera es programarà com una càrrega d'energia cinètica que es pot activar en repòs, disparant el personatge a la màxima velocitat de forma instantània.

2.6.2 Funcionalitat 2- Col·lecció i puntuació

L'objectiu d'aquesta tasca és assegurar que el personatge interactuï correctament amb el disseny del nivell, especialment en zones complexes.

- Detecció de pendents: S'implementarà un sistema de RayCast2D situat a la base del personatge. Aquests rajos permeten detectar l'angle del terra en temps real, permetent que el personatge s'incline i s'adapti visualment a les rampes.

- Seguretat física: S'utilitzaran nodes CollisionShape2D per al cos principal i nodes Area2D per a la detecció d'enemics i trampes, separant clarament la col·lisió física de la col·lisió de dany per evitar conflictes en el processament de dades del motor.

2.6.3 Funcionalitat N- Enemics i vides

Aquesta funcionalitat gestiona la supervivència del jugador i el sistema de puntuació.

- Recollida: Cada anell serà una instància d'una Area2D que, en detectar el contacte amb el jugador, emetrà un senyal al GameManager per augmentar el comptador i reproduir un efecte sonor.

- Mecànica de dany: En el moment en què el jugador rep un impacte, el sistema executarà un algorisme de dispersió. Els anells acumulats s'expulsaran en múltiples direccions.

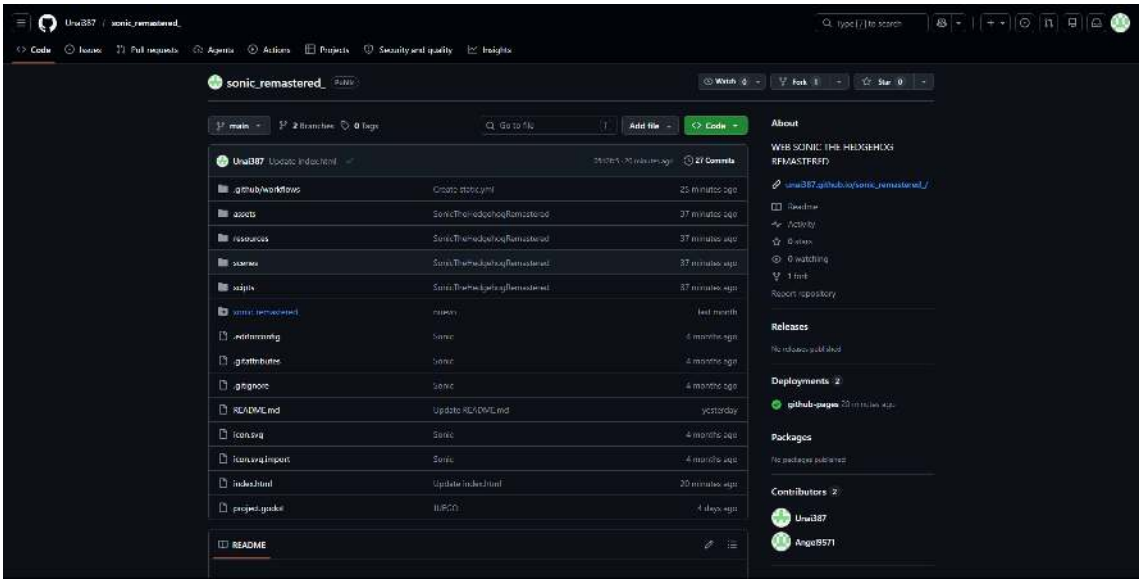
3 Altres capítols

3.1 Configuració de l'entorn de desenvolupament

La configuració de l'entorn de treball ha estat un pas previ fonamental. S'ha instal·lat Godot Engine 4.2 en ambdós equips de l'equip de desenvolupament (sistemes Linux Ubuntu 22.04 i Windows 11), assegurant que el comportament del motor és idèntic en les dues plataformes. El repositori de GitHub actua com a punt de sincronització, i s'han establert convencions de nomenclatura de branques i commits al document CONTRIBUTING.md del repositori.

S'han configurat les següents eines addicionals a l'entorn:

- Aseprite compilat des del codi font amb suport per a exportació de sprites en format PNG amb metadades JSON per a Godot.
- Audacity per a l'edició i normalització dels efectes sonors descarregats de fonts lliures de drets.
- Git amb configuració de LFS (Large File Storage) per als fitxers d'àudio i gràfics que superen els 50 MB.



3.2 Arquitectura del codi

L'arquitectura del codi segueix els patrons de disseny recomanats per al desenvolupament en Godot Engine:

- **Component-Based Architecture:** Cada entitat del joc (jugador, enemic, anell) és una escena independent amb el seu propi arbre de nodes i script associat. Això facilita la reutilització i les proves unitàries.
- **Singleton (Autoload):** El GameManager i l'AudioManager s'implementen com a Autoloads de Godot, disponibles globalment des de qualsevol escena sense necessitat de instanciar-los manualment.
- **Observer (Signals):** La comunicació entre components s'efectua exclusivament mitjançant el sistema de senyals de Godot, que implementa el patró Observer. Això evita l'acoblament directe entre classes i facilita les modificacions futures.

- **State Pattern:** El comportament del jugador es gestiona amb una màquina d'estats explícita, on cada estat (Idle, Running, Jumping, Hurt, SpinDash) té les seves pròpies funcions d'entrada, actualització i sortida.

3.3 Sistema de físiques i moviment

La implementació del moviment de Sonic ha estat un dels reptes tècnics principals del projecte. Hem analitzat documentació de la comunitat de fans ('Sonic Physics Guide') per obtenir els valors de referència:

Paràmetre	**Valor original (Mega Drive)**	**Valor implementat (Godot)**
Acceleració a terra	0,046875 px/frame²	350 px/s²
Fricció (frenada)	0,046875 px/frame²	350 px/s²
Velocitat màxima a terra	6 px/frame (360 px/s)	600 px/s
Gravetat	0,21875 px/frame²	1800 px/s²
Velocitat inicial de salt	6,5 px/frame	650 px/s (cap amunt)
Velocitat SpinDash màxima	12 px/frame	1200 px/s

Una de les dificultats principals ha estat adaptar la sensació de velocitat original de Sonic a les físiques modernes de Godot Engine 4. Els primers prototips mostraven problemes de control excessivament sensible i errors en les col·lisions amb rampes.

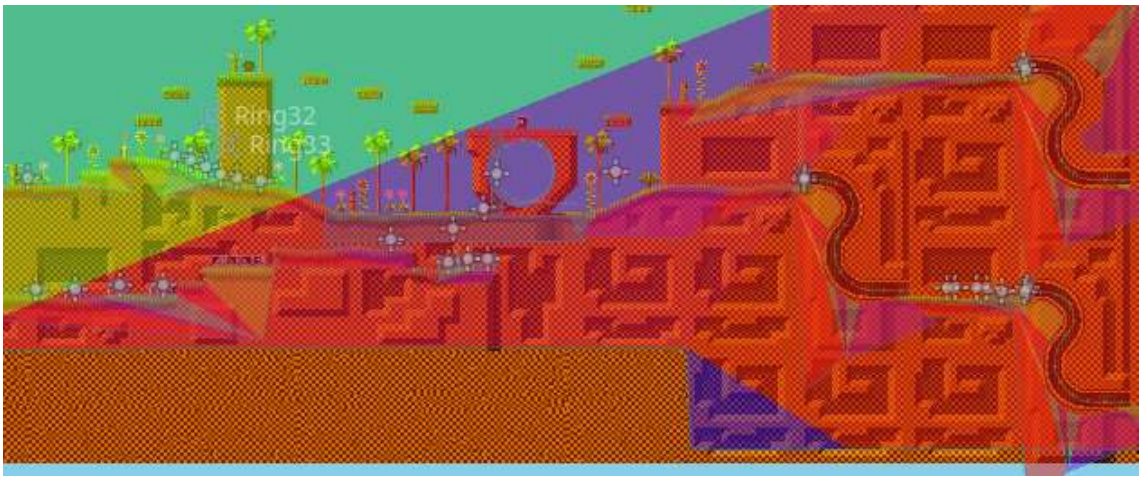
Per solucionar-ho, es van ajustar progressivament els valors d'acceleració i fricció, realitzant proves constants amb diferents configuracions. També es va implementar un sistema de detecció de pendents mitjançant RayCast2D per adaptar correctament el moviment del personatge a superfícies inclinades.

El sistema final aconsegueix una experiència fluida que manté l'essència del joc original, però adaptada a resolucions i velocitats modernes.

3.4 Disseny de nivells

Els nivells del joc s'han creat amb l'editor integrat de TileMap de Godot Engine 4. El procés de disseny de cada nivell ha seguit els passos següents:

1. Creació del TileSet amb els tiles gràfics corresponents al tema visual del nivell.
2. Construcció del mapa de col·lisions al TileMap, assegurant que cada tile té la forma de col·lisió correcta.
3. Col·locació d'enemics, anells i checkpoints com a nodes Fill de l'escena del nivell.
4. Configuració de les capes de parallax per als fons i la càmera de seguiment.
5. Play-testing intensiu per identificar zones de dificultat desequilibrada i ajustar el disseny.

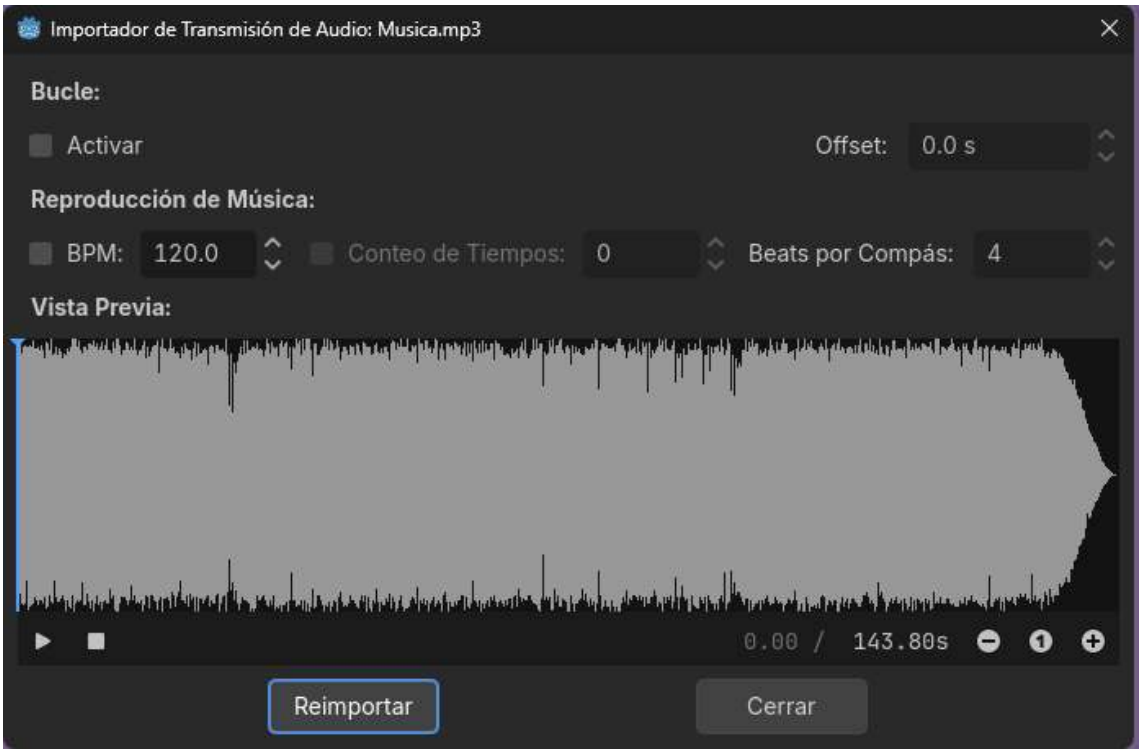


3.5 Sistema d'àudio

L'àudio del joc s'ha implementat amb el node AudioStreamPlaier de Godot, gestionat de forma centralitzada per l'AudioManager Singleton. Tots els efectes sonors s'han obtingut de repositoris de sons lliures de drets (freesound.org) i s'han editat amb Audacity per normalitzar el volum, ajustar la durada i aplicar efectes bàsics.

La música de fons de cada nivell s'implementa amb un AudioStreamPlaier que es carrega i reprodueix en loop quan s'inicia el nivell, i es detura (amb un fade out suau) en sortir-ne. S'ha implementat una transició suau entre les cançons del nivell principal i el tema de boss per no trencar la immersió.

MÚSICA NIVELLS:



3.6 Interfície d'usuari

La interfície d'usuari s'ha creat amb el sistema CanvasLaier de Godot, que permet superposar elements UI sobre la gameplay sense que la càmera de joc els afecti.

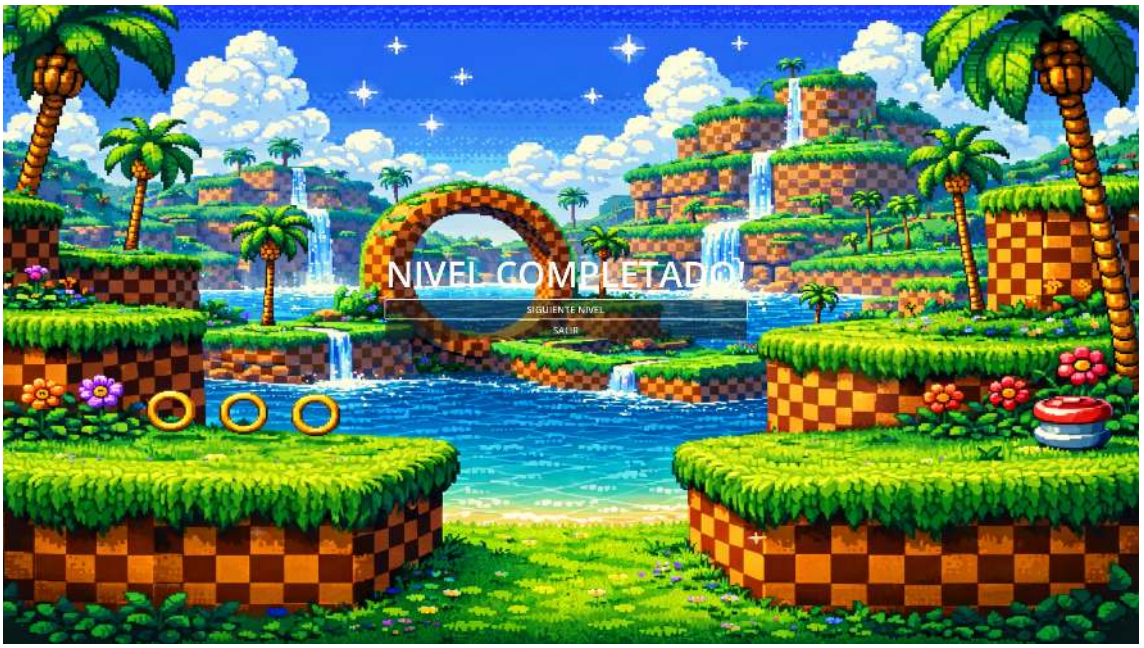
El HUD mostra en temps real: el comptador d'anells (actualitzat via senial del GameManager), la puntuació acumulada, el temps transcorregut i les vides restants representades per icones del personatge.

El menú principal utilitza un Control Node amb un fons animat (un bucle del primer nivell en moviment lent) i botons estilitzats que emulen l'estètica visual del joc original. S'han implementat transicions animades entre pantalles per millorar la percepció de qualitat del producte.

INTERFÍCIE JOC


Rings: 0
0:01

4 Conclusions



4.1 Conclusions generals del projecte

El projecte ha permès desenvolupar un videojoc funcional basat en Sonic utilitzant Godot Engine 4. S'han aplicat conceptes de programació, fisica 2D i disseny d'interferències, obtenir un prototip estable i jugable.

El treball en equip ha estat un element clau de l'èxit del projecte. La divisió de responsabilitats (un membre centrat en la programació de mecàniques i l'altre en el disseny de nivells i gràfics) ha permès avançar en paral·lel sense bloquejos significatius. La comunicació diària i les revisions de codi setmanals han contribuït a mantenir la qualitat del producte.

Des del punt de vista tècnic, Godot Engine 4 ha demostrat ser una eina excel·lent per als tipus de joc desenvolupat. El sistema de senyals, els nodes especialitzats per a plataformes i la integració nativa de TileMap han facilitat enormement la implementació de les funcionalitats requerides.

4.2 Consecució dels objectius

L'objectiu s'ha assolit, ja que s'ha creat un joc de plataforma en 2D amb moviment, enemics, anells, nivell i sistema de puntuació. Els objectius específics s'ha completat majoritàriament segons la planificació inicial.

4.3 Valoració de la metodologia i planificació

La metodologia per sprints ha permès avança de manera progressiva i organitzada. Tot i alguns ajustos tècnics, la planificació ha estat adequada per garantir el correcte desenvolupament del projecte.

El tauler de Kanban de GitHub Projects ha estat una eina molt útil per a la gestió visual del progrés. El fet que estigui integrat al repositori de codi ha facilitat la traçabilitat entre les tasques i els commits corresponents.

Com a punt de millora de la metodologia, hauria estat beneficiós establir sessions de play-testing externes des de les primeres fases del projecte, per identificar problemes de jugabilitat abans i amb menys cost de correcció.

4.4 Visió de futur

Com a millores futures es poden explorar les línies de treball següents:

- Nivells addicionals: Incorporar almenys tres zones temàtiques noves (zona d'aigua, zona de lava, zona final amb boss Eggman).
- Més enemics i varietat: Afegir nous Badniks amb comportaments més complexos, com enemics que requereixen múltiples impactes per ser eliminats.
- Compatibilitat amb comandament: Afegir suport natiu per a gamepads via el sistema d'Input Map de Godot.
- Sistema de desament de partida: Implementar un sistema de guardat persistent de la puntuació i el nivell assolit, amb suport per a múltiples perfils d'usuari.
- Exportació per a Windows i Web: Compilar el joc per a Windows i per al navegador (exportació HTML5) per facilitar la distribució i l'accés.

5. Glossari

- **Godot Engine:** Motor de videojocs de codi obert utilitzat per al desenvolupament del projecte. Permet crear videojocs 2D i 3D amb eines integrades de física, renderitzat i scripting.
- **GDScript:** Llenguatge de programació propi de Godot Engine. Té una sintaxi similar a Python i està optimitzat per al desenvolupament ràpid de videojocs.
- **HUD (Heads-Up Display):** Interfície gràfica mostrada durant la partida que proporciona informació en temps real al jugador, com ara anells, puntuació, temps o vides.
- **TileMap:** Sistema de construcció d'escenaris basat en mosaics o "tiles", utilitzat per crear nivells 2D de manera eficient.
- **FPS (Frames Per Second):** Nombre de fotogrames que el joc mostra cada segon. Una taxa estable de 60 FPS garanteix una experiència fluida.
- **Sprite:** Imatge 2D utilitzada per representar personatges, objectes o enemics dins del videojoc.
- **Sprite Sheet:** Fitxer que conté múltiples sprites agrupats en una sola imatge per facilitar les animacions.
- **CollisionShape2D:** Node de Godot que defineix la forma física utilitzada en la detecció de col·lisions.
- **CharacterBody2D:** Node especialitzat de Godot utilitzat per controlar personatges amb moviment programat i detecció de col·lisions.
- **RaiCast2D:** Sistema que proyecta un "raig" invisible per detectar superfícies, enemics o obstacles.
- **Area2D:** Node utilitzat per detectar interaccions entre objectes sense necessitat de col·lisió física completa.
- **Singleton:** Patró de disseny que permet tenir una única instància global d'un sistema, com el GameManager.
- **GameManager:** Sistema central encarregat de controlar l'estat general del joc, incloent puntuació, vides i canvi de nivells.
- **AudioManager:** Component responsable de gestionar la música i els efectes sonors del videojoc.
- **Parallax Background:** Tècnica visual que mou diferents capes del fons a velocitats diferents per crear sensació de profunditat.
- **Spin Dash:** Mecànica clàssica de Sonic que permet carregar velocitat abans de sortir impulsat.

- **Badnik:** Nom genèric dels enemics robòtics presents als videojocs de Sonic.
- **Checkpoint:** Punt de control que permet reprendre la partida des d'una posició intermèdia després de perdre una vida.
- **UI (User Interface):** Conjunt d'elements visuals amb què interactua l'usuari.
- **QA (Quality Assurance):** Procés de proves i verificació per detectar errors i assegurar la qualitat del producte final.
- **Kanban:** Sistema visual de gestió de tasques utilitzat per organitzar el flux de treball del projecte.
- **Git:** Sistema de control de versions utilitzat per registrar i gestionar els canvis del codi font.
- **GitHub:** Plataforma online utilitzada per allotjar el repositori del projecte i coordinar el treball en equip.
- **Open Source:** Programari de codi obert que permet modificar i distribuir lliurement el seu contingut.
- **Play-testing:** Procés de proves del videojoc amb jugadors per detectar problemes de jugabilitat, dificultat o rendiment.
- **MVP (Minimum Viable Product):** Primera versió funcional del projecte que inclou les característiques essencials.
- **Looping:** Reproducció contínua d'un arxiu d'àudio o animació sense interrupcions visibles.
- **Input Map:** Sistema de Godot que permet assignar tecles o botons a accions del joc.
- **Renderitzat 2D:** Procés gràfic mitjançant el qual el motor dibuixa els elements visuals en pantalla.

6. Bibliografia

Webs i documentació tècnica

Godot Engine Documentaico: Godot Docs

Godot Engine Official Website: Godot Engine

Sprites Sonic: Sonic the Hedgehog 2 - Sega Genesis - The Spriters Resource

Mapas: Sonic the Hedgehog/Maps

GDQuest: Tutorials · GDQuest

Sprites Enemigos: Badniks - Sonic the Hedgehog - Sega Genesis - The Spriters Resource

Sprites Eggman: Death Egg Robot - Sonic the Hedgehog 2 - Sega Genesis - The Spriters Resource

Audacity: Audacity

7 Annexos

Aquí posarem els scripts del personatges perquè són bastant llargs.

SCRIPT SONIC

```gdscript
extends CharacterBody2D

const ACCELERATION = 1200.0
const DECELERATION = 800.0
const MAX_SPEED = 400.0
const FRICTION = 600.0
const JUMP_VELOCITY = -500.0
const GRAVITY = 1200.0
const MAX_FALL_SPEED = 800.0
const WALL_JUMP_VELOCITY = Vector2(400, -500)
const WALL_SLIDE_SPEED = 100.0
const COYOTE_DURATION = 0.15

@onready var RingScene = preload("res://scenes/objects/ring.tscn")
@onready var anim = $sonic_animations

@export var vidas: int = 3
var coyote_timer = 0.0
var esta_invulnerable = false
var speed = 0.0
var rings = 0
var is_on_wall_slide = false

func _ready():
    if GameManager: GameManager.lives = vidas

func _physics_process(delta):
    coyote_timer = COYOTE_DURATION if is_on_floor() else coyote_timer - delta

    if not is_on_floor():
        velocity.y = min(velocity.y + GRAVITY * delta, MAX_FALL_SPEED)

    is_on_wall_slide = is_on_wall() and not is_on_floor()
    if is_on_wall_slide: velocity.y = min(velocity.y, WALL_SLIDE_SPEED)

    var floor_normal = get_floor_normal()
    var is_on_slope = is_on_floor() and abs(floor_normal.x) > 0.2
    var input_dir = Input.get_axis("ui_left", "ui_right")

    if input_dir != 0:
        speed = clamp(speed + input_dir * ACCELERATION * delta, -MAX_SPEED, MAX_SPEED)
        anim.scale.x = 1 if input_dir > 0 else -1
    elif is_on_slope:
        speed += floor_normal.x * 800.0 * delta
    else:
        speed = move_toward(speed, 0, FRICTION * delta)

    velocity.x = speed

    if Input.is_action_just_pressed("ui_accept"):
        if is_on_wall_slide:
            velocity = Vector2(get_wall_normal().x * WALL_JUMP_VELOCITY.x, WALL_JUMP_VELOCITY.y)
            speed = velocity.x
        elif coyote_timer > 0:
            velocity.y = JUMP_VELOCITY
            coyote_timer = 0
```

```
57
58      move_and_slide()
59      update_animation()
60
61  func update_animation():
62      if is_on_wall_slide or not is_on_floor():
63          anim.play("saltar")
64      elif abs(velocity.x) > 10:
65          if anim.animation not in ["correr_loop", "acelerar"]: anim.play("acelerar")
66          elif anim.animation == "acelerar" and not anim.is_playing(): anim.play("correr_loop")
67      else:
68          anim.play("quieto")
69
70  func collect_ring():
71      rings += 1
72      _update_hud_rings()
73      if GameManager: GameManager.add_ring()
74
```

```
75  func soltar_anillos_al_aire():
76      for i in range(min(rings, 15)):
77          var r = RingScene.instantiate()
78          get_parent().call_deferred("add_child", r)
79          r.global_position = global_position
80          if "soltado" in r:
81              r.soltado = true
82              var ang = randf_range(0, TAU)
83              r.velocidad = Vector2(cos(ang), sin(ang)) * randf_range(200, 450)
84
85  func _update_hud_rings():
86      var hud = get_tree().current_scene.find_child("Hud", true, false)
87      if hud: hud.actualizar_interfaz_anillos(rings)
88
89  func recibir_dano():
90      if esta_invulnerable: return
91      vidas -= 1
92
```

```
93      var hud = get_tree().current_scene.find_child("Hud", true, false)
94      if hud: hud.actualizar_interfaz_vidas(vidas)
95      if GameManager: GameManager.lives = vidas
96
97      if vidas <= 0: die()
98      else: aplicar_efecto_dano()
99
100     if has_node("Camera2D"): $Camera2D.apply_shake(5.0)
101
102 func hit():
103     if esta_invulnerable: return
104     if rings > 0:
105         soltar_anillos_al_aire()
106         rings = 0
107         _update_hud_rings()
108         if GameManager: GameManager.lose_rings()
109         aplicar_efecto_dano()
110     else: recibir_dano()
```

```
111
112 func aplicar_efecto_dano():
113     esta_invulnerable = true
114     velocity = Vector2(-300 if anim.scale.x > 0 else 300, -250)
115     var tw = create_tween().set_loops(10)
116     tw.tween_property(anim, "modulate:a", 0.2, 0.1)
117     tw.tween_property(anim, "modulate:a", 1.0, 0.1)
118     await get_tree().create_timer(2.0).timeout
119     esta_invulnerable = false
120
121 func die():
122     var path = "res://scenes/ui/GameOver.tscn"
123     get_tree().change_scene_to_file(path) if ResourceLoader.exists(path) else get_tree().reload_current_scene()
```

SCRIPT BUZZ BOMBER

```gdscript
extends Area2D

@export var speed = 150.0
@export var move_distance = 300.0

var direction = 1
var start_pos = Vector2.ZERO

func _ready():
    start_pos = position
    body_entered.connect(_on_body_entered)

func _process(delta):
    position.x += speed * direction * delta
    if abs(position.x - start_pos.x) > move_distance:
        direction *= -1
        $EnemysSprite.scale.x *= -1

func _on_body_entered(body):
    if body.name == "Sonic":
        if body.velocity.y > 0 and body.position.y < position.y - 10:
            body.velocity.y = -400
            queue_free()
        else:
            body.hit()
```

SCRIPT BOSS (EGGMAN)

```
extends CharacterBody2D

@export var health = 5
@export var move_speed = 130.0
@export var attack_speed = 450.0
@export var follow_range = 600.0

# --- ESCENA DE LA BOMBA ---
@export var escena_bomba: PackedScene

var player = null
var is_stunned = false
var is_defeated = false
var is_attacking = false
var furia_factor = 1.0

# --- CONTROLADOR DEL TIEMPO DE BOMBAS (1 SEGUNDO) ---
var b_timer = 0.0

@onready var sprite = $AnimatedSprite

func _ready():
    process_mode = Node.PROCESS_MODE_ALWAYS
    set_physics_process(true)

    player = get_tree().current_scene.find_child("Sonic", true, false)
    $HeadHitbox.body_entered.connect(_on_head_hit)
    $DamageArea.body_entered.connect(_on_damage_player)

func _physics_process(delta):
    if not is_on_floor():
        velocity.y += 980 * delta

    if is_defeated or is_stunned:
        velocity.x = move_toward(velocity.x, 0, 10)
    else:
        # Controlamos el disparo de bombas mientras se mueve o ataca
```

```
37          # Controlamos el disparo de bombas mientras se mueve o ataca
38          manejar_disparo_bombas(delta)
39
40          if not is_attacking:
41              ia_persecucion()
42
43      move_and_slide()
44
45  func ia_persecucion():
46      if player:
47          var dist = player.global_position.x - global_position.x
48          if abs(dist) < follow_range:
49              sprite.flip_h = dist > 0
50              if abs(dist) < 180:
51                  atacar()
52              else:
53                  velocity.x = sign(dist) * (move_speed * furia_factor)
54                  sprite.play("idle")
55          else:
56              velocity.x = 0
57
58  func atacar():
59      if is_attacking: return
60      is_attacking = true
61      velocity.x = 0
62      await get_tree().create_timer(0.4 / furia_factor).timeout
63
64      if not is_defeated and not is_stunned:
65          var dir = 1 if sprite.flip_h else -1
66          velocity.x = dir * (attack_speed * furia_factor)
67          await get_tree().create_timer(0.8).timeout
68      is_attacking = false
69
70  # --- NUEVA LÓGICA: DISPARAR MIENTRAS SE MUEVE HACIA SONIC ---
71  func manejar_disparo_bombas(delta):
72      # Si Eggman está quieto por el golpe (stunned) o derrotado (defeated), NO lanza bombas
73      if is_stunned or is_defeated or player == null:
```

```
74          return
75
76      # Si no ha recibido el primer golpe (furia_factor es 1.0), tampoco lanza bombas aún
77      if furia_factor <= 1.0:
78          return
79
80      b_timer += delta
81      # Espera exactamente 4.0 segundos entre cada lanzamiento
82      if b_timer >= 2.5:
83          b_timer = 0.0 # REINICIAMOS el contador inmediatamente para evitar ráfagas locas
84          lanzar_bomba_teledirigida()
85
86  func lanzar_bomba_teledirigida():
87      if escena_bomba == null or player == null: return
88
89      var bomba = escena_bomba.instantiate()
90      # Aparece justo en el centro de Eggman
91      bomba.global_position = global_position + Vector2(0, 10)
92      get_tree().current_scene.add_child(bomba)
93
94      # Si tu bomba tiene la función "configurar_direccion", calcula el tiro hacia Sonic
95      if bomba.has_method("configurar_direccion"):
96          bomba.configurar_direccion(player.global_position)
97
98  func _on_head_hit(body):
99      if body.name == "Sonic" and not is_stunned and not is_defeated:
100         if body.velocity.y > 0:
101             take_damage()
102             body.velocity.y = -400
103
104 func take_damage():
105     health -= 1
106     furia_factor += 0.25
107     is_stunned = true
108     sprite.play("stunned")
109     await get_tree().create_timer(0.8).timeout
```

```
110         is_stunned = false
111         if health <= 0: defeat()
112
113 func defeat():
114     is_defeated = true
115     sprite.play("defeated")
116     await get_tree().create_timer(4.0).timeout
117     $BodyCollision.set_deferred("disabled", true)
118     queue_free()
119
120 func _on_damage_player(body):
121     if body.name == "Sonic" and not is_defeated and not is_stunned:
122         if body.has_method("recibir_dano"): body.recibir_dano()
```