Institut Puig Castellar
Santa Coloma de Gramenet

Collect Carrot
Projecte de desenvolupament
CFGM Sistemes Microinformàtics i Xarxes

Autors: Mario Alcaraz i Jon Bornás
Grup: 2SMXB
Curs acadèmic

A) Creative Commons:

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Resum del projecte (màxim 250 paraules):

Aquest treball recull la programació de Collect Carrot, un videojoc de plataformes en dues dimensions desenvolupat amb el motor Godot Engine. Per jugar cal controlar un personatge (un conill) al llarg de 10 nivells on el bucle se centra a recollir ítems i esquivar enemics. El mapejat s'ha dividit en dos entorns diferents. El bosc exterior acull els primers set mapes de la partida. Després, entre els nivells 8 i 10, l'acció es desplaça a coves subterrànies. En aquest tram s'aplica una reducció de la il·luminació general. La corba de dificultat augmenta de forma mecànica.

El HUD gestiona el sistema de salut de manera reactiva. El motor commuta els gràfics dels cors al moment segons el dany de la instància del jugador. Pel que fa als inputs de control, prémer la tecla ESC obre un menú de pausa funcional. A l'últim mapa, la meta activa directament la pantalla de victòria. Scripts en GDScript i nodes vinculats per senyals natius fonamenten l'arquitectura del codi font. S'allibera així l'ús de memòria RAM. S'evita duplicar funcions dins de l'arbre d'execució. El rendiment és estable; el rendiment gràfic es clava en 60 FPS estables amb resolució de col·lisions a temps real.

Paraules clau (entre 4 i 8):

Collect Carrot

Videojoc de plataformes

Godot Engine

GDScript

Desenvolupament 2D

Arquitectura de nodes

Disseny de nivells

GitHub

Abstract (in English, 250 words or less):

This work covers the programming of Collect Carrot, a two-dimensional platform videogame developed with the Godot Engine. To play, it is necessary to control a character (a rabbit) throughout 10 levels where the loop focuses on collecting items and dodging enemies. The mapping has been divided into two different environments. The exterior forest hosts the first seven maps of the game. Afterwards, between levels 8 and 10, the action moves to underground caves. In this section, a reduction of the general illumination is applied. The difficulty curve increases mechanically.

The HUD manages the health system reactively. The engine switches the heart graphics instantly according to the damage of the player instance. Regarding the control inputs, pressing the ESC key opens a functional pause menu. In the last map, the goal directly activates the victory screen. Scripts in GDScript and nodes linked by native signals fundate the architecture of the source code. This frees up the use of RAM. Duplicating functions within the execution tree is avoided. The performance is stable; the graphic performance locks at 60 stable FPS with real-time collision resolution.

Keywords (entre 4 i 8):

Collect Carrot

Platformer

Godot Engine

GDScript

2D Development

Node-based architecture

Level design

GitHub

Índex

1 Introducció 8
1.1 Context 9
1.2 Justificació 10
1.3 Objectius 10
1.4 Estratègia i planificació del projecte 11
1.5 Metodologia de treball 11
1.6 Estudi econòmic i pressupostari 11
2 Descripció del projecte 12
2.1 Anàlisi de requisits 12
2.2 Previsió de tasques d'investigació 14
2.3 Tecnologies 15
2.4 Estructura del projecte 16
2.5 Descripció dels components 17
2.6 Definició de les tasques 19
2.7 Definició de les funcionalitats 20
3 Altres capítols 23
3.1 Anatomia i estructura del jugador (Player) 23
3.2 Arquitectura i lògica dels enemics (Guineu i Goblin) 23
3.3 Disseny de nivells i escenaris (de l'1 al 10) 24
3.4 Interfície d'usuari i HUD 24
3.5 Sistema de meta i canvi de nivell 25
3.6 Gestió de menús i flux de joc 25
4 Conclusions 25
4.1 Conclusions generals del projecte 25
4.2 Consecució dels objectius 26
4.3 Valoració de la metodologia i planificació 26
4.4 Visió de futur 27
5. Glossari 28
6. Bibliografia 43
7 Annexos 44
7.1. Implementació del Sistema de Moviment i Física del Jugador 44
7.2. Arquitectura de Dades i Persistència amb el Script Global 45
7.3. Intel·ligència Artificial i Patrulla dels Enemics 47
7.4. Mecàniques de Recollida i Gestió de Memòria dels Objectes 48
7.5. Programació de la Interfície i Comunicació amb el HUD 49
7.6. Flux de Transicions als nivells, Menús de Pausa, Victòria i Escenes de Cova 50

Llista de Figures

Figura 1: Interfície de Godot i animació de mort del personatge a SpriteFrames.

Figura 2: Configuració de l'animació d'atac del personatge a Godot

Figura 3: Animació en estat de repòs del personatge a SpriteFrames

Figura 4: Configuració i fotogrames de l'objecte pastanaga a Godot

Figura 5: Lògica de moviment i gravetat a la funció _physics_process

Figura 6: Vista de l'escenari de joc i nodes de col·lisió a Godot

Figura 7: Disseny de la interfície d'usuari (HUD) del videojoc

Figura 8: Propietats de col·lisió del node CharacterBody2D del jugador

Figura 9: Ajust de l'àrea de contacte amb CollisionShape2D

Figura 10: Configuració del contenedor de vides (HUD) a Godot

Figura 11: Propietats i paràmetres del node Camera2D

Figura 12: Script Zanahoria.gd i detecció de col·lisió de l'objecte

Figura 13: Configuració de RayCast2D a l'escena de l'enemic

Figura 14: Connexió del senyal body_exited de AreaVista

Figura 15: Funció per gestionar la sortida del jugador de l'àrea de visió

Figura 16: Disseny i configuració del nivell amb TileMapLayer

Figura 17: Propietats de monitorització i col·lisió de l'Area2D

Figura 18: Configuració de l'àudio de fons del nivell a Godot

Figura 19: Definició de constants de moviment a l'script del jugador

Figura 20: Control de gravetat i aturada per mort a _physics_process

Figura 21: Lògica de l'acció de salt i activació del so a l'script

Figura 22: Configuració de l'script d'Autoload a les variables globals

Figura 23: Lògica de variables globals i funció per reiniciar partida

Figura 24: Lògica d'activació de l'animació d'atac a l'script

Figura 25: Nodes i àrea de col·lisió de l'escena Goblin a Godot

Figura 26: Lògica de recol·lecció i eliminació de la pastanaga a l'script

Figura 27: Estructura de nodes de l'escena de la pastanaga

Figura 28: Editor de la interfície d'usuari i comptador al HUD

Figura 29: Estructura del contenidor de vides a l'escena HUD

Figura 30: Lògica d'actualització dinàmica de les vides a l'script

Figura 31: Llistat de fitxers de configuració dels 10 nivells

Figura 32: Captura del disseny de l'entorn exterior (herba)

Figura 33: Captura de l'entorn interior del nivell de la cova

Figura 34: Interfície gràfica de la pantalla de Game Over

Figura 35: Menú de Pausa del videojoc a l'editor

Figura 36: Pantalla final de victòria del joc

1 Introducció

Vam obrir aquest projecte amb una fita tancada: picar des de zero un plataformes 2D mitjançant Godot Engine. Buscàvem un producte executable complet i jugable sense complicacions. L'usuari controla directament un conill per saltar nivells, recollir pastanagues i esquivar enemics durant la ruta. Hem gestionat el cicle sencer del programari, des del codi pur de les mecàniques fins al disseny individual dels mapes.

L'aprenentatge purament pràctic ha guiat cada línia de codi. Ens calia comprovar de primera mà la distribució interna d'un projecte real en Godot, entenent la connexió entre escenes, l'arbre de scripts en GDScript i l'estalvi de memòria del sistema. Vam invertir moltes hores en característiques indispensables (moviment fluid del conill, controls per teclat, ajust de caixes de col·lisions i gestió del sistema de vides).

Crear 10 nivells operatius va ser el repte per entendre el disseny d'espais jugables i aplicar una corba de dificultat progressiva. Pel que fa a la interfície, vam programar un menú principal amb rutes de joc i sortida, l'apartat de controls, un menú de pausa funcional fet amb taulons de fusta i la pantalla de victòria terminal.

El resultat d'aquesta feina dona una perspectiva directa de com funciona una producció de programari real. El valor no és només extreure un executable estable i sense errors de codi; l'ús rutinari de GDScript consolida les bases mecàniques necessàries per construir, optimitzar i tancar aplicacions interactives.

1.1 Context

El panorama del desenvolupament independent (indie) experimenta una transformació notable gràcies a l'accessibilitat d'eines de programari actuals. Motors gràfics com Godot, Unity o Unreal Engine eliminen la necessitat de grans infraestructures corporatives per executar projectes funcionals. L'elecció de Godot Engine per a aquest treball respon a criteris tècnics específics: naturalesa de codi obert (open source), baix consum de recursos del sistema i una corba d'aprenentatge òptima per a entorns acadèmics.

Dins de la indústria, el gènere dels videojocs de plataformes manté una vigència sòlida a causa de l'acoblament de lògiques comprensibles amb un ventall ampli de disseny. El control de vectors (saltar, córrer), la gestió d'inventari bàsic i la detecció de col·lisions conformen una base idònia per a la instrucció en programació. Aquesta simplicitat mecànica justifica que gran part de la documentació tècnica i guies de desenvolupament utilitzin aquest gènere com a model de referència.

Un altre factor clau és l'expansió de la comunitat d'usuaris associada al motor. El volum de repositoris de codi obert, fòrums d'arquitectura de programari i resolució de fallades (debugging) agilitza el procés d'aprenentatge autònom. Disposar d'aquest suport permet dissenyar prototips funcionals i resoldre conflictes lògics en l'arbre de nodes sense estancar el cronograma del projecte.

L'execució d'aquest treball neix de l'oportunitat d'integrar un motor potent en un flux de treball assequible. El gènere de plataformes actua com a plataforma experimental per analitzar l'arquitectura de programari interna d'una aplicació interactiva. El desplegament del projecte troba la seva justificació en la utilització de tecnologies de producció vigents en el mercat actual.

1.2 Justificació

Els motius d'aquest desenvolupament són d'índole pràctica i d'autoaprenentatge. El projecte permet analitzar a temps real la programació d'aplicacions interactives amb Godot Engine, incidint directament en les lògiques tradicionals d'un plataformes. Així mateix, l'execució del programari fomenta l'adquisició d'habilitats en planificació tècnica, resolució de conflictes en el codi font (debugging) i optimització d'estructures modulars.

1.3 Objectius

1.3.1 Objectiu general

El propòsit central d'aquest treball consisteix a implementar un videojoc de plataformes funcional mitjançant el motor Godot Engine. Es busca consolidar competències en programació gràfica, disseny d'entorns interactius i gestió arquitectònica de programari des de la base.

1.3.2 Objectius específics

1. **Mecàniques físiques:** Desenvolupar el sistema de desplaçament, vectors de salt i caixes de col·lisió de la instància principal (el conill).
2. **Intel·ligència artificial simple:** Implementar rutines de patrulla i vectors de moviment en enemics actius per afegir dificultat mecànica.
3. **Disseny d'escenaris:** Estructurar el mapejat distribuint objectes col·leccionables (pastanagues) al llarg de la partida.
4. **Gestió d'estats:** Codificar un subsistema de salut reactiu a temps real connectat a les col·lisions del jugador.
5. **Interfície d'usuari (UI):** Dissenyar un entorn visual neta (HUD) que actualitzi els indicadors de vida, comptadors d'ítems i control de menús.
6. **Arquitectura de codi:** Establir una jerarquia de carpetes i nodes modular dins de l'entorn de treball per garantir la mantenibilitat.
7. **Optimització del sistema:** Executar protocols de prova de rendiment (framerates) per corregir errors de programari i calibrar l'equilibri de joc.

1.4 Estratègia i planificació del projecte

La fase de planificació va avaluar dues alternatives de desenvolupament diferenciades. La primera opció consistia a utilitzar un motor de plantilles preconfigurat dins de Godot Engine per modificar aspectes gràfics superficials i afegir paràmetres menors. La segona via plantejava l'escriptura íntegra del programari des de zero, assumint el disseny complet de les mecàniques i la creació de l'entorn.

Es va seleccionar el desenvolupament autònom des de zero. Aquesta ruta garantia la comprensió real del cicle de vida d'un programari interactiu i l'explotació de l'arquitectura del motor seleccionat. Tot i l'increment de càrrega laboral en la programació de lògiques i l'estructuració d'actius (assets), la viabilitat de l'execució es mantenia plenament alineada amb el calendari establert.

1.5 Metodologia de treball

L'organització del projecte s'ha basat en el marc de treball àgil Scrum. Aquest sistema metodològic va facilitar l'execució de proves de programari contínues, la reconfiguració immediata de mòduls ineficients i la priorització setmanal de les tasques del codi font.

La gestió de configuració de programari es va resoldre mitjançant un repositori allotjat a GitHub per evitar conflictes de versions. De manera paral·lela, el seguiment temporal es va monitoritzar amb un diagrama de Gantt integrat a Google Drive. Aquesta eina visual controlava el compliment de les fites i els terminis de lliurament fixats, minimitzant els colls d'ampolla en la producció de l'executable.

1.6 Estudi econòmic i pressupostari

La càrrega de treball del projecte es va segmentar en paquets de treball específics: programació d'entitats (jugador i enemics), maquetació de 10 nivells, distribució d'objectes col·leccionables, codificació d'interfícies (menús de pausa i victòria) i redacció de la documentació tècnica. Els costos de material es quantifiquen en zero unitats monetàries. El desenvolupament va requerir exclusivament maquinari preexistent (dos ordinadors portàtils) i entorns de programari gratuïts o de codi obert (Godot Engine, GitHub i eines de Google).

En conseqüència, el pressupost del projecte té un caràcter purament nominal. Com que els recursos gràfics i auditius utilitzats compten amb llicències lliures de drets, el valor de la inversió se centra en les hores d'enginyeria de programari dedicades. El manteniment futur es limita a la correcció de possibles errors de codi (bug fixing) detectats després del lliurament. El retorn de la inversió no busca rendibilitat financera, sinó l'adquisició de coneixement tècnic en sistemes interactius.

2 Descripció del projecte

2.1 Anàlisi de requisits

La fase prèvia a la codificació va requerir la definició exhaustiva de les tasques operatives del programari i les restriccions del sistema de computació. Aquesta especificació es divideix entre les lògiques directes de l'execució (requisits funcionals) i els estàndards de qualitat i rendiment de l'aplicació (requisits no funcionals).

2.1.1 Requisits funcionals

Codi	Requisit Funcional	Descripció Tècnica
RF1	Accés lliure	Execució directa des del menú principal sense necessitat de registre d'usuaris o base de dades.
RF2	Control d'instància	Gestió de vectors de moviment horitzontal i impuls vertical (salt) mitjançant mapeig de teclat.
RF3	Estat de salut	Renderització a temps real de l'indicador gràfic de vides estructurat per cors reals.
RF4	Comptador d'actius	Registre i acumulació dinàmica d'ítems recollits (pastanagues) en la sessió de joc.
RF5	Rutina de col·lisió	Detecció d'intersecció entre hitboxes (enemics o trampes) amb subrutina de descompte de salut.
RF6	Condició de terminal	Disparador de fi de partida per pèrdua total de recursos o per superació del darrer mapa (nivell 10).
RF7	Feedback visual	Canvis en els fulls de sprites (spritesheets) en rebre impactes o capturar col·leccionables.

RF8	Control de flux	Menú de pausa interactiu (interfície de fusta) que congela l'arbre de nodes a temps real.
RF9	Volatilitat de dades	Reinici complet de les variables del sistema en tancar i obrir la finestra de l'executable.

2.1.2 Requisits no funcionals

Codi	**Requisit No Funcional**	**Descripció Tècnica**
RNF1	Portabilitat	Compatibilitat multiplataforma en sistemes d'escriptori sense requeriments de targeta gràfica dedicada.
RNF2	Temps de resposta	Latència d'arrencada inferior a 10 segons des de la crida de l'executable fins al menú inicial.
RNF3	Llegibilitat	Resolució d'interfície optimitzada per a la visualització clara de fonts de lletra i menús.
RNF4	Taxa de refresc	Rendiment d'execució bloquejat a 60 FPS estables per evitar caigudes de frames (stuttering).
RNF5	Arquitectura modular	Segregació del codi font en escenes independents i components reutilitzables dins de Godot.

RNF6	Traçabilitat	Documentació interna de l'escriptura en GDScript i control de versions actiu allotjat a GitHub.
RNF7	Robustesa	Absència de fallades crítiques de memòria (memory leaks) que provoquin el tancament forçat del programari.

2.2 Previsió de tasques d'investigació

La corba d'aprenentatge inicial va requerir l'establiment d'un catàleg de línies de recerca tècnica abans d'executar el codi font. L'adquisició d'aquests coneixements va determinar l'arquitectura final de l'executable. Les fites d'investigació es van definir sota els següents paràmetres:

- **TI1 (Arquitectura de l'entorn):** Anàlisi del sistema de jerarquies basat en l'arbre de nodes i la modulació d'escenes independents dins de Godot Engine.
- **TI2 (Dinàmiques físiques):** Estudi del comportament dels cossos cinemàtics en entorns bidimensionals i configuració de màscares de col·lisió (node CollisionShape2D).
- **TI3 (Optimització d'inputs):** Desenvolupament d'algorismes de desplaçament reactius en l'script de control de l'entitat principal (jugador).
- **TI4 (Control de flux):** Disseny de connexions lògiques per a la transició automatitzada de mapes en interceptar els llindars terminals de cada nivell.
- **TI5 (Sistemes d'animació):** Implementació del node AnimationPlayer per coordinar els estats visuals de l'asset (reposar, córrer, saltar i rebre danys).
- **TI6 (Rendiment gràfic):** Recerca de pràctiques de refactorització de codi per garantir l'estabilitat del refresc de pantalla (bloqueig a 60 FPS).

2.3 Tecnologies

2.3.1 Comparativa de les tecnologies valorades

La selecció del motor gràfic va requerir una avaluació tècnica prèvia de les opcions més viables del mercat, buscant una eina optimitzada per a entorns interactius en dues dimensions (2D) i compatible amb els terminis d'un marc acadèmic. La primera opció analitzada va ser Godot Engine, que va destacar de forma immediata per la seva naturalesa de codi obert (open source) i el seu consum pràcticament nul de recursos del sistema. Aquest motor ofereix un disseny natiu excel·lent per a lògiques bidimensionals, fet que agilitza la creació de mapes i la gestió de col·lisions. L'únic desavantatge real d'aquesta eina és la seva menor penetració a la indústria de grans produccions comercials (jocs AAA), un factor que no afectava en absolut l'abast d'aquest treball.

Com a segona alternativa es va valorar Unity, una plataforma àmpliament consolidada a nivell professional que compta amb un mercat d'actius (Asset Store) molt extens i fòrums plens de documentació útil. Tot i la seva potència, l'executable requereix una instal·lació massa pesada per a ordinadors portàtils estàndard i compta amb una corba d'aprenentatge més complexa per a projectes curts. A més, les darreres decisions de l'empresa respecte a la inestabilitat en la seva política de llicències de distribució van generar desconfiança tècnica, actuant com un motiu de pes per descartar el seu ús en aquesta memòria.

Finalment, l'estudi va incloure Unreal Engine, reconegut actualment com el motor capdavanter en renderització gràfica d'alt rendiment i processament visual. Malgrat oferir un entorn de programació visual molt avançat, la seva arquitectura està dissenyada gairebé en la seva totalitat per a entorns tridimensionals (3D) de gran envergadura. El programari exigeix uns requeriments de maquinari molt elevats (targetes gràfiques dedicades), convertint-lo en una opció totalment inviable i sobredimensionada per a la naturalesa bidimensional d'aquest videojoc.

2.3.2 Tecnologies escollides

La decisió final va determinar la implementació de Godot Engine com a nucli del desenvolupament. Els factors determinants van ser la gestió nativa de components en dues dimensions (2D) i l'arquitectura basada en programari lliure. La construcció del projecte mitjançant un sistema d'escenes i nodes independents va garantir una segmentació modular eficient de tots els elements interactius.

Pel que fa a la codificació de la lògica de joc, s'ha utilitzat GDScript. Aquest llenguatge d'alt nivell presenta una sintaxi neta (molt propera a Python) que optimitza els temps de desenvolupament; d'aquesta manera, el treball es va poder centrar en la resolució d'algorismes mecànics i el depurat de col·lisions en lloc de gestionar estructures sintàctiques complexes.

El control de versions distribuït i la traçabilitat del codi font es van resoldre mitjançant la plataforma GitHub. L'ús d'aquesta eina va assegurar la disponibilitat de còpies de seguretat centralitzades al núvol i va permetre realitzar protocols de reversió de canvis (rollbacks) davant d'errors crítics en l'executable. L'absència de costos financers i l'existència d'una comunitat de suport activa van validar l'elecció d'aquest ecosistema tecnològic.

2.4 Estructura del projecte

L'arquitectura del programari dins de Godot Engine s'ha dissenyat sota un patró modular. Cada entitat operativa funciona de manera independent dins de la seva pròpia escena encapsulada, una metodologia que minimitza la redundància de codi i optimitza el manteniment dels nivells.

L'arrel del sistema s'inicia en una escena global (Main) encarregada de gestionar el menú principal. Aquest node inicial controls el flux cap a la càrrega del mapa inicial (Nivell 1) o cap a la interfície de configuració. Tot i que l'experiència d'usuari està dissenyada com una progressió lineal al llarg de 10 mapes, s'ha implementat un selector de nivells per facilitar les tasques de diagnòstic i proves de rendiment.

Els components nuclears de l'aplicació es divideixen en les següents instàncies modulars:

- **Escena Player:** Administra l'entitat de l'usuari (el conill), integrant els vectors de moviment, la física de salt i el script de control d'inputs.
- **Escena Enemy:** Funciona com a plantilla base per a les entitats hostils, governant les rutes de patrulla automatitzades mitjançant sensors de detecció de límits.
- **Escena Carrot:** Actua com l'actiu col·leccionable del mapa; inclou una rutina que afegeix unitats al comptador global i allibera l'objecte de la memòria del sistema (subrutina queue_free) en interceptar el jugador.
- **Sistema HUD:** Capa de visualització gràfica connectada de forma fixa a la càrrega de la càmera (CanvasLayer) que actualitza els indicadors de salut i els marcadors de progressió.
- **Menús de Pausa i Victòria:** Panells visuals amb interfície de fusta que s'activen per interrompre el temps d'execució del motor (pausa del joc) o per gestionar la terminació exitosa del desè mapa.

El flux de l'aplicació s'inicia obligatòriament en el menú principal del videojoc. Des d'aquesta interfície, el sistema permet l'accés a l'apartat de configuració o executa la càrrega del primer escenari actiu. A partir d'aquí, l'usuari experimenta una progressió mecànica lineal i continuada que enllaça de forma seqüencial els nivells compresos entre el segon i el novè mapa. La fita final del programari s'ubica en el desè nivell, on s'activen els condicionants terminals de la partida. Si la variable de salut de l'entitat es redueix a zero unitats, el sistema desvia l'usuari de manera immediata cap a la pantalla de Game Over per després reubicar-lo en el menú d'inici. Per contra, si es completen els objectius d'aquest darrer mapa, el joc valida la sessió executant la pantalla de victòria total i retornant el control a l'arrel de l'aplicació.

Aquesta distribució per mòduls tancats assegura l'escalabilitat del programari. L'addició de nous escenaris o diferents tipus d'enemics es pot realitzar de manera immediata, aprofitant completament les classes i les propietats hereditàries de les escenes que ja estan programades.

2.5 Descripció dels components

2.5.1 Player

L'entitat principal del programari s'estructura com un objecte cinemàtic dependent de la classe CharacterBody2D. Aquest mòdul unifica el control vectorial de la instància de l'usuari (el conill), integrant els desplaçaments horitzontals, el càlcul d'impuls vertical (salts) i els llindars de fricció amb el mapa.

La seva lògica interna capta els senyals d'entrada físics (mapeig de teclat) i els processa dins del mètode d'actualització de físiques (_physics_process). Les constants del sistema apliquen una acceleració gravitatòria contínua quan el sensor de contacte inferior (is_on_floor) retorna un valor fals. Blocs de codi addicionals dins d'aquest node administren la variable global de salut de l'entitat i emeten senyals dinàmics cap al gestor gràfic per commutar els estats de l'animació visual (re repòs, carrera o suspensió aèria).

2.5.2 Enemy

Aquest component encapsula la lògica d'execució de les entitats hostils del mapa actuant sota un patró d'intel·ligència artificial preprogramat. El sistema utilitza vectors de desplaçament automàtic lineals combinats amb nodes de detecció de buits o col·lisió lateral (RayCast2D) per invertir automàticament el sentit de la marxa en arribar als límits d'una plataforma.

L'entitat compta amb un node d'àrea interactiva (Area2D) dedicat a interceptar la caixa de col·lisió del jugador. En el moment que el sistema valida un encreuament de geometries (superposició de hitboxes), el script de l'enemic acciona una subrutina de dany immediat que descompte unitats en la matriu de salut de l'usuari, enviant de forma instantània una petició d'actualització gràfica al visualitzador de la interfície.

2.5.3 Carrot

Aquest element s'ha dissenyat com una escena modular estàtica destinada a funcionar com a objectiu secundari col·leccionable. La seva infraestructura es fonamenta en un node d'àrea de detecció que roman a l'espera de registrar col·lisió amb la màscara específica del personatge principal.

Quan s'activa el senyal d'intersecció geomètrica, el programari executa una funció en cadena: afegeix una unitat sencera al registre de puntuació acumulat en el script central de la partida i inicia l'ordre d'alliberament d'actius en memòria (queue_free) per retirar l'element de l'arbre de nodes actiu. Aquest mecanisme regula la corba d'interès del jugador mitjançant la recompensa directa dins de l'entorn de joc.

2.5.4 HUD

L'entorn visual d'informació permanent (Heads-Up Display) utilitza la capa de nodes de control (CanvasLayer) per fixar elements gràfics directament sobre la pantalla, impedint que es desplacin amb el moviment de la càmera de joc. L'arquitectura del component allotja nodes de text dinàmic i contenidors de textures configurats per mostrar l'estat en temps real.

El sistema manté una connexió lògica amb els comptadors de variables globals de l'aplicació, actuant com a receptor de senyals cada vegada que hi ha alteracions en els paràmetres crítics. El mòdul actualitza els elements visuals dels cors de vida, incrementa

els marcadors numèrics d'ítems recopilats i llança cadenes de caràcters emergents per indicar esdeveniments terminals del sistema, com ara la superació d'un nivell o la mort del personatge.

2.5.5 Game Manager

Aquest component centralitzat funciona com l'administrador de l'estat de joc i opera sense representació gràfica directa en els nivells (node de tipus Node o Autoload). La seva funció és centralitzar la presa de decisions globals de l'aplicació per mantenir la independència del codi de les escenes i eliminar la duplicació de funcions.

El script allotja els mètodes primaris per controlar les pauses del motor gràfic, governar la càrrega seqüencial de fitxers de mapa, coordinar el buidatge de memòria en reiniciar escenaris i gestionar els disparadors del final de la sessió. El mòdul actua com el nucli d'interconnexió entre els menús de control i les instàncies jugables, garantint la integritat del flux del programari.

2.6 Definició de les tasques

2.6.1 Prova 1 – Sistema de moviment

El primer assaig tècnic es va centrar en el sistema de moviment amb l'objectiu de determinar el mètode de programació òptim per garantir la fluïdesa, precisió i estabilitat del desplaçament horitzontal i vertical del personatge principal. Per a aquesta validació es van sotmetre a estudi dues arquitectures de codi diferents. La primera consistia en una alteració manual i directa de les coordenades vectorials de la instància en l'espai mitjançant script, mentre que la segona alternativa aprofitava el motor de físiques natiu de Godot a través de la classe específica CharacterBody2D i la seva funció d'execució integrada move_and_slide(). Durant les simulacions s'alteraven de manera iterativa els paràmetres de potència de salt, els coeficients de fricció i les constants d'acceleració gravitatòria per avaluar-ne el comportament en diferents superfícies de l'entorn de treball.

Els resultats van demostrar que el sistema de desplaçament manual per coordenades generava fallades crítiques d'intersecció geomètrica, provocant que l'entitat travessés els límits dels blocs o s'encallés en els segments verticals del mapa. Per contra, l'algorisme natiu de Godot va resoldre automàticament els vectors de lliscament i la detecció de superfícies sense generar pèrdues de control. Com a conseqüència directa d'aquesta prova, es va prendre la decisió de descartar definitivament la programació manual de coordenades, incorporant el node CharacterBody2D com la base estructural definitiva per a tot el control de l'usuari.

2.6.2 Prova 2 – Sistema de col·lisions

La segona línia d'assaig se va dissenyar per validar la precisió mil·limètrica del sistema de col·lisions, concretament en les rutines d'intersecció de geometries entre el jugador, els actius col·leccionables i les entitats hostils. El protocol de treball va consistir a testejar diferents topologies de caixes de col·lisió (CollisionShape2D), comparant l'ús de geometries primitives com rectangles i cercles en contraposició a polígons complexos adaptats de forma exacta al contorn dels fulls de sprites. Les proves van sotmetre el programari a situacions d'estrès cinemàtic, forçant col·lisions a alta velocitat, salts ajustats en els vèrtexs dels blocs i impactes simultanis de caixes de dany o hitboxes.

Les dades obtingudes en els mesuraments van revelar que les màscares poligonals d'alta densitat de vèrtexs generaven un consum ineficient de temps de processament i provocaven una sobrecàrrega inútil de la CPU sense aportar cap millora real a l'experiència de joc. Per aquest motiu, es va prendre la decisió tècnica d'estandarditzar l'ús de formes geomètriques primitives i simplificades ajustades al nucli visual de cada objecte gràfic. Aquesta configuració simplificada va eliminar completament els errors de registre d'impacte i va mantenir el consum de recursos sota els mínims tolerables pel sistema d'execució.

2.6.3 Prova 3 – Rendiment

Finalment, la tercera prova es va enfocar en el rendiment gràfic per tal de garantir l'estabilitat de l'aplicació en sistemes de computació d'escriptori estàndard sense necessitat de targetes dedicades. Per dur a terme aquest experiment, es va dissenyar un entorn de proves d'estrès saturant un mapa experimental amb una densitat anòmala d'instàncies actives, incloent enemics en patrulla simultània, objectes col·leccionables animats i elements visuals de fons. Es va monitoritzar l'execució mitjançant el comptador

de rendiment intern del motor per registrar qualsevol fluctuació en la taxa de refresc de la pantalla durant les situacions de màxima interacció.

La saturació inicial d'elements va provocar caigudes severes en la taxa de fotogrames i fenòmens d'stuttering, derivats d'una superposició accidental de nodes col·liders duplicats i d'una manca d'optimització en les dimensions de determinats arxius d'imatge. La resolució tècnica d'aquest problema va requerir una refactorització profunda del mapa per eliminar els col·liders redundants de l'arbre de nodes, a més de reajustar les textures als paràmetres òptims de resolució del projecte. Aquestes accions correctives van aconseguir fixar la taxa de refresc en 60 FPS totalment estables, assegurant la fluïdesa del programari fins i tot en els escenaris amb major càrrega computacional i de disseny.

2.7 Definició de les funcionalitats

2.7.1 Funcionalitat 1 – Iniciar partida

El mòdul d'arrencada inicial governa la transició directa des de la interfície d'usuari cap a l'entorn interactiu executable. En prémer l'actuador de començament de joc, el script de control invoca el mètode de canvi d'escena natiu del motor, realitzant una càrrega immediata del fitxer de codi del primer mapa en un interval de temps gairebé imperceptible. Aquesta operació evita la necessitat de dissenyar subrutines complexes de càrrega asíncrona i garanteix una neteja en el bloc de dades del fil d'execució. En paral·lel a la inicialització visual, el sistema executa una funció de reinici de variables estructurals de la sessió.

Aquest procés restableix la matriu de salut del personatge a un valor complet de tres unitats actives, buida completament el registre comptador d'objectes col·leccionables i assigna les coordenades de vectors de posició inicials a la instància del jugador. Així mateix, el motor activa els arbres de nodes que regeixen el processament de les entitats hostils i renderitza tots els elements estàtics distribuïts per l'escenari sense comprometre la pila de memòria del sistema.

2.7.2 Funcionalitat 2 – Progressió i selecció de nivell

L'arquitectura lògica de l'experiència jugable utilitza una seqüència de progressió lineal segmentada en 10 nivells independents. Cada mapa conté un node sensor de fita terminal connectat per senyals a l'administrador global de la partida; en el moment exacte en què la caixa de col·lisió del jugador intercepta el llindar final, el codi executa una subrutina d'alliberament del mapa vigent i carrega l'escena corresponent a l'índex numèric immediatament superior de manera automatitzada.

Per oferir mètodes alternatius de control de flux, s'ha programat una eina modular de selecció d'escenaris acoblada al menú inicial de l'aplicació. Aquest panell interactiu mapeja de manera directa cadascun dels fitxers d'escena de la memòria de treball, permetent a l'usuari enviar ordres directes de càrrega per saltar a mapes avançats com els entorns de coves. Aquesta funcionalitat actua de manera autònoma i isolada, assegurant la viabilitat de les proves de depuració en qualsevol tram del recorregut sense obligar a l'execució sencera del programari.

2.7.3 Funcionalitat 3 – Control del personatge

El control dinàmic de l'entitat principal es fonamenta en la intercepció i processament dels esdeveniments de teclat lligats a l'actualització de les físiques del motor. El script utilitza la crida d'inputs en temps real per modificar de manera directa els vectors de velocitat

horitzontal i aplicar forces d'impuls vertical inverses al vector de gravetat establert en el sistema d'execució.

Aquesta configuració proporciona una resposta immediata i un control suau durant les accions combinades de desplaçament en l'eix de les abscisses i de salt en l'eix de les ordenades, evitant problemes de bloqueig mecànic. De forma subordinada a la velocitat cinemàtica, un gestor d'estats s'encarrega d'analitzar contínuament el valor de les variables de moviment i el contacte amb la superfície per commutar els fragments d'animació del full de sprites en funció de si el personatge es troba en posició de repòs, en carrera contínua o en fase de suspensió aèria.

2.7.4 Funcionalitat 4 – Sistema de vides i mal

El subsistema de salut té com a funció registrar les pèrdues de recursos derivades del contacte no desitjat amb les caixes de dany presents en els enemics o en l'entorn. En produir-se la superposició geomètrica de màscares de col·lisió, la lògica del component executa una rutina de reducció que resta una unitat sencera del valor actual de la salut i activa un estat temporal d'invulnerabilitat.

Aquest canvi d'estat es comunica visualment mitjançant una variació de l'opacitat en el bucle de renderitzat del personatge i modifica el disseny de la interfície gràfica canviant els cors plens per textures de contenidor buit. Si el valor analitzat en el comptador de vides arriba a zero unitats, el motor suspèn momentàniament els scripts actius, interromp el flux de la partida i desplega el node terminal de final de joc, oferint alternatives d'execució per reiniciar el nivell des de la memòria o retornar el control a l'arrel de l'aplicació.

2.7.5 Funcionalitat 5 – Recollir pastanagues

La captura d'elements col·leccionables s'executa a través d'estructures geomètriques de detecció estàtiques basades en la classe de nodes d'àrea interactiva. Aquests elements s'ubiquen en coordenades estratègiques de les plataformes dels escenaris, romanent en un estat d'espera passiva fins que registren la intersecció de la capa lògica assignada al personatge.

La validació del senyal de contacte desencadena l'execució instantània d'un mètode que incrementa en una unitat el valor emmagatzemat en la variable d'ítems recollits de la partida i envia una ordre de refresc gràfic directament al text visible de la interfície. Un cop completada la transmissió de dades a temps real, el programari fa una crida a la funció d'alliberament estructural per eliminar el node de l'objecte de la memòria dinàmica, eliminant la persistència gràfica de l'element i optimitzant el consum de recursos.

2.7.6 Funcionalitat 6 – Menú de pausa

La interrupció voluntària del temps d'execució del joc s'ha resolt mitjançant l'escriptura d'una rutina associada de manera fixa a la tecla d'escapament. En validar-se la polsadura d'aquest botó, l'script principal canvia el paràmetre d'estat del motor gràfic a mode de pausa, aturant de forma immediata els vectors d'acceleració, el processament de les intel·ligències artificials de patrulla i el comptador intern de temps.

Aquesta acció superposa a la pantalla una interfície gràfica d'usuari tancada, dissenyada amb estètica de fusta, que funciona de manera independent a l'estat congelat del nivell de fons. El panell allotja una matriu de botons amb lògiques definides per permetre la represa de l'arbre de nodes per on s'havia deixat, la reexecució del script de càrrega del mapa des de zero per corregir errors de l'usuari, o el tancament net de l'escena jugable per restablir el flux del programari cap al menú principal de l'aplicació.

3 Altres capítols

Aquest apartat analitza de manera exhaustiva la implementació de la jerarquia de nodes, la configuració tècnica de les escenes i els mecanismes d'interconnexió que articulen l'executable. S'exposa el procediment metodològic seguit dins de l'inspector i els panells de configuració de Godot Engine per vertebrar cadascun dels components del sistema.

3.1 Anatomia i estructura del jugador (Player)

La construcció de l'entitat principal s'articula mitjançant un node arrel de la classe CharacterBody2D, seleccionat per la seva capacitat nativa per gestionar cossos cinemàtics en entorns bidimensionals. Des de l'inspector es van calibrar els paràmetres de gravetat, fricció i vectors de moviment; aquesta configuració permet invocar de manera eficient el mètode move_and_slide(), el qual processa automàticament els algorismes de lliscament i detecció de col·lisions superficials, evitant errors d'intersecció o bugues de traspàs de geometries.

Pel que fa al sistema d'animació, s'ha utilitzat un node AnimatedSprite2D vinculat a un recurs de tipus SpriteFrames gestionat des del panell inferior de l'editor. Les seqüències de moviment es van estructurar segmentant els fotogrames a partir d'un full de gràfics unificat; la taxa de refresc de l'animació es va fixar en un rang d'entre 10 i 12 FPS per tal d'assegurar una transició visual orgànica en la carrera de l'entitat, deshabilitant la propietat de bucle (Loop) en l'estat de mort per evitar la repetició indefinida de la seqüència terminal.

La delimitació de l'espai físic de l'entitat es va resoldre mitjançant un node CollisionShape2D configurat amb una geometria de càpsula, realitzant un ajust manual dels punts de control per cobrir exclusivament el tors del personatge i excloure les orelles de la màscara de col·lisió activa. Aquesta decisió de disseny prevé situacions de bloqueig cinemàtic involuntari quan el jugador es desplaça sota plataformes baixes on, segons la representació gràfica de l'entorn, la instància hauria de poder transitar lliurement.

L'apartat acústic lligat a les accions del personatge s'integra mitjançant un node AudioStreamPlayer2D que allotja el fitxer de text sonor jump3.wav. Dins de les seves propietats es va desactivar la casella d'execució automàtica (Autoplay) per impedir l'activació del recurs en iniciar l'escena, sotmetent la reproducció del fitxer a l'emissió de l'ordre d'impuls vertical gestionada per l'script de control.

L'enquadrament visual de la partida utilitza un node Camera2D subordinat directament a la jerarquia del jugador, activat mitjançant la propietat Enabled de l'inspector. Per garantir la integritat de la visualització, es van acotar manualment les coordenades dels límits (Limits) a la matriu de píxels de cada nivell, forçant la càmera a detenir el seu recorregut en arribar als marges definits per al mapa i evitant que es renderitzi el buit gris per defecte de l'editor de Godot.

3.2 Arquitectura i lògica dels enemics (Guineu i Goblin)

La implementació de les entitats hostils s'ha estructurat utilitzant el principi de modularitat i reutilització d'escenes de Godot. La instància base de la Guineu s'ha encapsulat en un fitxer independent amb extensió .tscn, governat per un script que avalua contínuament les col·lisions horitzontals. En el moment que el sistema registra un contacte lateral amb un obstacle o límit d'entorn, s'executa una operació multiplicadora sobre la variable d'escala horitzontal del component (*scale.x = -1), provocant la inversió immediata de la matriu gràfica i del sentit vectorial de la marxa.

L'entitat Goblin presenta un disseny de comportament més complex basat en esdeveniments, incorporant un node Area2D anomenat AreaVista encarregat de delimitar el radi de detecció passiu de l'enemic. La connexió d'aquest mòdul es va realitzar mitjançant el panell lateral de senyals, connectant el mètode body_entered directament amb el codi del personatge; quan el sistema valida que l'objecte que intersecta l'àrea correspon a la màscara del jugador, el script commuta immediatament l'estat de l'enemic cap a la rutina i animació d'atac.

Per impedir que els enemics caiguin de les plataformes durant la seva patrulla, es van acoblar dos nodes de llançament de raigs independents, anomenats Rayo1 i Rayo2, orientats cap avall a la part davantera de la instància. Aquests nodes requereixen l'activació explícita de la propietat Enabled a l'inspector i es configuren amb un vector de projecció descendent de coordenades (0, 20); quan el raig deixa de registrar col·lisió amb les capes de terra, el codi detecta el límit de la superfície i força la inversió del vector de moviment.

3.3 Disseny de nivells i escenaris (de l'1 al 10)

La producció d'entorns, que inclou els 7 escenaris ambientats en zones boscoses i els 3 nivells ubicats en entorns de coves, utilitza la tecnologia del node TileMapLayer. La fase inicial d'edició va requerir la definició d'un recurs TileSet on es van importar les textures de les superfícies; dins del configurador de capes físiques (Physics Layer), es van traçar de manera manual les geometries blaves de col·lisió sobre cadascuna de les rajoles de la quadrícula, permetent posteriorment utilitzar les eines de pinzell de l'editor per maquetar els nivells de forma àgil sobre la reixilla espacial.

L'efecte de profunditat visual en les pantalles s'aconsegueix mitjançant el desplegament d'estructures de tipus ParallaxBackground que allotgen diferents nodes ParallaxLayer. La configuració d'aquests components es basa en el correcte calibratge del paràmetre d'espillat de moviment (Motion Mirroring), introduint els valors en píxels corresponents a la resolució exacta de la imatge de fons utilitzada; d'aquesta manera, el motor gràfic duplica la textura de manera contínua i infinita paral·lelament al desplaçament del jugador, simulant un fons amb moviment.

Amb l'objectiu de mantenir l'organització de l'arbre de nodes i evitar la saturació d'elements en la jerarquia del nivell, es va dissenyar un node node genèric de tipus Node2D anomenat ContenedorZanahorias per encapsular de manera agregada tots els col·leccionables de cada mapa. Aquest component es va configurar des de l'inspector amb un valor de Z Index igual a 2, mesura tècnica que va resoldre un conflicte gràfic d'ordre de renderitzat on les pastanagues quedaven incorrectament ocultes darrere els elements de la capa de decoració de fons.

3.4 Interfície d'usuari i HUD

L'entorn gràfic informatiu permanent s'ha aïllat del pla bidimensional on es desenvolupa l'acció interactiva mitjançant l'ús d'un node arrel CanvasLayer. Aquest node crea una capa de renderització independent que fixa de manera absoluta els elements de la interfície d'usuari sobre la pantalla, garantint que el HUD es mantingui estable a la coordenada d'origen (0,0) independentment de les variacions de posició i dels desplaçaments realitzats per la càmera de joc.

El dinamisme del comptador de salut es gestiona a través d'una escena HUD connectada a un contenidor horitzontal de tipus HBoxContainer que allotja les textures corresponents a les tres unitats de vida. El script que governa aquest mòdul monitoritza de manera contínua la variable de salut de l'entitat principal; quan es registra un esdeveniment de

dany que redueix el valor de vides, el codi localitza la posició exacta de la instància gràfica del cor afectat i altera la seva propietat de textura per carregar la imatge del contenidor buit, oferint informació instantània del seu estat de salut.

La visualització numèrica de la puntuació utilitza un node d'etiqueta de text de la classe Label. Per assegurar una correcta integració visual sobre els fons cromàtics dels nivells, es va accedir a l'apartat de sobreescriptura de temes (Theme Overrides) de l'inspector, modificant de manera manual el recurs de font tipogràfica utilitzat i incrementant l'escala dels caràcters per maximitzar el contrast i la llegibilitat durant la partida.

3.5 Sistema de meta i canvi de nivell

La fita terminal que determina la conclusió de cadascun dels 10 mapes utilitza un element interactiu basat en el node Area2D, representat gràficament com una bandera o portal. Aquest node incorpora un CollisionShape2D configurat amb un radi geomètric ampli per garantir la correcta intercepció per part de l'usuari; des del panell lateral de connexions del motor, es va enllaçar el senyal body_entered directament cap a les funcions de control d'escena del gestor global.

Quan la subrutina rep la notificació d'intersecció validada del personatge, s'executa la línia de codi lògica get_tree().change_scene_to_file(). La implementació d'aquest canvi de flux va requerir la revisió exhaustiva de les rutes de fitxers de recursos (res://) assignades a cada nivell, atès que qualsevol incongruència en la cadena de caràcters d'accés provoca una fallada crítica de càrrega asíncrona i el tancament immediat de l'aplicació per error de referència de programari.

3.6 Gestió de menús i flux de joc

L'organització del menú d'inici es fonamenta en l'ús d'un node de distribució vertical VBoxContainer, component encarregat d'aplicar algorismes d'alineació automàtica sobre els elements fills que configuren els botons d'accés ("JUGAR", "CONTROLES" i "SORTIR"). Cada botó té el seu senyal pressed connectat mitjançant programació a l'script central de la interfície, el qual avalua la petició d'entrada de l'usuari i commuta l'arbre de nodes cap a l'escena sol·licitada.

En escenaris on la variable de salut arriba al límit inferior de zero unitats, el gestor de la partida interromp la simulació de fons i desplega l'escena terminal de Game Over. Per evitar errors de persistència on el personatge podria reaparèixer sense punts de salut, l'activació del botó de reintent força l'execució d'una funció de neteja prèvia anomenada reset_stats(), rutina encarregada de restablir la matriu de 3 cors en el script de dades globals abans d'ordenar la recàrrega de l'arxiu del nivell des de la seva posició inicial.

4 Conclusions

4.1 Conclusions generals del projecte

L'execució d'aquest projecte de desenvolupament ha permès analitzar de manera integral el cicle de vida d'un programari interactiu des de la seva gènesi. La transició des de la conceptualització teòrica fins a la implementació pràctica de les lògiques dins de Godot Engine ha constatat la complexitat de traduir mecàniques de joc en algorismes funcionals i estables. L'adquisició de competències en el llenguatge GDScript constitueix una de les fites formatives més significatives d'aquest treball, especialment pel que fa a la gestió de cossos cinemàtics, el control de caixes de col·lisió i l'estructuració de projectes mitjançant arquitectures modulars i jerarquies d'escenes independents.

Més enllà dels aspectes de programació gràfica, el projecte ha exigit un alt nivell de rigor metodològic en la resolució autònoma de conflictes en el codi font (debugging). Les tasques sistemàtiques d'anàlisi d'errors han consolidat l'adquisició de bones pràctiques d'enginyeria de programari, com ara la documentació interna de les funcions i la implementació de fluxos de treball basats en el control de versions distribuït. L'ús de la plataforma GitHub ha resultat determinant per mitigar els riscos de pèrdua de dades de l'executable, validant la importància de mantenir entorns de desplegament centralitzats i traçables en qualsevol producció tècnica.

4.2 Consecució dels objectius

L'avaluació del programari final confirma el compliment de l'objectiu general establert en la fase de planificació. El videojoc de plataformes és plenament funcional, presenta una taxa de refresc bloquejada a 60 FPS estables i opera lliure d'errors crítics de memòria en els sistemes d'escriptori testejats. Respecte als objectius específics definits a l'inici del treball, el grau de consecució és pràcticament absolut gràcies al disseny modular del codi:

- **Mecàniques físiques:** El sistema de desplaçament i vectors de salt de l'entitat principal respon als inputs de l'usuari amb una latència mínima.
- **Intel·ligència artificial simple:** Les rutines de patrulla automatitzada dels enemics operen correctament gràcies a l'acoblament de sensors RayCast2D que inverteixen el sentit de la marxa davant de desnivells.
- **Disseny d'escenaris i col·leccionables:** Es van estructurar els 10 nivells previstos integrant lògiques d'intersecció Area2D que gestionen el sumatori de pastanagues i el seu posterior buidatge en memòria.
- **Gestió d'estats:** El subsistema de salut respon en temps real als impactes, coordinant la pèrdua de recursos de forma coherent amb la interfície gràfica d'usuari (HUD).
- **Control de flux:** La progressió entre escenaris s'executa de forma lineal automatitzada, incorporant un selector manual de nivells integrat en el menú que agilitza les tasques de diagnòstic i la validació de mapes específics.

Tot i que l'apartat estètic i la diversitat de comportaments dels enemics admeten futurs nivells de poliment, el nucli lògic de l'aplicació compleix de manera estricta amb els requisits de programari fixats.

4.3 Valoració de la metodologia i planificació

L'organització del treball sota les directrius del marc àgil Scrum ha demostrat ser una estratègia eficient per a la governança del projecte. Aquest sistema de gestió ha facilitat la fragmentació de la producció en iteracions setmanals de caràcter incremental, permetent avaluar el rendiment de l'executable sobre la marxa i aplicar accions correctives en les línies de codi sense comprometre l'estabilitat estructural de la resta d'escenes encapsulades. El cronograma inicial s'ha complert dins dels terminis establerts, garantint el lliurament del producte de forma puntual.

No obstant això, durant el desenvolupament de les fases de programació cinemàtica i calibratge de les col·lisions, es va detectar una desviació temporal que va requerir una reassignació d'hores de treball superior a la prevista en la planificació inicial. Aquesta incidència va obligar a flexibilitzar de manera interna el calendari de determinades tasques secundàries de maquetació; tot i així, l'ús combinat del diagrama de Gantt com a monitor visual de progrés i la gestió de branques paral·leles a GitHub va evitar l'aparició de colls d'ampolla crítics. Com a lliçó extreta per a futurs projectes de programari, es constata la

necessitat d'introduir marges de contingència més amplis en les estimacions de temps vinculades exclusivament a l'escriptura i depuració de codi font.

4.4 Visió de futur

L'estat actual de l'aplicació ofereix un entorn tancat i completament funcional, convertint-se alhora en una arquitectura altament escalable gràcies al seu disseny per components independents. Una línia d'expansió natural per al programari seria l'ampliació de la corba de dificultat mitjançant la creació de nous escenaris que incorporin elements dinàmics complexos, com ara vectors d'acceleració variables en plataformes mòbils o trampes interactives amb temporitzadors lògics. Així mateix, l'algorisme de les entitats hostils podria evolucionar cap a sistemes de persecució activa mitjançant la introducció de nodes de visió per raigs capaços d'avaluar la distància relativa respecte a la posició horitzontal del jugador.

En l'àmbit de la gestió de dades de l'aplicació, una millora crítica seria la implementació d'un subsistema de persistència mitjançant la codificació de rutines de guardat i càrrega de fitxers locals (en format JSON o mitjançant la classe ConfigFile de Godot). Aquest component permetria emmagatzemar l'estat exacte de la partida, conservant els registres d'ítems recopilats i els índexs de mapes superats entre diferents sessions d'execució. Finalment, l'arquitectura admetria la integració d'un panell de configuració avançat dedicat a la modificació dels paràmetres de resolució de pantalla, ajustos de ràtio d'aspecte i regulació del guany en els busos d'àudio del sistema.

5. Glossari

Aquest apartat defineix els termes tècnics, nodes i funcionalitats clau de Godot Engine utilitzats en el projecte, detallant el procés de configuració seguit en cadascun:

- **AnimatedSprite2D (Gestió d'animacions):** Aquest node és el que realment fa que el conill no sigui una imatge estàtica. El procés que vaig seguir va ser crear un recurs de SpriteFrames i anar "tallant" cada dibuix de la llista de moviments (correr, saltar, morir). Aquí vaig haver d'ajustar a mà la velocitat (FPS) perquè el moviment del personatge se sentís natural i no massa ràpid.

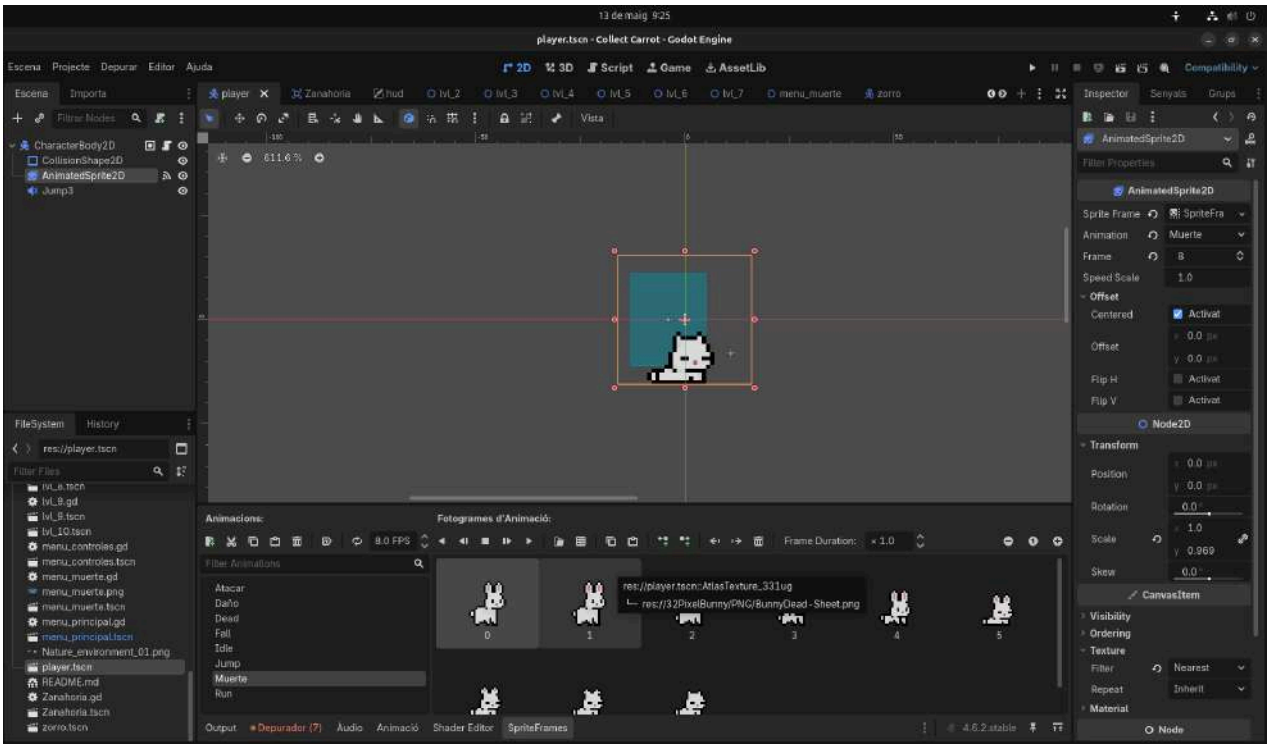


Figura 1: Interfície de Godot i animació de mort del personatge a SpriteFrames.

Aquesta captura mostra la seqüència de mort. Podeu veure com el personatge canvia de posició a terra; vaig haver d'ajustar manualment els fotogrames perquè l'animació s'activés just en el moment que la vida arriba a zero.

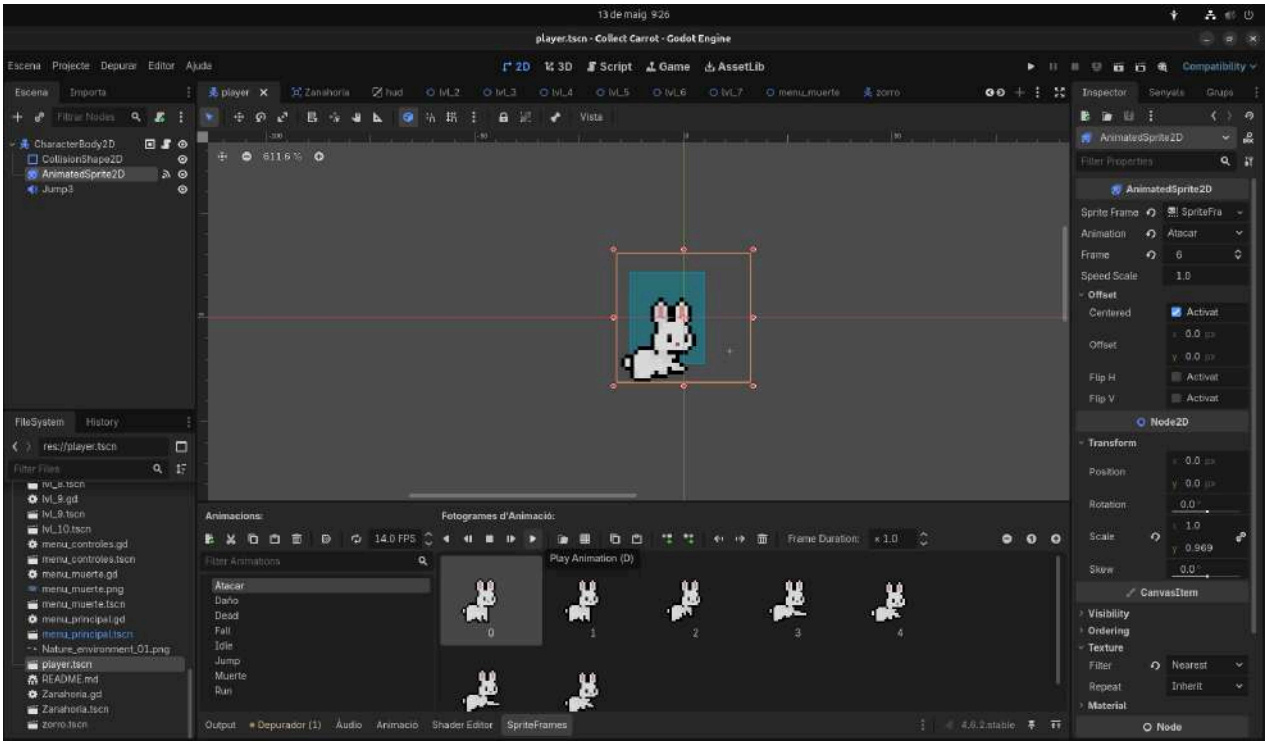


Figura 2: Configuració dels fotogrames de l'animació d'atac del personatge a l'editor de Godot.

En aquesta imatge es pot veure l'editor de fotogrames obert a la part inferior mentre configuro l'animació de quan el conill ataca. Aquí és on vaig tallar cada sprite individual de la fulla de personatges per crear la seqüència.

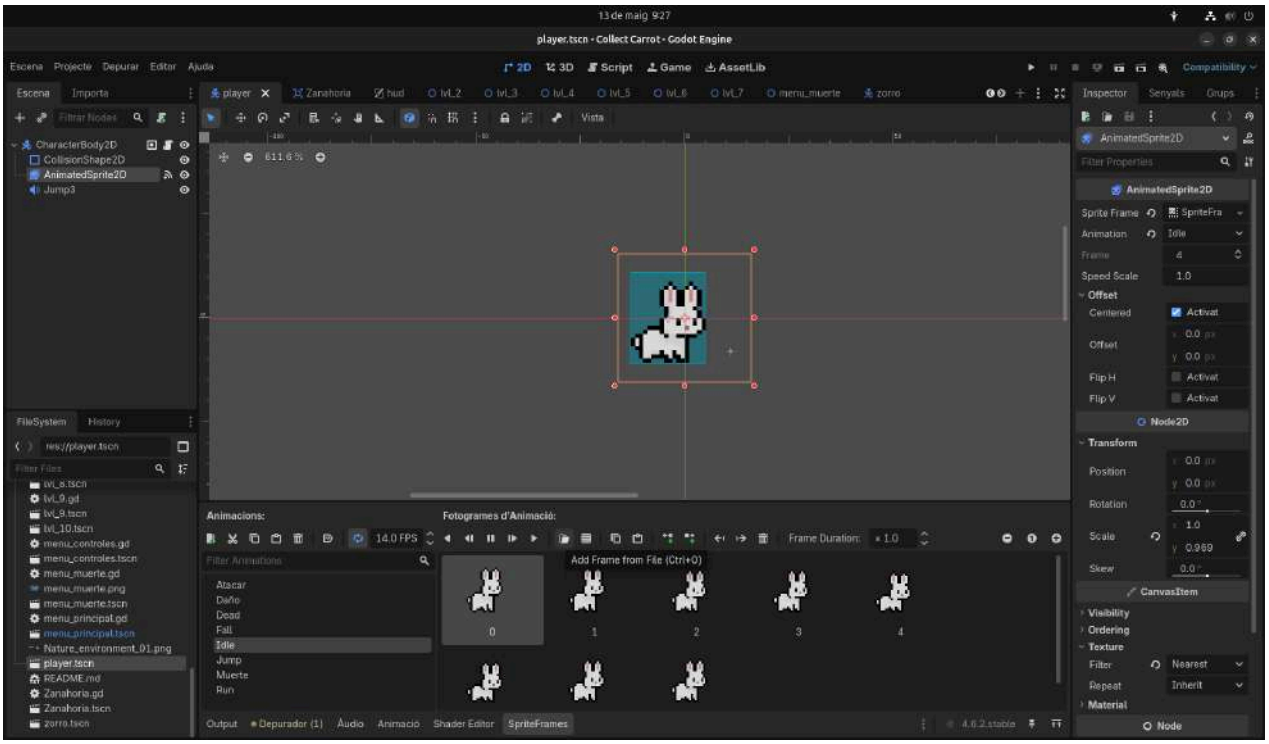


Figura 3: Panell de SpriteFrames amb la seqüència d'animació en estat de repòs (idle).

Aquí estic ajustant la velocitat de l'animació "Idle". És el moviment que fa el conill quan el jugador no prem cap tecla; vaig configurar una velocitat més lenta perquè l'animació respiratòria se sentís natural.

- **Area2D (Sensores i interaccions):** El vaig fer servir com un detector invisible. Per exemple, les pastanagues porten aquest node perquè el joc sàpiga quan el conill les està "trepitjant" sense que aquest xoqui i s'aturi. Quan el personatge entra en aquesta àrea, el node envia un avís (senyal) que fa que la pastanaga desaparegui i sumi un punt.

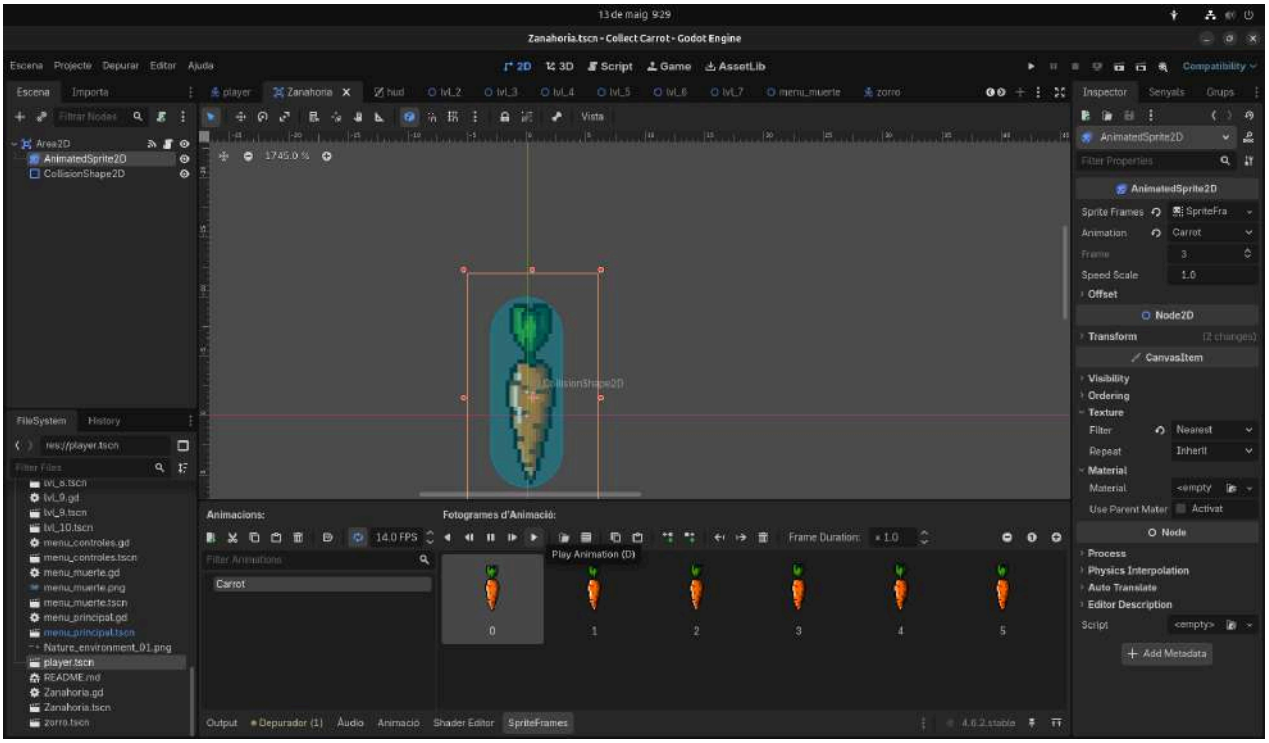


Figura 4: Configuració de l'objecte pastanaga (Zanahoria.tscn) i els seus fotogrames d'animació a Godot.

En aquesta captura es veu clarament l'estructura del node de la pastanaga. L'objecte està envoltat per una forma de col·lisió blava (la hitbox); quan el conill entra en aquesta àrea blava, el programa sap que s'ha de sumar un punt i eliminar l'objecte de la pantalla.

- **Vector2 (Càlcul de moviment):** Aquest és un dels conceptes més importants que he après. En un joc 2D, tot es mou mitjançant "vectors" (X per a l'horitzontal i Y per a la vertical). Per fer que el conill camini cap a la dreta, el meu script suma valor a la X, i per fer que caigui, li suma valor a la Y. Entendre com combinar aquests dos números ha estat clau perquè el salt tingui una corba natural.

```
14  func _physics_process(delta: float) -> void:
15      if esta_muerto:
16          velocity.x = 0
17          if not is_on_floor():
18              velocity += get_gravity() * delta
19          move_and_slide()
20          return
```

Figura 5: Codi de la funció _physics_process que gestiona el moviment i la gravetat de l'objecte quan està mort.

En aquesta captura es veu com es controla la velocitat del conill. Manipulo l'eix X per aturar-lo quan mor i l'eix Y sumant-li la gravetat constantment (get_gravity). La funció move_and_slide() és la que agafa aquests vectors i fa que el personatge realment es mogui.

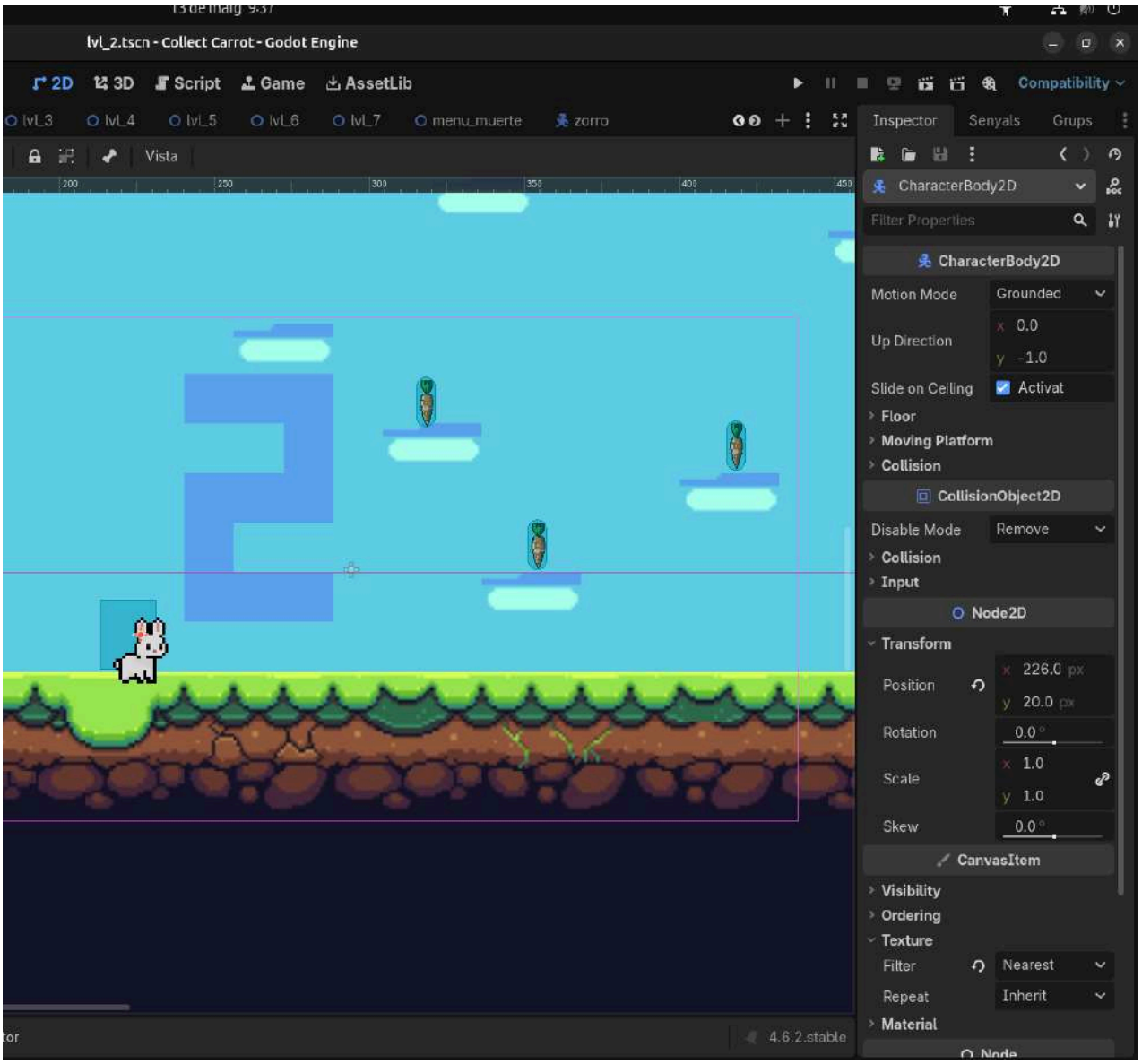


Figura 6: Vista de l'escenari de joc a l'editor amb el personatge CharacterBody2D i els nodes de col·lisió.

Aquí es veu el resultat físic. A l'inspector (pestanya dreta), es pot observar la Position amb els seus valors X i Y canviant segons el moviment. També es veu la Up Direction (y: -1.0), que és la configuració que permet al conill saltar cap amunt.

- **CanvasLayer (Capa d'interfície):** He fet servir aquest node per crear una capa de dibuix independent de la càmera del joc. L'he configurat per allotjar el HUD (vides i marcador), garantint que aquests elements es quedin sempre fixos a la pantalla mentre el conill es mou pel mapa.

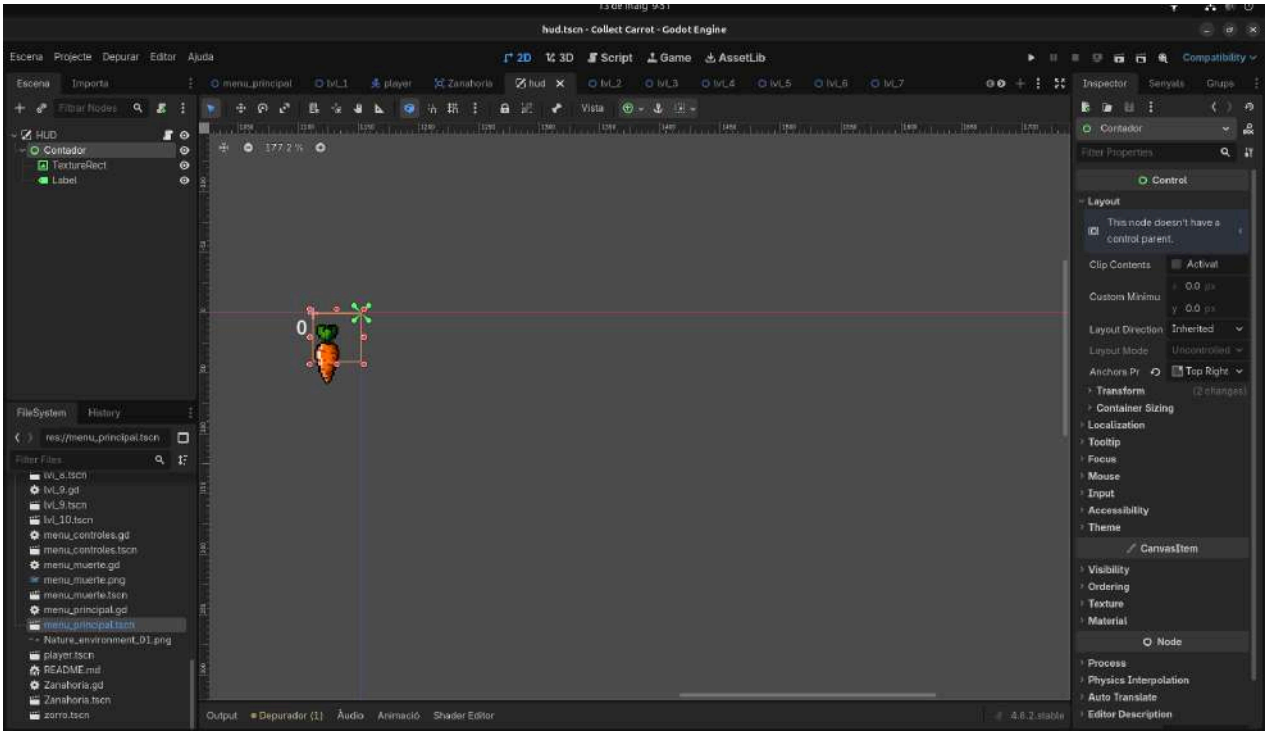


Figura 7: Disseny de la interfície d'usuari (HUD) amb el comptador de pastanagues recol·lectades.

En aquesta imatge es veu com estic configurant l'escena hud.tscn. A l'arbre de nodes de l'esquerra es pot observar el node Contador i els seus elements visuals (TextureRect per a la icona de la pastanaga i Label per al número). Aquesta estructura s'afegeix després al CanvasLayer del nivell principal per mantenir els punts sempre visibles a la cantonada de la pantalla.

- **CharacterBody2D (Lògica física)**: És el node arrel que he fet servir per al jugador i els enemics. Per configurar-lo, vaig haver de definir manualment la gravetat i la velocitat en el codi, utilitzant la funció move_and_slide() per gestionar les col·lisions amb el terra de forma fluida.

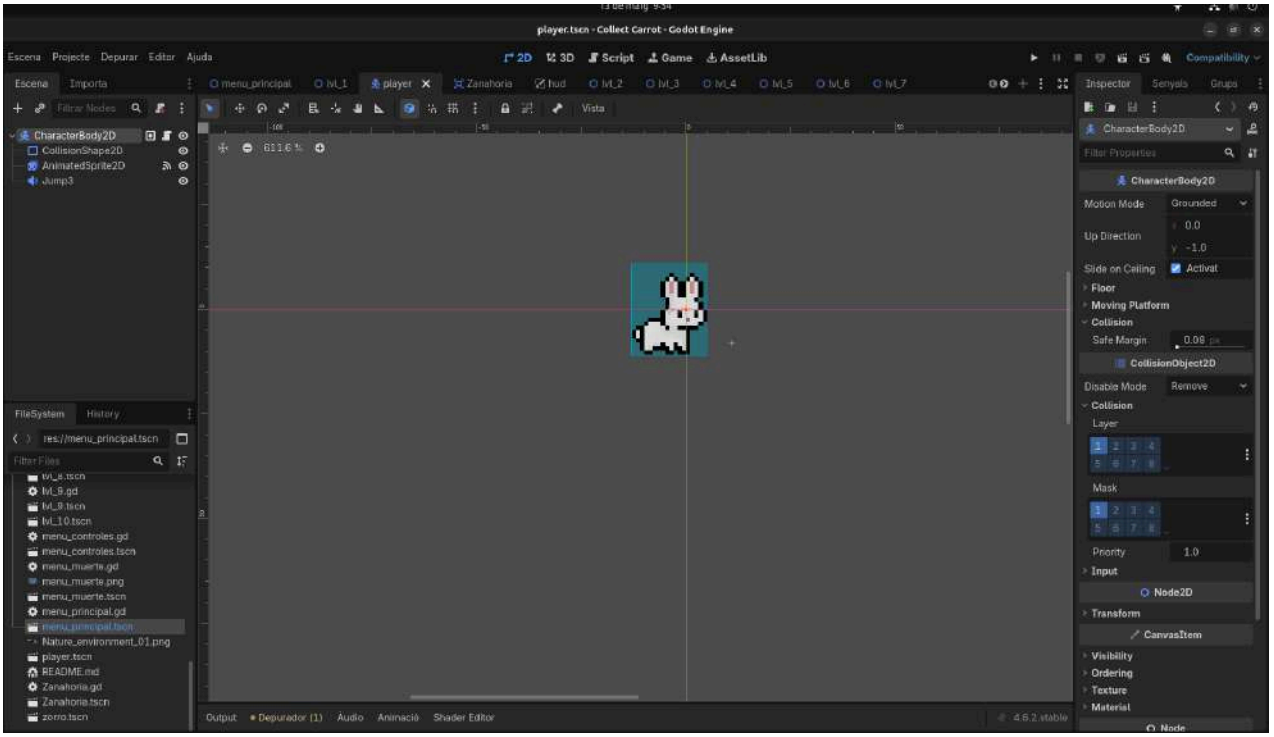


Figura 8: Configuració de les propietats i màscares de col·lisió del node CharacterBody2D del jugador.

En aquesta captura de l'escena del jugador, es pot veure a l'inspector com he configurat el Motion Mode en "Grounded" i la Up Direction en l'eix Y negatiu (-1.0). També és clau la secció de Collision, on he definit que el personatge pertany a la capa 1 (Layer) i que ha de detectar objectes en la mateixa capa (Mask), permetent així que el motor físic de Godot processi els xocs amb el terra i les plataformes.

- **CollisionShape2D (Hitbox):** Aquesta és l'àrea física que defineix on pot xocar un objecte. La vaig configurar seleccionant una forma de càpsula per al conill i quadrada per als enemics, ajustant manualment els vèrtexs perquè la col·lisió coincideixi bé amb el dibuix del sprite.

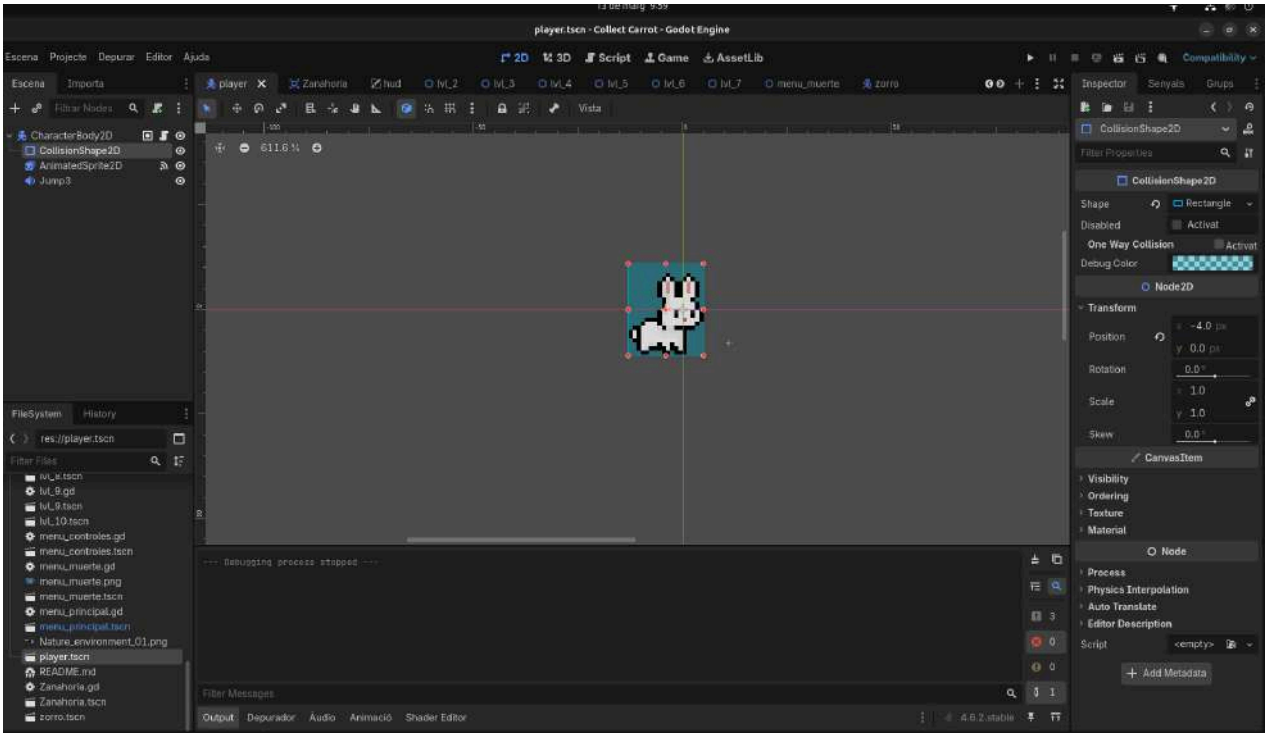


Figura 9: Ajust de la forma rectangular del node CollisionShape2D per definir l'àrea de contacte del personatge.

En aquesta imatge es veu el node CollisionShape2D en acció. He seleccionat una forma de rectangle (RectangleShape2D) per al cos del jugador. A l'editor visual es pot apreciar la caixa blava que envolta el sprite; aquesta caixa és la que realment "toca" el terra i els enemics, i l'he ajustada perquè no sobresurti massa del dibuix i així el moviment se senti més precís i just per al jugador.

- **HBoxContainer (Organitzador de cors):** És un node d'interfície que alinea els elements horitzontalment. L'he configurat dins del HUD perquè els cors de vida es mantinguin separats a la mateixa distància de forma automàtica, sense haver de col·locar-los d'un en un.

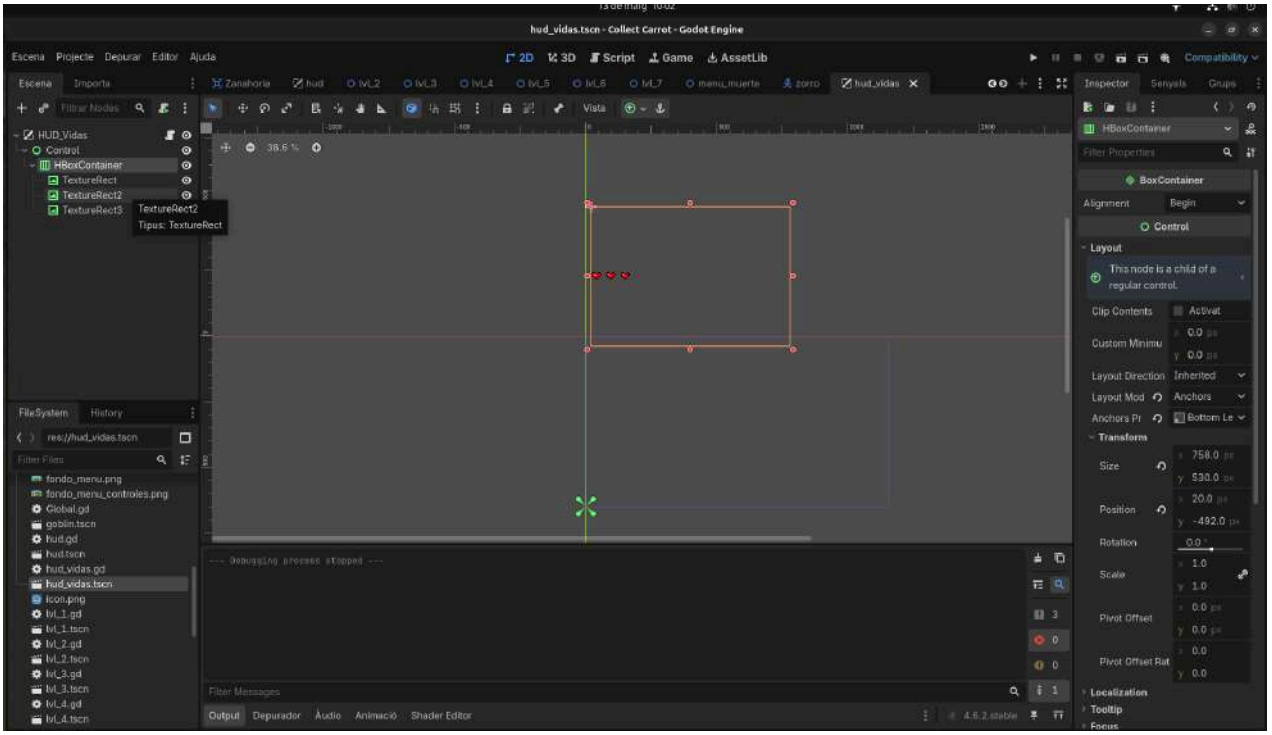
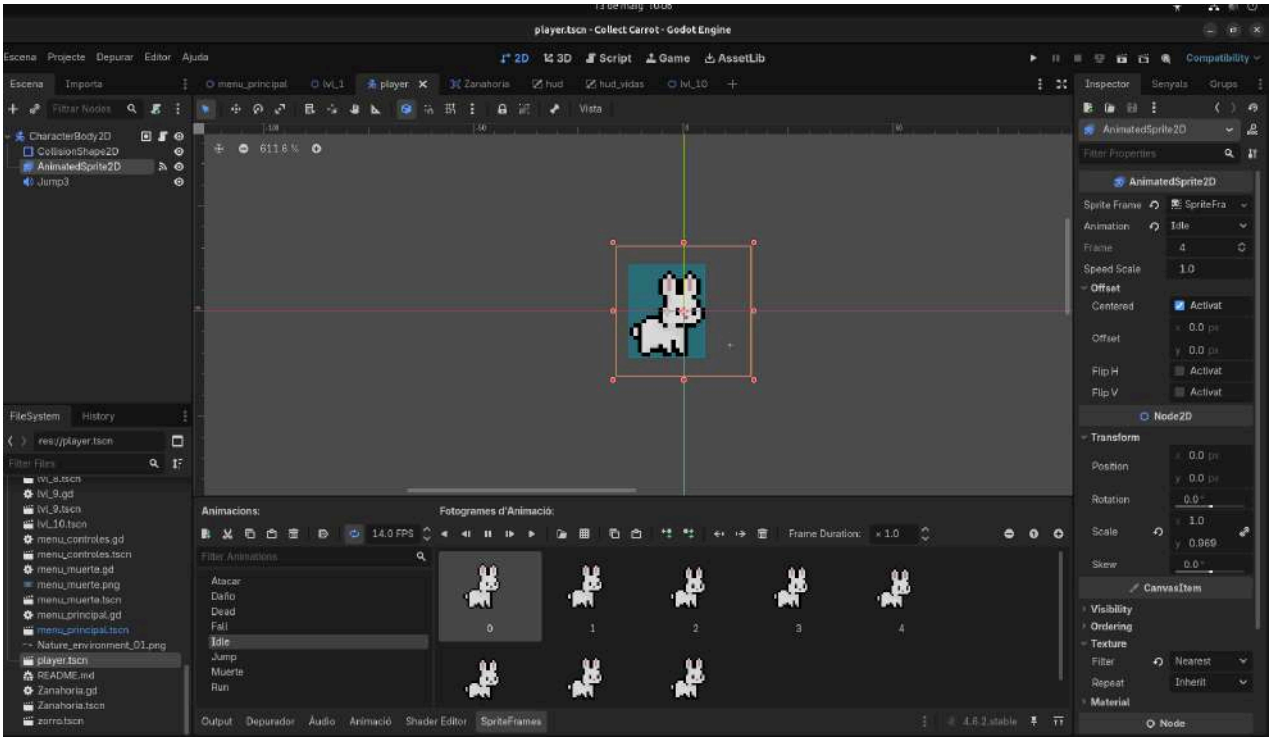


Figura 10: Configuració del contenedor HBoxContainer i els nodes TextureRect per a la visualització de les vides al HUD.

En aquesta captura de l'escena hud_vidas.tscn, es pot veure com el node HBoxContainer actua com a pare de tres nodes TextureRect (que són els cors). Gràcies a aquest node, si decideixo afegir més vida al personatge o canviar la mida dels icones, el contenidor els reordena automàticament en una fila perfecta sense que jo hagi de calcular les posicions X i Y de cada cor per separat.

- Offset (Correcció de posició): És un paràmetre que em permet desplaçar el sprite respecte al centre del node. El vaig configurar mitjançant codi (per exemple, offset.y = -12) per corregir visualment la posició del conill quan s'activa l'animació de mort i que no quedés mal col·locat.



En aquesta captura de l'inspector de l'AnimatedSprite2D, es pot veure la propietat Offset. Modificant els valors de l'eix X i Y, puc moure el dibuix del conill sense alterar la posició del seu motor físic ni de la seva caixa de col·lisió. Això és fonamental perquè les animacions quedin sempre ben alineades amb el terra, independentment de si el dibuix del conill canvia de mida entre fotogrames.

- **Camera2D (Enquadrament i seguiment):** És el node que s'encarrega de seguir el personatge perquè no surti mai de la vista del jugador mentre avança pel mapa. El vaig configurar per centrar l'acció en el conill, permetent que el món es mogui al seu voltant de forma natural i constant.

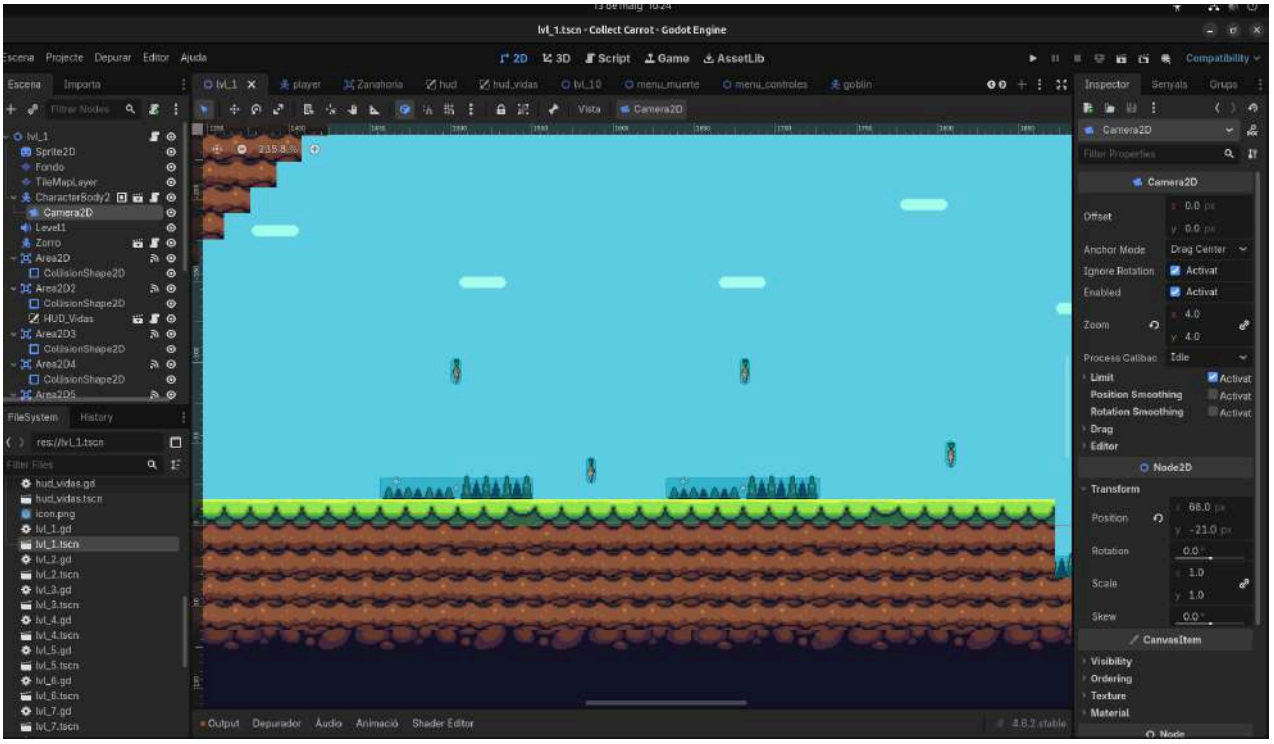


Figura 11: Configuració i límits del node Camera2D per al seguiment del jugador dins de l'escenari.

En aquesta captura es pot observar com he ajustat un Zoom de 4.0 a l'inspector per apropar la càmera i ressaltar el disseny del personatge. També vaig activar les propietats de Limit per bloquejar la visió als marges del nivell i el Position Smoothing, que fa que el moviment de la càmera sigui suau i no doni estrebades brusques.

- **Queue_free() (Alliberament de memòria):** És una funció de GDScript que elimina un node del joc definitivament. La vaig configurar perquè es disparés cada vegada que el jugador toca una pastanaga; així l'objecte desapareix i el joc no gasta recursos innecessaris.

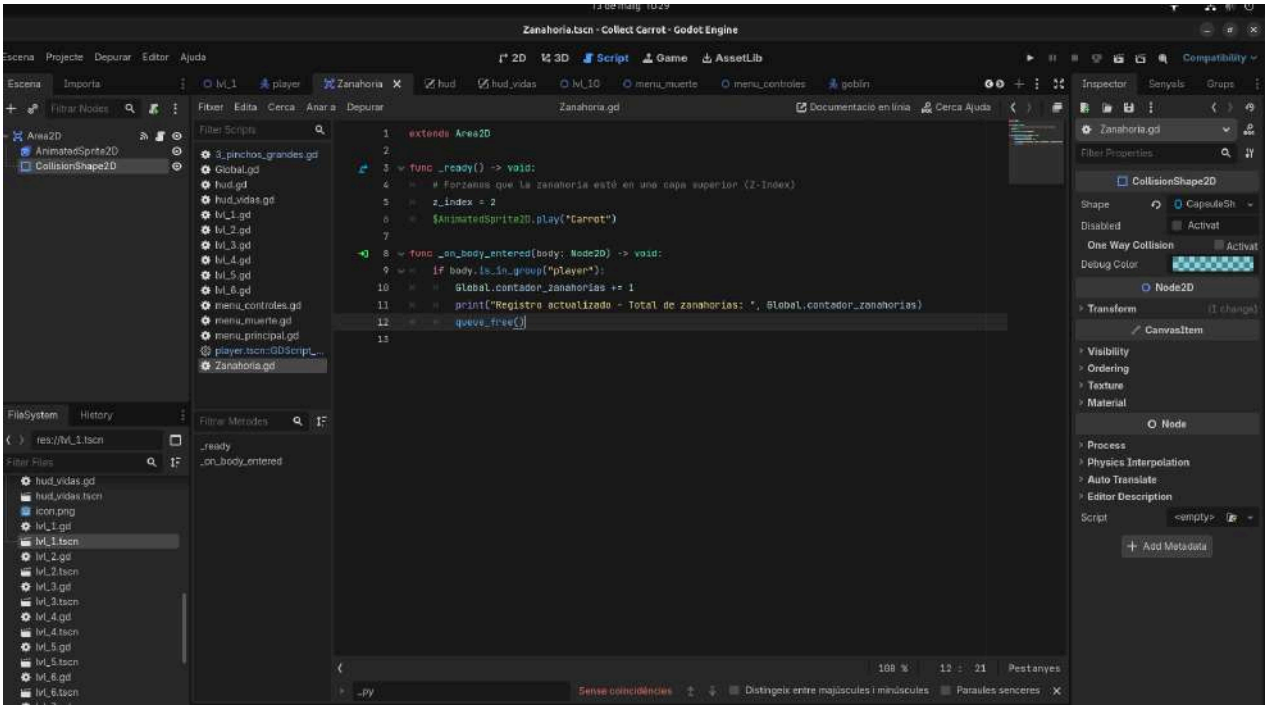


Figura 12: Codi de l'script Zanahoria.gd amb les funcions de configuració inicial i detecció de col·lisió del jugador.

En aquesta captura es pot veure el fragment de codi dins de l'script Zanahoria.gd on s'executa aquesta instrucció. Concretament, la funció queue_free() s'activa a la línia 12, just després que el sistema detecti que el jugador ha entrat en l'àrea de col·lisió de la pastanaga i s'hagi actualitzat el comptador global de punts.

- **RayCast2D (Sensor d'enemics):** El vaig fer servir com un raig invisible de detecció. L'he configurat activant la propietat Enabled i orientant-lo cap avall. Si el raig no detecta terra, l'enemic sap que ha arribat a un precipici i ha de girar.

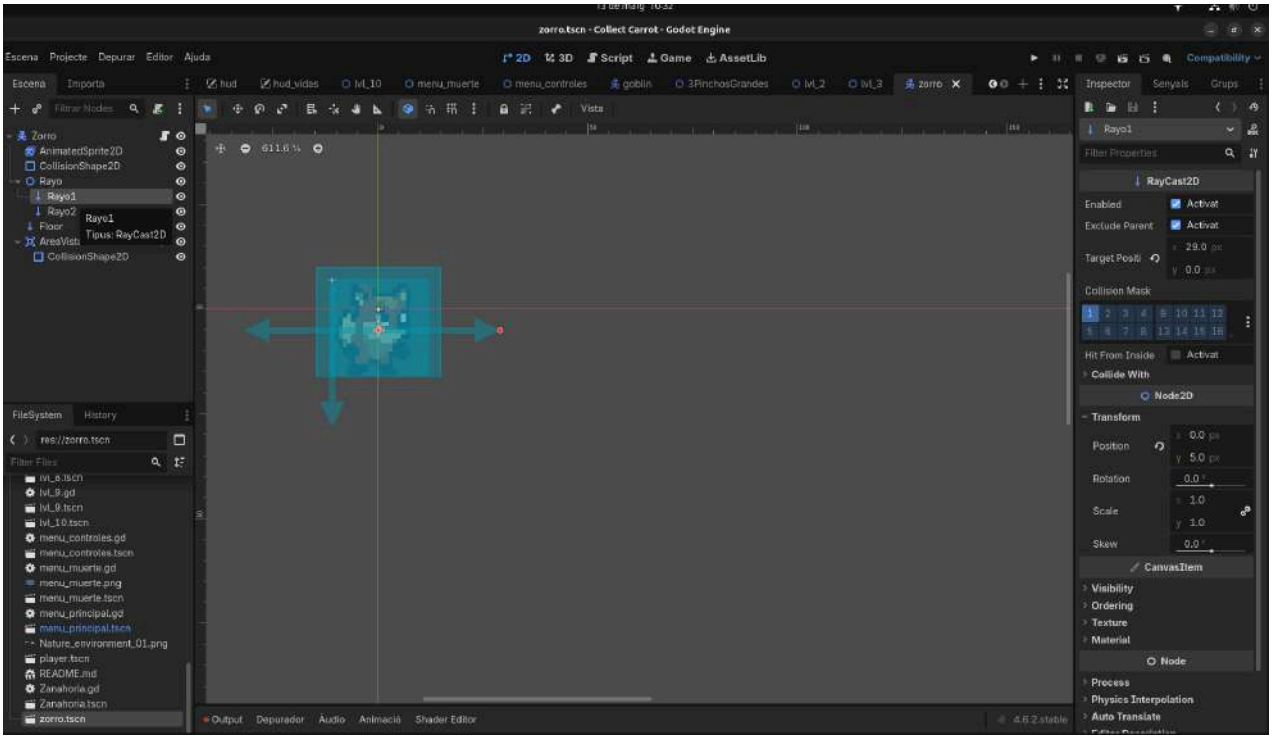


Figura 13: Configuració dels nodes RayCast2D per a la detecció de l'entorn de l'enemic (Zorro.tscn).

En aquesta captura es pot observar com el node Rayo1 (de tipus RayCast2D) està integrat dins de l'escena de l'enemic zorro.tscn. A l'editor visual es veu la línia que indica la direcció del raig cap a la dreta, i a l'inspector es pot veure la configuració del Target Position a 29.0 px en l'eix X.

Això permet a la guineu detectar obstacles abans de xocar-hi o saber si ha arribat al límit d'una plataforma per canviar de direcció.

- **Signals (Senyals de connexió):** És el sistema que fa que els nodes parlin entre ells. Els he configurat des de la pestanya "Node", connectant esdeveniments físics (com tocar la meta) amb funcions de l'script (com saltar al següent nivell).

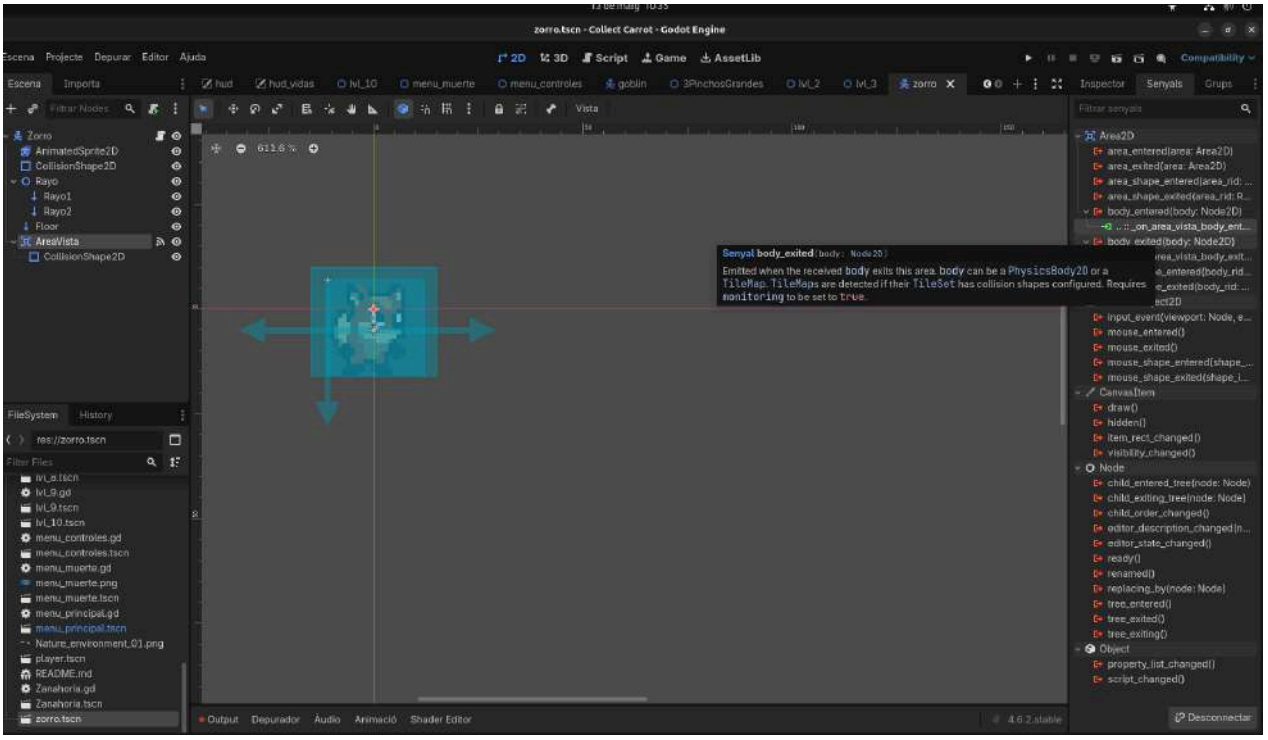


Figura 14: Connexió del senyal body_exited del node Area2D (AreaVista) a l'editor de Godot.

```
81
82  func _on_area_vista_body_exited(body: Node2D) -> void:
83      if esta_muerto: return
84      if body.is_in_group("player"):
85          esta_atacando = false
86          sprite.play("Run")
87          velocity.x = CorrerZorro * (-1 if sprite.flip_h else 1)
```

Figura 15: Codi de la funció _on_area_vista_body_exited per gestionar el comportament de l'enemic quan el jugador surt de la seva àrea de visió.

A la primera captura es pot veure la pestanya de senyals del node AreaVista de l'enemic, on el senyal body_exited apareix amb una icona verda que indica que està connectat a l'script. Com es mostra a la segona, aquesta connexió activa la funció _on_area_vista_body_exited, la qual detecta quan el jugador surt del rang de visió de la guineu per aturar l'atac (esta_atacando = false) i fer que l'enemic torni a córrer normalment.

- **TileMapLayer (Disseny modular):** Aquesta és l'eina que he fet servir per "pintar" els nivells. Vaig crear un TileSet on vaig definir manualment les capes de col·lisió perquè el terra pintat fos sòlid i el personatge no el travessés.

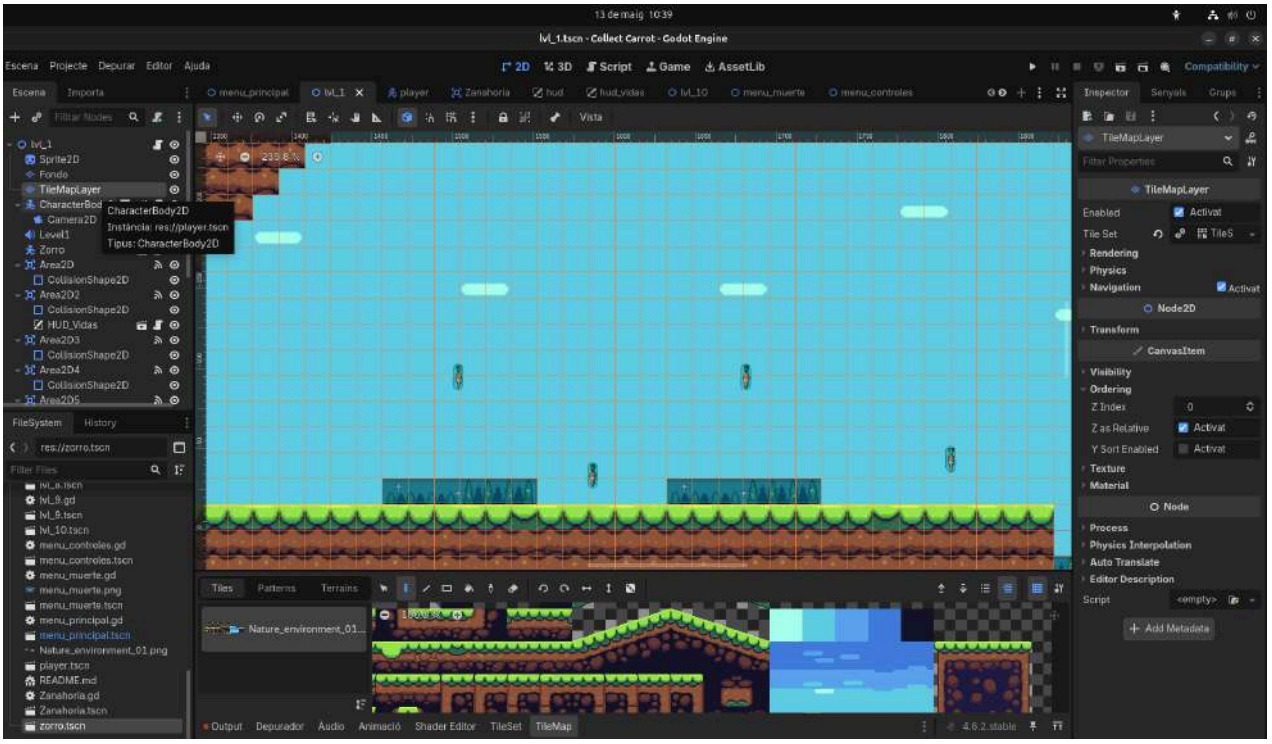


Figura 16: Editor de mapes de rajoles (TileMapLayer) i configuració del set de rajoles (TileSet) per al disseny del nivell.

En aquesta captura, es pot observar el node TileMapLayer seleccionat i la quadrícula blava activada sobre l'escenari, la qual cosa facilita la col·locació precisa de cada bloc. A la part inferior es veu el panell de Tiles amb el recurs de la natura carregat, demostrant com he anat seleccionant i pintant les diferents peces de terra, plataformes i elements decoratius per construir el nivell de forma modular i organitzada.

- **Z-Index (Ordre de renderitzat):** Aquest valor determina què es dibuixa primer i què queda al darrere. Vaig configurar un valor de 2 per a les pastanagues i el personatge per assegurar que es veiessin sempre per sobre del fons i dels detalls del mapa.

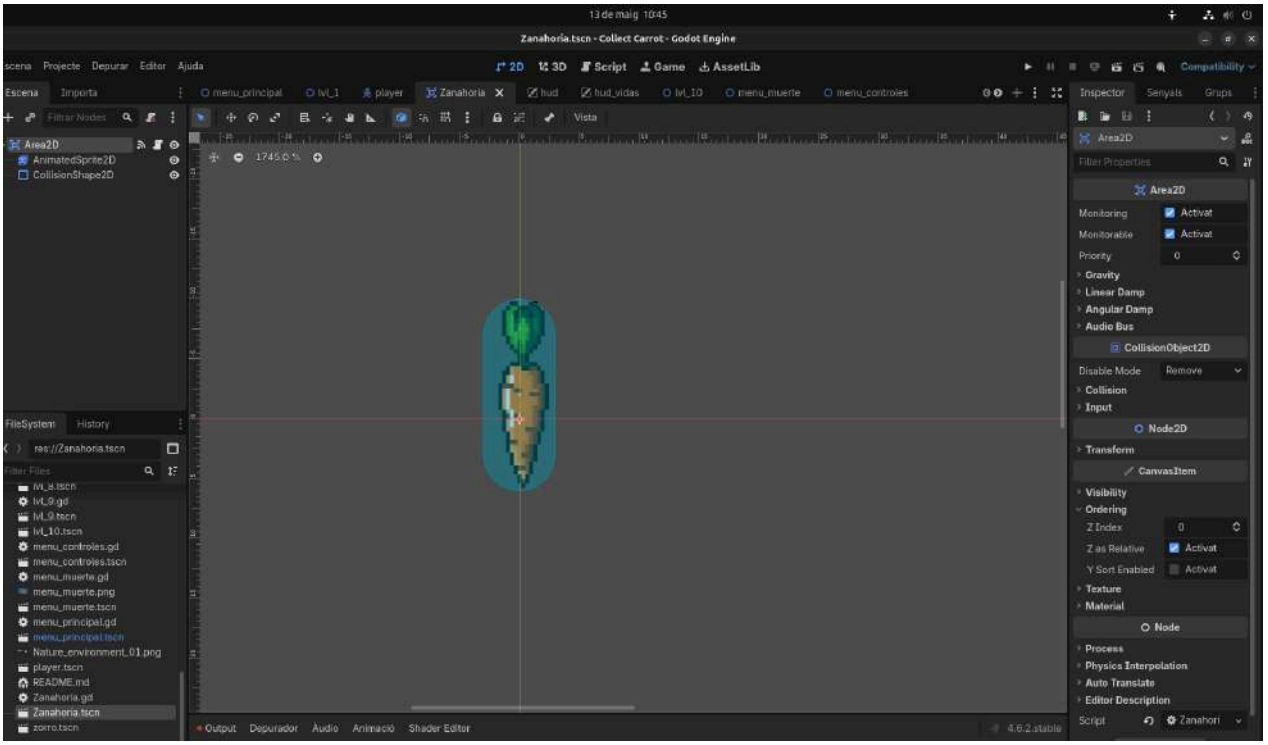


Figura 17: Configuració de les propietats de monitorització i col·lisió del node Area2D de l'objecte pastanaga.

En aquesta captura es pot veure la configuració de l'escena Zanahoria.tscn. A l'inspector de la dreta, dins la secció CanvasItem > Ordering, apareix el paràmetre Z Index. Tot i que a la imatge el valor està a 0 (el valor per defecte), és des d'aquest apartat on el pujo a 2 per garantir que, un cop instanciada al nivell, la pastanaga no quedi mai tapada pel terra o el decorat, facilitant així que el jugador la identifiqui ràpidament.

- **AudioStreamPlayer2D (Gestió de so posicional):** És el node que permet reproduir música i efectes de so dins de l'espai 2D del joc. L'he configurat per carregar fitxers d'àudio en format .wav o .mp3, activant l'opció **Autoplay** perquè la música del nivell comenci a sonar tan bon punt s'inicia la partida.

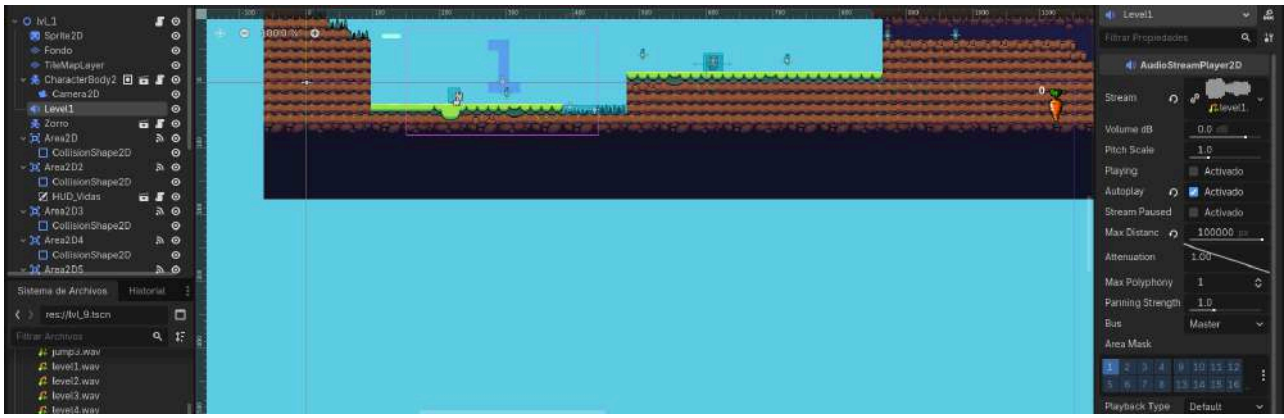


Figura 18: Configuració del node AudioStreamPlayer2D per a la reproducció de la música de fons del primer nivell.

En aquesta captura es pot veure el node AudioStreamPlayer2D seleccionat a l'escena del nivell. A l'inspector de la dreta, s'observa el fitxer de música assignat a la propietat Stream i la casella Autoplay marcada. Gràcies a ser un node 2D, també permet configurar l'atenuació perquè el so s'escolti més fort o més fluix segons la distància de la càmera, tot i que en aquest cas l'he fet servir per a la música ambiental de fons amb un volum constant.

6. Bibliografia

- https://youtu.be/6hj6PwndLJE?si=eN4Rqdgo_D8qPw3m (21-10-2025)
- https://youtu.be/F3T_ZhllzJs?si=VrYS-BsqAF_sqnp- (18-11-2025)
- https://youtu.be/SR7mdh0_i6Q?si=zd6Du1IkTjrzpYkE (18-11-2025)
- https://youtu.be/z_9oVyl1bKk?si=I1re3VExCRvn3Isv (24-11-2025)
- https://youtu.be/eQ_HBvtdoiU?si=4oK5BQTTG2PlwZES (25-11-2025)
- https://youtu.be/RbZtHPRR7fg?si=CXQYnZPhEgy8ydl1 (7-4-2026)

7 Annexos

En aquest capítol s'analitza a fons el procés de creació dels scripts que fan funcionar Collect Carrot. Tota la programació s'ha picat en GDScript des de zero, aprofitant que és un llenguatge orientat a objectes que lliga molt bé amb l'estructura de nodes de Godot. Hem utilitzat molt el sistema de senyals per aconseguir que tot quedi modular, net i fàcil d'ampliar.

7.1. Implementació del Sistema de Moviment i Física del Jugador

El desenvolupament de l'script player.gd el vam començar definint a dalt de tot les variables clau que controlen el moviment del protagonista. Vam decidir utilitzar constants per a la velocitat de córrer i la força del salt; d'aquesta manera podem ajustar la dificultat del joc o fer proves de control en un moment sense haver de buscar i canviar línies de codi per tot el document. El nucli de la física està tancat dins de la funció nativa _physics_process, que és la que recalcula el vector de velocitat a cada fotograma. Perquè el salt tingués una caiguda que es nota real, vam escriure una lògica que va sumant el valor de la gravetat de manera progressiva cada vegada que la funció is_on_floor() detecta que el conill no està trepitjant el terra.

A banda d'ajustar els números de les físiques, ens vam matar a polir el feedback visual i sonor perquè l'usuari s'ho passi bé jugant. Vam muntar un controlador que canvia els estats del node AnimatedSprite2D segons el que estigui fent el personatge en l'eix de les Y. Si el vector vertical és negatiu s'activa el dibuix del salt, i si passa a ser positiu es llança automàticament l'animació de caiguda. Per girar el conill quan camina cap a l'esquerra vam posar una condició molt senzilla que inverteix l'escala horitzontal del node alterant el paràmetre scale.x segons la tecla de direcció premuda. El so de l'impuls el vam lligar exactament al mateix moment de l'execució del salt al codi per donar una resposta auditiva instantània.

purar player.tscn::GDScript_pbfsw

```gdscript
extends CharacterBody2D

const SPEED = 300.0
const JUMP_VELOCITY = -400.0
```

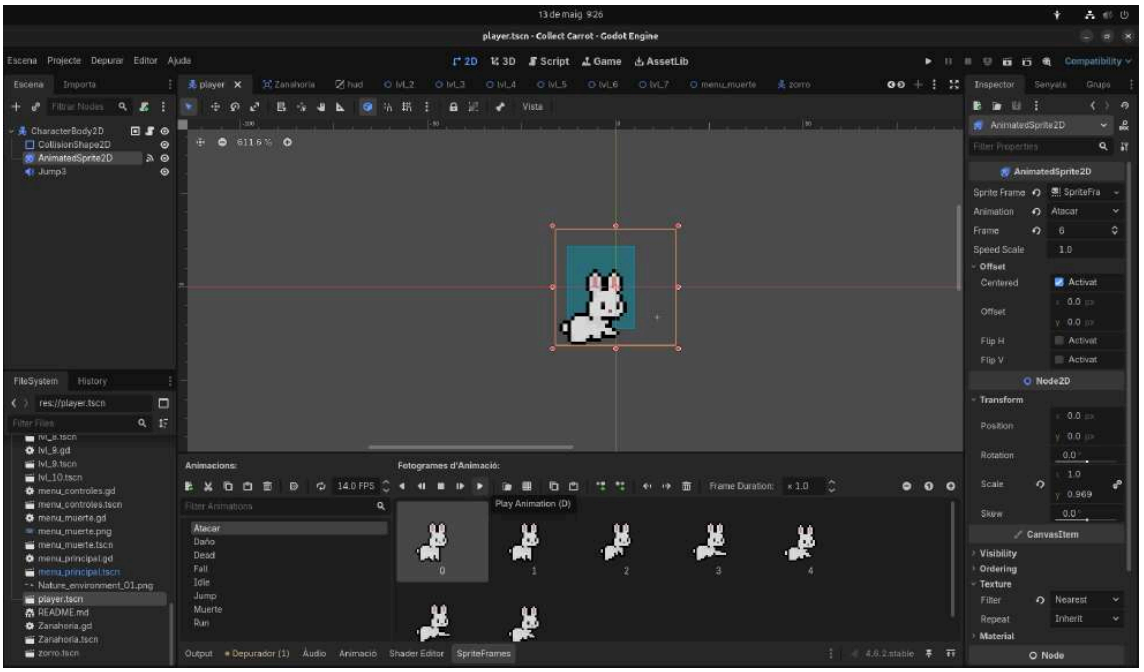
Figura 19: Declaració de constants per a la velocitat de moviment i la força de salt a l'script del jugador.

```
14  func _physics_process(delta: float) -> void:
15      if esta_muerto:
16          velocity.x = 0
17          if not is_on_floor():
18              velocity += get_gravity() * delta
19          move_and_slide()
20          return
21
22      if not is_on_floor():
23          velocity += get_gravity() * delta
```

Figura 20: Fragment de codi de la funció `_physics_process` que controla la gravetat i deté el moviment lateral si el personatge mor.

```
48  if Input.is_action_just_pressed("ui_accept") and is_on_floor():
49      velocity.y = JUMP_VELOCITY
50      $Jump3.play()
51
```

Figura 21: Lògica del salt del personatge en detectar l'acció d'entrada i reproducció de l'efecte de so $Jump3.



7.2. Arquitectura de Dades i Persistència amb el Script Global

Aconseguir que les dades de la partida, sobretot la salut del conill i les pastanagues que portes a sobre, no s'esborressin en canviar de pantalla entre els 10 nivells era un dels punts crítics del projecte. Per solucionar aquest problema de persistència, vam crear un script independent anomenat Global.gd i el vam configurar dins dels ajustos del projecte com un Autoload. Aquesta eina fa que Godot carregui el fitxer a la memòria RAM abans que qualsevol altra pantalla del joc i es quedi allà fix durant tota la partida. Com que les variables viuen en aquest fitxer global, qualsevol enemic o objecte del mapa pot comprovar o editar els valors sense importar en quin nivell es trobi el jugador.

Dins d'aquest mateix fitxer global vam aprofitar per programar tota la gestió del dany centralitzada. En el moment que el conill toca un enemic, l'script d'aquest dolent crida una funció del Global que s'encarrega de restar un cor de la salut i comprovar automàticament si el comptador ha arribat a zero. Si et quedes sense vides, l'Autoload pren el control i carrega directament l'escena de Game Over a la pantalla. Gràcies a aquest plantejament ens hem estalviat haver de repetir codi de mort a cada enemic o a cada racó del mapa, centralitzant les regles de joc en un únic lloc segur.

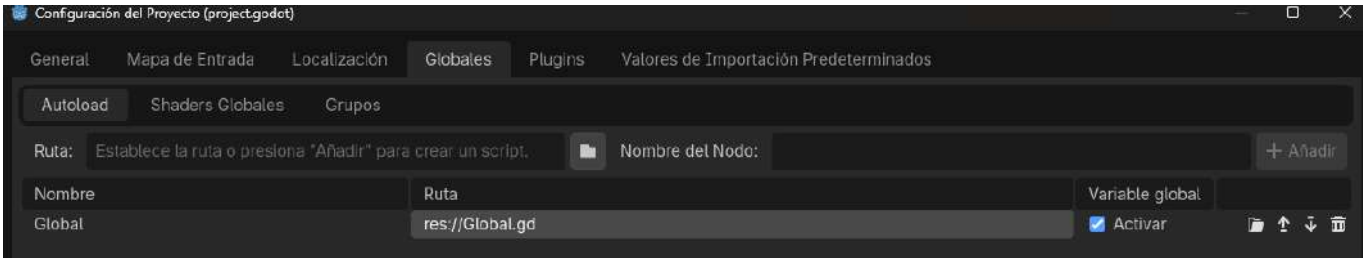


Figura 22: Configuració de l'script Global.gd a la secció d'Autoload de la configuració del projecte a Godot.

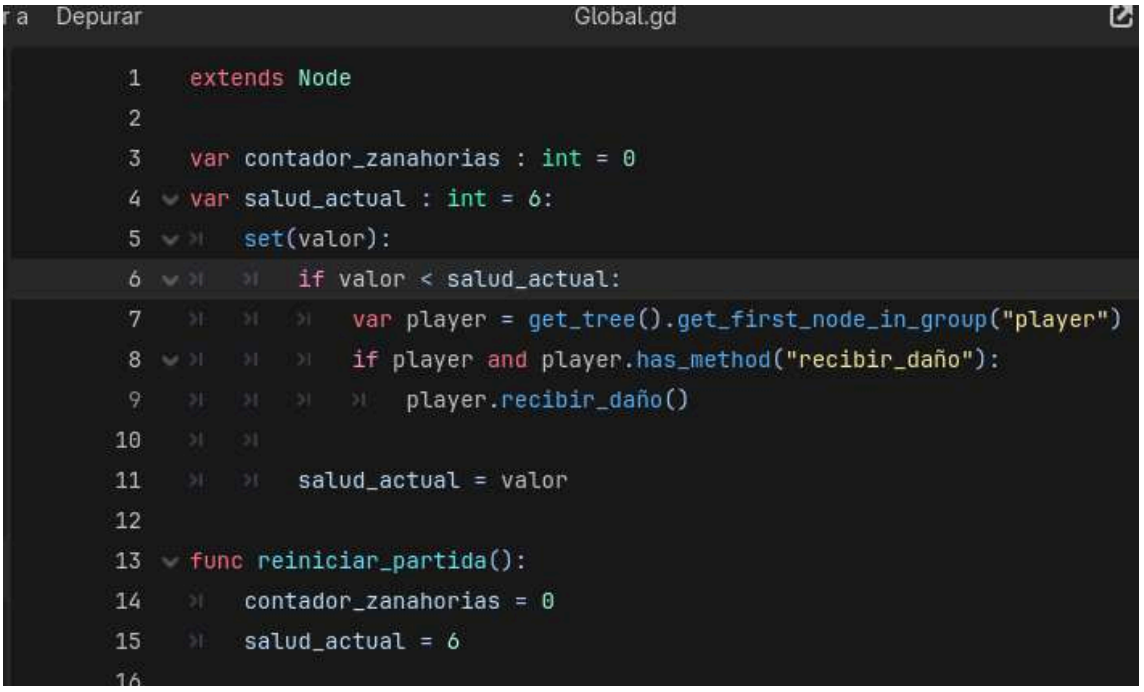
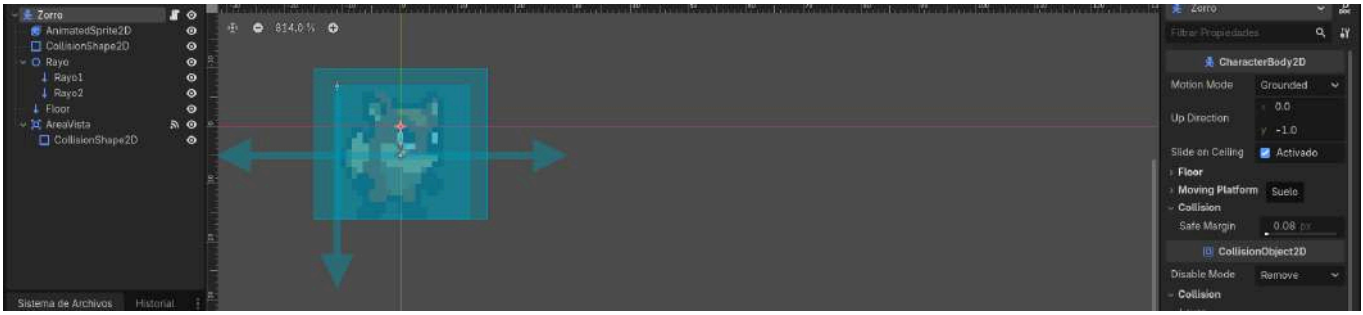


Figura 23: Codi de l'script Global.gd amb les variables globals de salut, comptador i la funció de reinici de partida.

7.3. Intel·ligència Artificial i Patrulla dels Enemics

La programació de la IA de la guineu i del gòblin utilitza vectors de velocitat bàsics i un sistema de detecció anticipada molt pràctic. Per aconseguir que els enemics patrullin sense caure pels marges dels blocs ni quedar-se encallats contra els murs del TileMap, l'script llegeix constantment el que diuen els nodes RayCast2D que tenen acoblats. Tan bon punt el raig col·locat a la part davantera deixa de tocar les col·lisió del terra, s'activa un bloc de codi que multiplica la velocitat actual per -1; això inverteix el sentit del moviment i fa que el sprite de la guineu es giri automàticament cap a l'altre costat.

Amb el gòblin vam voler anar una mica més enllà i li vam posar una capa extra de complicació utilitzant les àrees de detecció. Vam connectar un senyal que s'activa quan el node Area2D detecta que el cos del personatge principal ha entrat dins del seu camp de visió. Quan això passa, l'script de l'enemic posa la seva velocitat a zero a l'instant i comença a reproduir l'animació de l'atac. En el moment que el jugador s'allunya i surt de l'àrea, el gòblin surt d'aquest estat i torna a la seva rutina de patrulla horitzontal normal. Vam haver de fer força proves amb els paràmetres per assegurar que els canvis d'estat no donessin errors rars amb les caixes de col·lisió.




```
76 func _on_area_vista_body_entered(body: Node2D) -> void:
77     if esta_muerto: return
78     if body.is_in_group("player"):
79         esta_atacando = true
80         sprite.play("Atacar")
```

Figura 24: Codi de la funció _on_area_vista_body_entered per activar l'estat d'atac i reproduir l'animació quan el jugador entra en l'àrea.

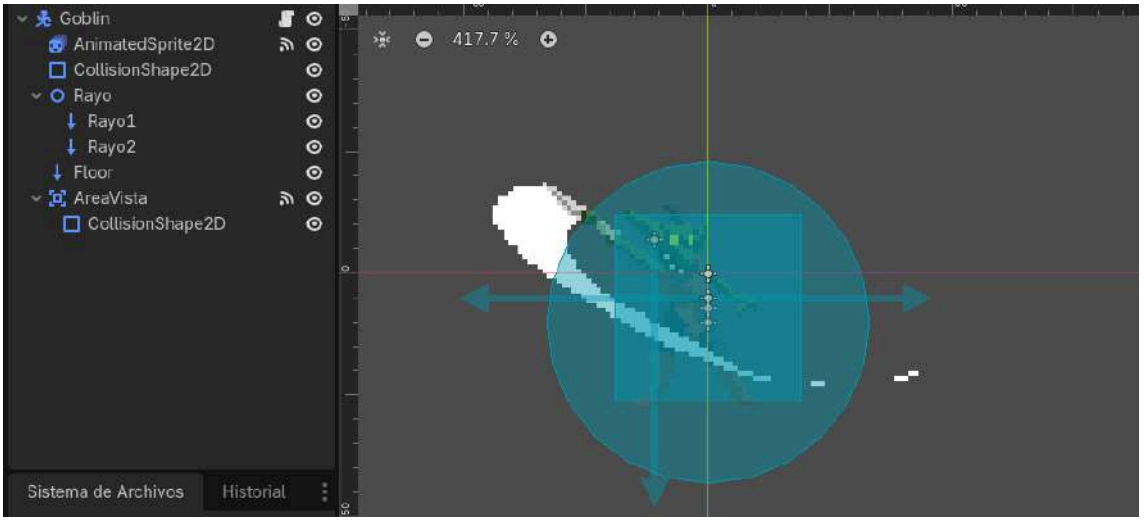


Figura 25: Configuració de l'escena de l'enemic Goblin (Goblin.tscn), els seus nodes de visió i l'àrea de col·lisió circular.

7.4. Mecàniques de Recollida i Gestió de Memòria dels Objectes

L'script de la pastanaga és força senzill i utilitza els senyals natius de Godot per gestionar el contacte amb el jugador. Tota la lògica es dispara mitjançant el senyal body_entered de l'àrea de col·lisió. El primer que fa el codi és un filtre de seguretat amb la funció is_in_group("player") per comprovar que el node que ha entrat a l'àrea és realment el conill i no un enemic que passava per allà. Un cop validat que és el jugador, l'script es connecta amb el fitxer Global.gd per sumar un punt al comptador de la puntuació general que es mostra a la interfície.

Just després de tancar la suma de la puntuació es crida l'ordre queue_free(). Aquesta instrucció és clau per al rendiment de l'ordinador, ja que s'encarrega d'esborrar el node de la pastanaga de la memòria i de l'arbre del joc de forma definitiva. Tenint en compte que hem dissenyat 10 nivells plens de col·leccionables, si no anéssim netejant els elements agafats el joc s'aniria ralentitzant amb el temps. També vam posar un petit efecte gràfic un instant abans de l'alliberament de la memòria perquè agafar l'objecte fos molt més vistós i agradable per a l'usuari.

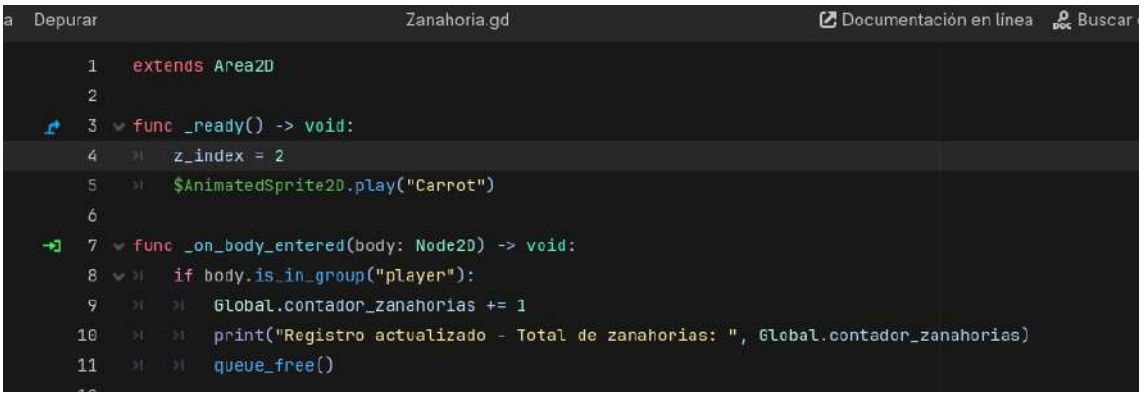


Figura 26: Fragment de l'script Zanahoria.gd amb la lògica completa per incrementar el comptador global i eliminar l'objecte en ser recollit.

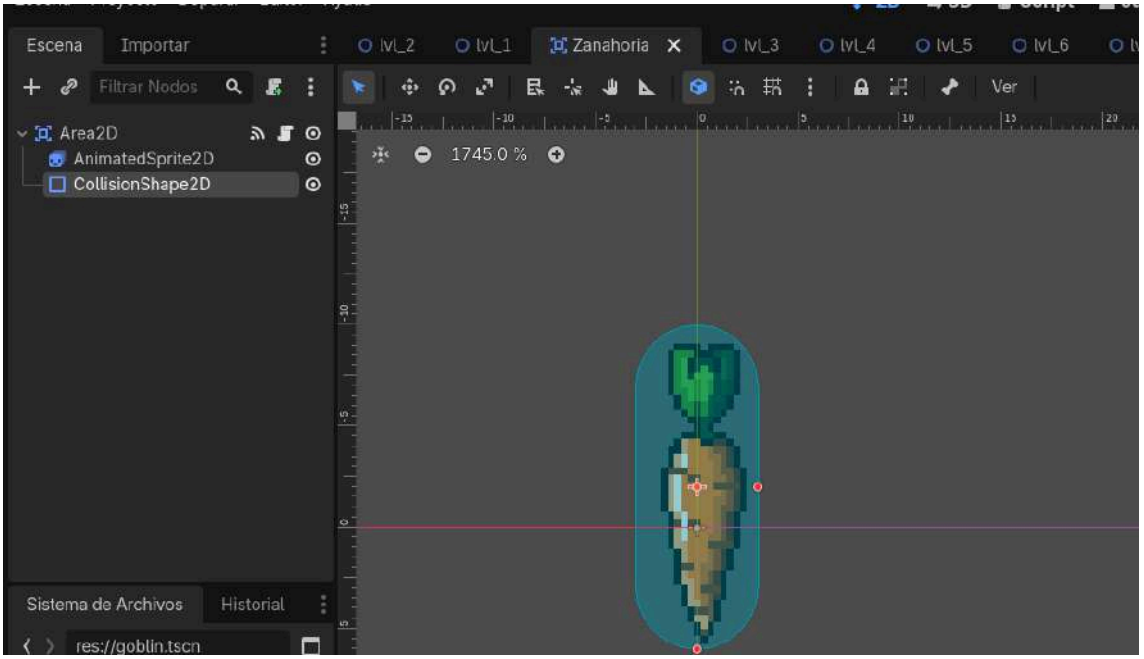


Figura 27: Estructura de nodes de l'escena de la pastanaga dins de l'editor de Godot.

7.5. Programació de la Interfície i Comunicació amb el HUD

L'script que controla la interfície en pantalla serveix per traduir els valors numèrics de les variables globals en components visuals que el jugador pugui entendre. Per al marcador de punts vam programar una funció de refresc que canvia el text del node Label corresponent cada vegada que es recull una pastanaga. Vam haver de vigilar amb la mida i la col·locació del text per assegurar que no sobresortís dels marges de la pantalla ni quedés tapat per altres elements gràfics quan el jugador acumulés molts punts seguits.

El sistema visual dels cors de vida fa servir una lògica similar que revisa l'estat actual de la salut dins del fitxer Global.gd. Vam crear una estructura de control que tria quina textura s'ha d'aplicar a cada espai de vida segons els cops rebuts. Si l'usuari rep mal, l'script modifica directament la propietat texture dels nodes TextureRect de la interfície, canviant els fitxers d'imatge entre un cor ple, mig buit o buit del tot. Aquest mètode és prou obert com per si més endavant volem posar ampolles de salut o objectes per curar-se, ja que el HUD només ha de tornar a passar el bucle de comprovació per pintar els cors plens de nou.

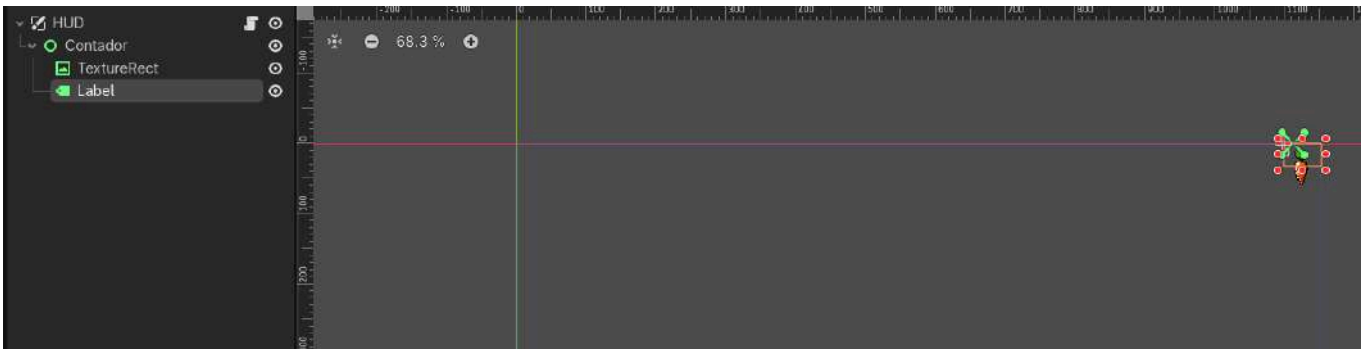


Figura 28: Vista de la interfície de l'editor del HUD a Godot i detall del node Label per al comptador.

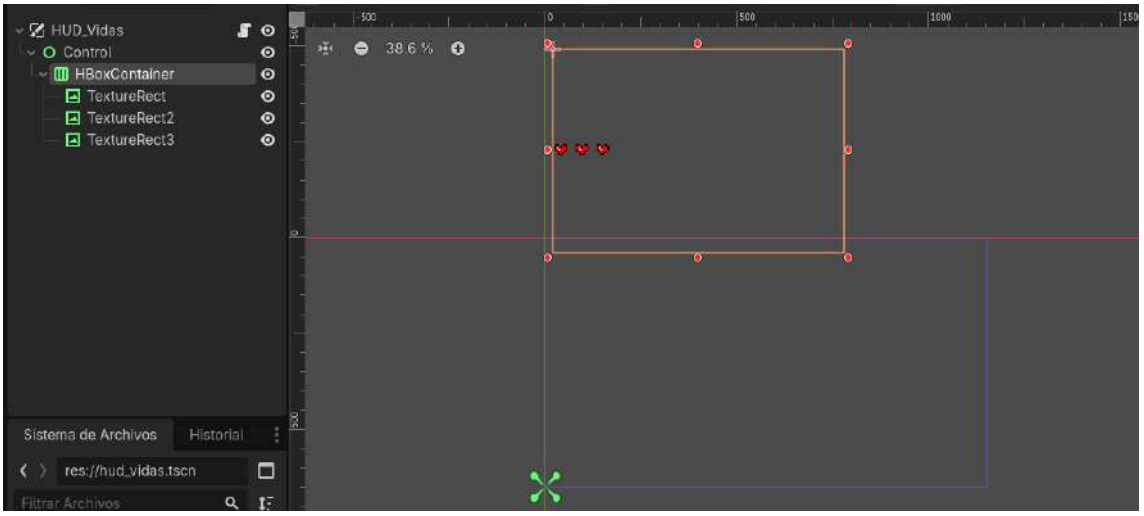


Figura 29: Configuració del contenidor HBoxContainer i els nodes TextureRect de les vides al HUD.

```
14  func _process(_delta):
15      var salud = Global.salud_actual
16
17      for i in range(3):
18          var valor_corazon = salud - (i * 2)
19
20          if valor_corazon >= 2:
21              corazones[i].texture = tex_lleno
22          elif valor_corazon == 1:
23              corazones[i].texture = tex_medio
24          else:
25              corazones[i].texture = tex_vacio
26
```

Figura 30: Codi de la funció _process que actualitza dinàmicament les textures dels corasons segons la salut actual.

7.6. Flux de Transicions als nivells, Menús de Pausa, Victòria i Escenes de Cova

El moviment entre pantalles depèn de les metes que hi ha col·locades a cada mapa. Quan el conill xoca amb l'àrea de la meta, el codi executa el mètode change_scene_to_file() i

salta al següent fitxer d'escena. L'ordre va seguit del lvl_1.tscn al lvl_10.tscn sense trencar-se. El canvi arriba quan et fiques a les coves, del nivell 8 al 10. Allà vam programar un bloc de codi que rebaixa la il·luminació general de la pantalla i canvia els tons del fons. Volíem donar un ambient fosc i més fotut de superar, però ho vam fer sense tocar la programació que ja tenia el conill per moure's o saltar.

Després està el tema del Menú de Pausa. Vam muntar una pantalla que surt al moment si el jugador prem la tecla ESC mentre juga. Això posa la propietat pause de Godot en actiu i tot es queda quiet, enemics inclosos. Dins d'aquest menú vam col·locar tres botons de fusta típics: Continuar per treure la pausa i seguir jugant, Reiniciar nivell per si vols tornar a començar el mapa des del principi, i Salir al menú principal per tancar la partida i ja està.

Per acabar el recorregut del joc està el Menú de Victòria. Aquesta pantalla només s'activa quan el jugador aconsegueix trepitjar la meta de l'últim nivell de tots. El codi detecta que és la fi de la partida, llança un missatge per felicitar l'usuari i posa l'animació del conill content celebrant que ha guanyat. Un cop aquí, l'escena dona dues opcions bàsiques amb botons: Volver al menú principal per si vols tornar a la pantalla d'inici o Salir per tancar el joc completament.

Si la cosa va malament i perds els cors pels enemics, la pantalla de mort serveix de salvavides. El joc obre un menú de caiguda on el botó de Reintentar crida una funció de reset de tota la vida que està a l'script global. Aquesta ordre et torna a omplir la salut a 3 i et col·loca un altre cop a la sortida del mapa inicial. Gràcies a aquest cicle, ens assegurem que el motor físic de Godot esborri la memòria RAM vella i torni a pintar els recursos gràfics des de zero a cada intent.

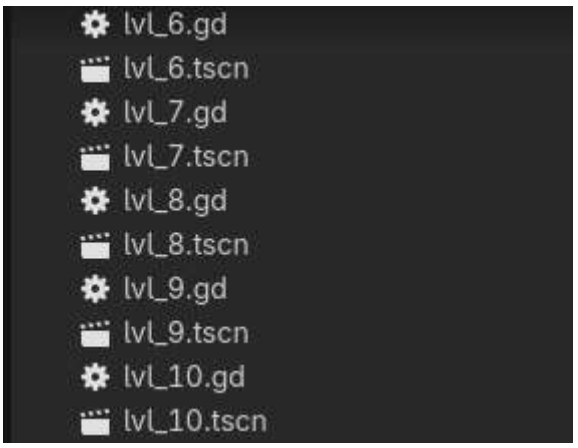


Figura 31: Estructura i llistat de fitxers d'escripts i escenes corresponents als 10 nivells que componen el projecte.

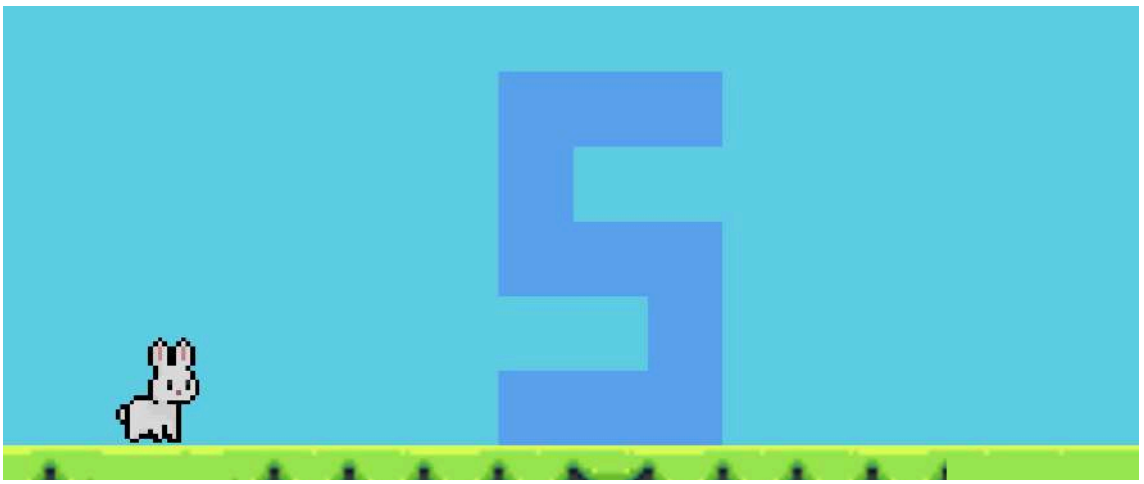


Figura 32: Mostra de l'entorn de joc en un mapa exterior amb rajoles de terra i herba (Nivell 5).

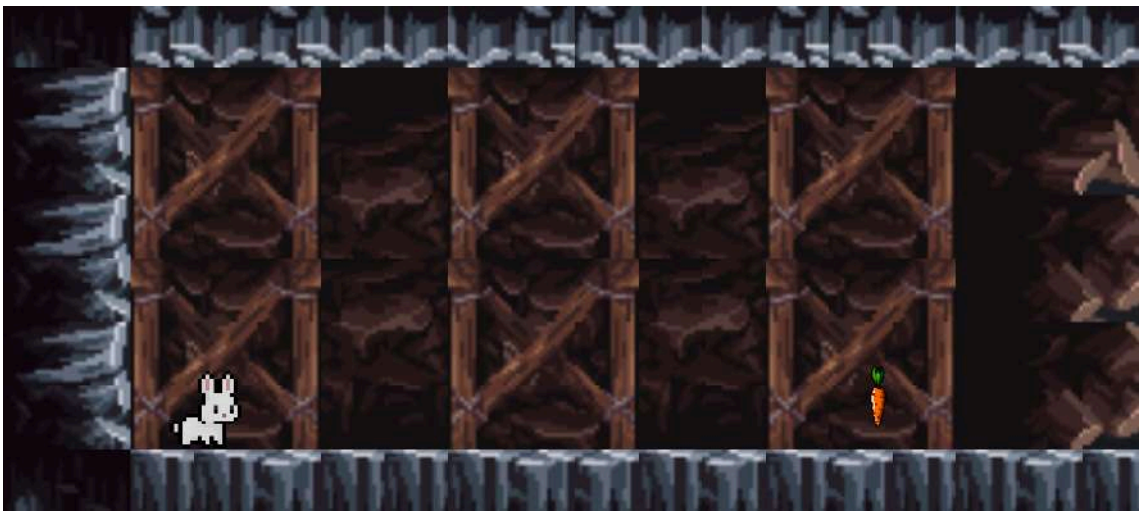


Figura 33: Vista de l'escenari interior ambientat en una cova amb estructures de fusta i obstacles.

Figura 34: Interfície gràfica de la pantalla de Game Over que es mostra en exhaurir-se els punts de vida.

Figura 35: Disseny del menú de pausa dinàmic amb opcions per continuar, reiniciar o sortir al menú principal.

Figura 36: Interfície final de victòria (Has Ganado) que es desplega en completar satisfactòriament tots els nivells.