Universitat de Girona
Escola Politècnica Superior

TREBALL FINAL DE GRAU

Títol: Disseny i desenvolupament d'un videojoc Metroivania de plataformes i acció

Alumne: Roger Gibert Rovira

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Document:	Memòria
Tutor:	Gustavo Ariel Patow
Departament:	Informàtica, matemàtica aplicada i estadística (IMAE)
Àrea:	Llenguatges i sistemes informàtics
Convocatòria (mes/any):	Juny/2022

Agraïments

A Gustavo Patow, tutor del treball, per l'orientació i el suport que m'ha donat des del principi del projecte.

A Arnau Gibert i Oriol Viu, per oferir-se a realitzar les proves externes del projecte que han permès analitzar-lo més detalladament i trobar-hi errors que d'una altra manera haurien passat desapercebuts.

A la meva família i als meus amics, pel suport que m'han donat durant el projecte i per les propostes i idees que sense ells no se m'haurien acudit mai.

Índex

Llistat de figures

Llistat de taules

1. Introducció, motivacions, propòsit i objectius i distribució de tasques

1.1. Introducció

A meitats de la dècada dels 80 va sorgir un nou gènere de videojocs que posteriorment es va anomenar Metroivania o Metroidvania, a causa de la fusió dels noms de dues de les sagues de jocs més importants del gènere (Metroid i Castlevania). Aquest gènere es caracteritza per tenir una perspectiva en dues dimensions, mecàniques de plataformes i combat i fomentar l'exploració minuciosa per mitjà de tornar a visitar zones ja explorades quan el jugador adquireix noves habilitats per desbloquejar noves zones de joc.

Figura 1: Captura Metroid (NES - 1986)

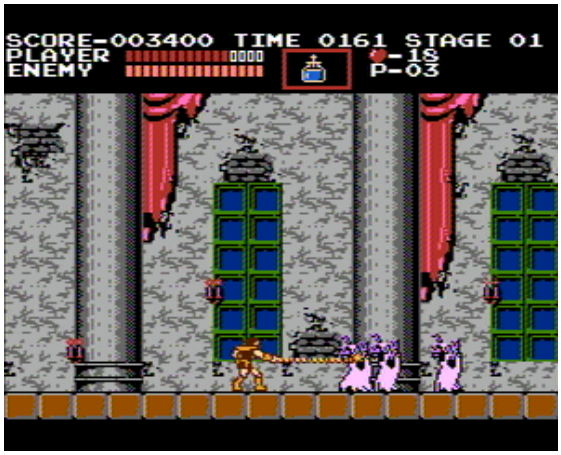

Figura 2: Captura Castlevania (NES - 1986)

Per aquest projecte s'ha decidit desenvolupar un videojoc que implementi les característiques clàssiques d'un Metroidvania. Estarà ambientat en un món de fantasia medieval amb una perspectiva de 2D, tindrà un estil estètic retro píxel art, implementarà mecàniques de plataformes, exploració i combat, estarà compost per un món estructurat en zones interconnectades entre si i una narrativa que el jugador anirà descobrint de manera natural segons avança per les diferents zones.

1.2. Motivacions personals

Les motivacions personals per les quals desenvolupar aquest projecte són les següents:

- Crear un videojoc des de zero englobant tots els àmbits que hi participen
- Adquirir noves habilitats i polir les ja adquirides
- Aplicar els coneixements apresos durant el grau
- Obtenir una visió més professional sobre el desenvolupament de videojocs

1.3. Motivacions del projecte

La principal motivació professional per la qual desenvolupar aquest projecte és omplir un forat que s'ha observat al mercat del gènere Metroidvania. S'han determinat les principals mancances i s'intentaran cobrir creant un videojoc amb les següents caraterístiques:

- Història i ambientacions originals, basades en una època medieval fantàstica
- Estil estètic 2D píxel art retro
- Combat que aposta per les màgies

1.4. Propòsit i objectius

L'objectiu general d'aquest projecte és desenvolupar un videojoc del gènere metroidvania amb mecàniques de plataformes i combat polides i un món interconnectat no lineal ambientat en una època medieval de fantasia.

Els principals objectius del projecte són:

- Aprendre a utilitzar exhaustivament el motor Godot i el seu llenguatge propi GDScript
- Aprendre a utilitzar el software Aseprite i les funcionalitats més rellevants
- Dissenyar per complet la part estètica (personatges, escenaris, objectes i animacions)
- Dissenyar i implementar les mecàniques del jugador
- Implementar la intel·ligència artificial dels enemics
- Desenvolupar les interaccions del jugador amb l'escenari i tots els seus components
- Desenvolupar els menús i interfícies

1.5. Distribució de tasques

Com que el desenvolupament d'aquest videojoc es realitzarà de manera individual, s'ha distribuït l'esforç i el temps que es dedicarà a cada una dels elements que el conformen. A la taula següent (Taula 1) es mostra un percentatge de com s'ha fet la distribució:

Estètica	35%
Narrativa	10%
Mecàniques	30%
Tecnologia	25%

Taula 1: Quadre de distribució de tasques

El camp que rebrà més dedicació serà l'estètica, no perquè tingui més importància sinó perquè la quantitat de temps que es necessita invertir per dissenyar i animar els personatges, objectes, enemics és molt elevat. A més, hi ha una gran quantitat d'elements que caldrà crear, ja que el videojoc està compost només per elements propis.

En quant a mecàniques, s'hi inclou el disseny del moviment i habilitats del jugador, dels diferents enemics i les seves interaccions amb el jugador, dels objectes amb el que el jugador pot interactuar i les diferents zones del mapa que s'implementaran, que inclou la disposició de les sales, dels enemics, de les trampes, etc.

Pel que fa a tecnologia, la tasca principal serà implementar les mecàniques que s'han dissenyat utilitzat el motor Godot i crear els sistemes dins el videojoc que gestionaran tots els elements que hi intervenen.

En el cas d'aquest projecte, la narrativa conforma una part molt rellevant, però no s'explicarà de manera explícita a través de diàlegs o narracions, sinó que es trobarà en gran part integrada en el disseny dels escenaris i dels habitants d'aquests.

2. Estudi de viabilitat

En aquest apartat es valoraran els recursos necessaris per a la realització del projecte, s'analitzarà l'estat actual del mercat per determinar si el videojoc hi té cabuda i es determinarà el públic objectiu.

2.1. Recursos necessaris

2.1.1. Recursos tècnics

Per la major part del desenvolupament del projecte s'ha usat hardware mitjanament potent i software gratuït i lliure, amb una excepció que ja es comentarà. Pel que fa al hardware utilitzat, ja es disposava d'aquest prèviament a l'inici del projecte i les especificacions són simplement orientatives, i no un requisit.

2.1.1.1. Ordinador portàtil

Aquest ha estat l'ordinador principal per al desenvolupament i testeig del projecte. Tot i ser menys potent que el que es comentarà a continuació, s'ha hagut de prioritzar la portabilitat per motiu d'estudis.

Aquestes són les especificacions principals:

- Processador: Intel i5-7300HQ
- Targeta gràfica: Nvidia GTX 1050 Ti
- Memòria RAM: 16GB

2.1.1.2. Ordinador personal

Aquest ordinador s'ha usat principalment pel testeig del projecte, ja que disposa de millors especificacions que l'anterior.

Aquestes són les especificacions principals:

- Processador: Intel i7-4790
- Targeta gràfica: Nvidia GTX 970
- Memòria RAM: 32GB

2.1.1.3. Godot

Godot és un motor de jocs completament gratuït i de codi obert. És menys conegut que Unity o Unreal però disposa d'eines i característiques similars. Disposa de diversos llenguatges de programació: GDScript (llenguatge propi similar a Python), C#, C++, visual scripting (programació visual amb blocs) i d'altres que els proporciona la pròpia comunitat. Per aquest projecte s'ha utilitzat la versió 3.4.2, que era l'última versió estable al començament del projecte.

Figura 3: Logotip Godot

En quant al llenguatge de programació, s'ha escollit usar GDScript per la facilitat d'ús, l'existència de documentació extensa i la familiaritat que ja es tenia de Python.

Altres raons per les quals s'ha escollit Godot han estat la facilitat de crear i editar tilemaps, la possibilitat d'animar qualsevol objecte o característica i la lleugeresa de l'editor, que és un fitxer de mida petita i no necessita instal·lació, la comunitat d'usuaris àmplia i la possibilitat d'exportar el projecte a les principals plataformes (escriptori, mòbil, consoles, web...).

2.1.1.4. Aseprite

Aseprite és una eina per a crear imatges i animacions 2D píxel art. El seu cost va ser de 20€ en el començament del projecte.

Les principals raons per les quals s'ha escollit Aseprite són la simplicitat d'ús, la possibilitat d'animar amb *onion skinning* (permet visualitzar els frames anterior i següent a l'actual) i la facilitat de generar els *sprite sheets* (imatge composta per cada una de les imatges de l'animació).

Figura 4: Logotip Aseprite

2.1.1.5. GitHub Desktop

GitHub Desktop és l'aplicació que simplifica l'ús de GitHub. S'ha utilitzat per mantenir una còpia de seguretat al núvol de tots els fitxers relacionats amb el projecte, per controlar-ne les diverses versions i per simplificar el desenvolupament i testeig en els dos ordinadors.

Figura 5: Logotip GitHub Desktop

2.1.1.6. HacknPlan

HacknPlan és una eina de gestió de projectes enfocada al desenvolupament de videojocs. S'ha utilitzat per definir les tasques i mantenir el control del progrés.

Figura 6: Logotip HacknPlan

2.1.1.7. Pixel Font Converter

Pixel Font Converter és una web que permet convertir una imatge d'una font pixel art a un fitxer ttf. S'ha utilitzat per crear la font d'aquest projecte.

2.1.1.8. Libreoffice

LibreOffice és una suite d'oficina lliure i de codi obert. D'aquest software s'ha utilitzat el Writer per escriure la memòria d'aquest projecte.

Figura 7: Logotip LibreOffice

2.1.1.9. Diagrams.net

Diagrams.net és una web per a dissenyar diagrames. S'ha utilitzat per dibuixar els diagrames d'aquesta memòria.

Figura 8: Logotip Diagrams.net

2.1.1.10. Canva.com

Canva.com és una web per crear diagrames, gràfics i altres tipus de documents. S'ha utilitzat per crear el diagrama de Gantt.

Figura 9: Logotip Canva.com

2.1.2. Recursos humans

Aquest projecte s'ha realitzat de manera individual, però es poden calcular els recursos humans hipotètics que caldrien per desenvolupar-lo de manera professional. S'ha suposat un equip format per un dissenyador, un programador i dos artistes (un s'encarregaria del disseny i l'altre de les animacions) per poder completar el projecte en un termini aproximat de dos anys. Cal comentar que els càlculs anteriors se suposen per a desenvolupar el videojoc complet, però per aquest projecte només se'n desenvoluparà un prototip més petit.

2.1.3. Recursos econòmics

Com que ja es disposava de tots els recursos tècnics necessaris per al desenvolupament del projecte, aquest no suposarà cap cost real. Tot i això, s'ha tingut en compte el cost dels recursos en el moment de la seva compra o descàrrega, que són els següents:

RECURS	COST
Ordinador portàtil	900€
Ordinador personal	1200€
Godot	0€
Aseprite	20€
GitHub Desktop	0€
HacknPlan	0€
LibreOffice	0€
Diagrams.net	0€

Taula 2: Costs del hardware i software necessaris

També s'ha calculat el cost teòric que suposarien els recursos humans ja esmentats, estimant el sou per any que cobrarien (els sous s'han extret la pàgina web *glassdoor.es*):

POSICIÓ	SOU (€/any)
Dissenyador	~35.000€
Programador	~40.000€
Artista	~30.000€

Taula 3: Sous aproximats segons posició

Amb la informació anterior s'ha pogut determinar el cost total aproximat del projecte, tenint en compte que cada treballador disposaria solament d'un ordinador personal similar al definit anteriorment:

CONCEPTE	**COST**
Equips informàtics	1200€ * 4 = 4800€
Dissenyador	35.000€
Programador	40.000€
Artistes	30.000€ * 2 = 60.000€
TOTAL	**139.800€**

Taula 4: Costs totals aproximats

2.2. Estudi de mercat

Un cop s'ha definit que el desenvolupament del projecte és viable, abans de començar cal estudiar l'estat de l'art per detectar els competidors i descobrir com destacar per sobre d'aquests. S'ha definit com a competidors videojocs amb jugabilitat i estètica similars, que tinguin una narrativa mitjanament desenvolupada i un públic objectiu amb característiques semblants.

2.2.1. Metodologia de cerca

Per a realitzar aquesta cerca s'ha usat el motor de cerca Google i s'ha cercat amb les paraules clau següents (s'ha cercat en anglès per obtenir més resultats):

- Metroidvania games
- Metroidvania píxel art
- Metroidvania medieval

També s'ha fet una cerca a la plataforma de venda de videojocs Steam, filtrant per la categoria Metroidvania.

2.2.2. Resultats de cerca

A continuació es comentaran els videojocs que s'han triat per avaluar, els que s'han considerat més rellevants sigui per temàtica, estètica o jugabilitat.

2.2.2.1. Hollow Knight

Figura 10: Captura Hollow Knight

Hollow Knight (2017), és una aventura d'acció 2D ambientada en un món interconnectat. El jugador podrà lluitar contra criatures corrompudes, resoldre misteris antics i fer amistats amb estranys insectes (veure Figura 10).

Les característiques principals són:

- Acció clàssica amb desplaçament lateral
- Controls molt ajustats i precisos
- Món no lineal interconnectat
- Gran quantitat d'enemics i boss finals
- Moltes habilitats i poders desbloquejables

2.2.2.2. Blasphemous

Figura 11: Captura Blasphemous

Blasphemous (2019), és una història que segueix al protagonista dins un món assolat per una terrible maledicció i com lluita per obrir-se pas i aconseguir acabar amb la maledicció (veure Figura 11).

Les característiques principals són:

- Món no lineal interconnectat
- Combats intensos amb diversitat de combinacions
- Personalització del personatge amb diferents habilitats i poders
- Lluites amb boss finals molt intenses

2.2.2.3. Salt and Sanctuary

Figura 12: Captura Salt and Sanctuary

Salt and Sanctuary (2016), és una història d'un mariner que naufraga a una illa no explorada que haurà d'explorar per descobrir-ne els secrets (veure Figura 12).

Les característiques principals són:

- Combats ràpids i amb molta acció
- Món no lineal interconnectat
- Gran quantitat d'armes, habilitats i armadures
- Agafa mecàniques del gènere RPG (elaboració d'objectes)

2.2.2.4. Have a nice death

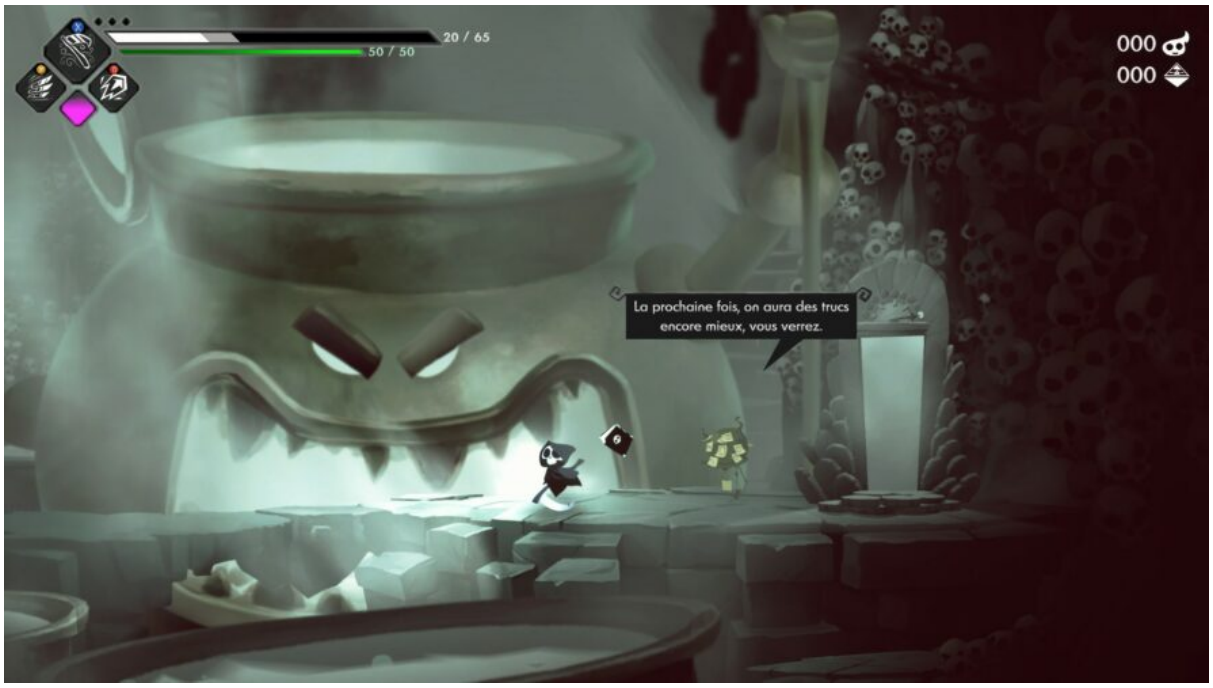

Figura 13: Captura Have a nice death

Have a nice death (2022), és una història que segueix a la Mort a l'oficina, on haurà de tornar l'ordre a l'equilibri de les ànimes que ha estat pertorbat pels seus empleats (veure Figura 13).

Es troba en accés anticipat, però s'ha considerat que la seva versió actual també és rellevant per a la comparació.

Les característiques principals són:

- Gran quantitat d'habilitats, armes i objectes desbloquejables
- Món no lineal interconnectat
- Diversitat d'enemics i boss finals
- Tocs d'humor

2.2.3. Comparació

Els criteris que s'han escollit per a comparar són:

- Ambientació: on transcorre la història
- Estil estètic: tècnica usada per crear l'art
- Duració: mitjana orientativa, en aquest gènere l'exploració i el col·leccionisme tenen un paper important i poden allargar-la bastant.
- Linealitat: del món i dels escenaris, diversitat de rutes possibles per avançar.

NOM	AMBIENTACIÓ	ESTIL ESTÈTIC	DURACIÓ	LINEALITAT
Hollow Knight	Món fantasia	Dibuix a mà	40h	Món obert interconnectat
Blasphemous	Món fantasia, època medieval	Pixel art	18h	Món obert interconnectat
Salt and Sanctuary	Illa no explorada, època medieval	Basat en sprites	21h	Món obert interconnectat
Have a nice death	Oficina paranormal de la Mort	Dibuix a mà	12h	Món obert interconnectat
Awaken	**Món fantasia, època medieval**	**Pixel art**	**5h**	**Món obert interconnectat**

Taula 5: Taula de comparació

2.2.4. Conclusions

Després d'analitzar l'estat de l'art, podem concloure que tenim una proposta original tot i tenir similituds amb la resta del mercat. Les principals característiques que ens diferencien són:

- Història i ambientacions originals, tot i estar basat en una època medieval fantàstica, el món que s'ha creat és completament diferent.
- Estil estètic píxel art retro, s'ha creat un estil amb sprites i animacions més simples i paletes de colors limitades, que li donen aquest toc retro.
- El combat fusiona atacs físics, però s'aposta més per les màgies per ampliar la complexitat i la varietat del combat.
- També s'ha apostat per combats menys intensos i freqüents i per intercalar puzles i exploració dels escenaris per aconseguir un videojoc amb acció però més relaxat.

2.3. Públic objectiu

Per determinar el perfil del nostre públic objectiu s'ha agafat com a referència la taxonomia de Richard Bartle, que classifica els jugadors en quatre grups segons les accions que més els agrada realitzar (veure Figura 14). Els grups són els següents:

- **Triomfador (Achiever):** Els interessa aconseguir nivells, equipament, assoliments… i completar reptes del videojoc o propis que s'inventen per demostrar el seu èxit dins el videojoc. Requereixen novetats constants.
- **Explorador (Explorer):** Els interessa descobrir noves zones i secrets amagats i explorar tot el que els ofereix el videojoc. Els agrada explorar al seu ritme sense limitacions de temps. Requereixen novetats i evolucions.
- **Socialitzador (Socializer):** Els interessa interaccionar amb altres jugadors, utilitzen el videojoc com a eina per conèixer nous jugadors. Requereixen una comunitat àmplia i activa.
- **Competidors (Killer):** Els interessa la competició amb altres jugadors abans que amb els enemics programats del videojoc, volen dominar el videojoc i als jugadors que el juguen.

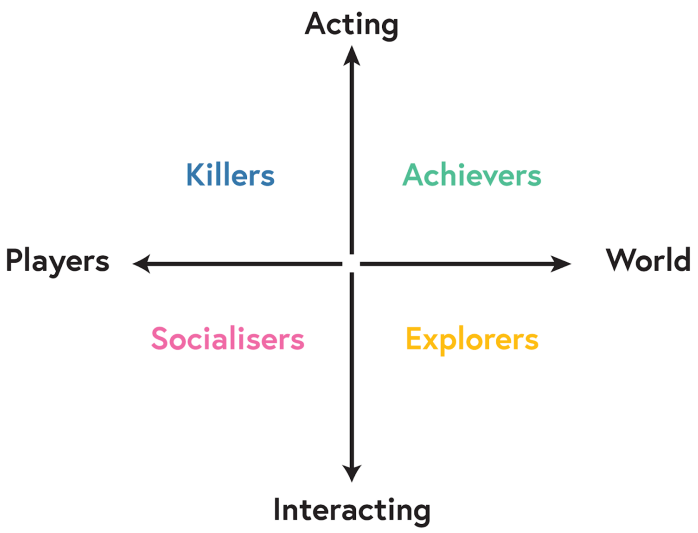

Figura 14: Taxonomia de Richard Bartle

A partir d'aquesta classificació s'ha observat que el perfil que encaixaria més amb el projecte seria una barreja entre **Explorador** (60%) i **Triomfador** (40%). Els jugadors els interessa explorar la totalitat del mapa per descobrir tots els detalls, col·leccionables i easter eggs i poder aconseguir tots els poders i habilitats disponibles, però també poder derrotar a tots els enemics possibles i completar els diferents puzles.

El rang d'edat es trobaria entre joves-adults i adults, ja que és un videojoc al qual se li ha de dedicar forces hores per completar-lo i conté una narrativa que cal anar seguint. També l'estètica píxel art i la jugabilitat s'inspiren en videojocs més retro i un públic més jove no se sent tan atret per aquest estil de videojocs.

3. Planificació

En aquest apartat es definiran el pla de treball a seguir i les tasques que cal realitzar per aconseguir l'objectiu proposat, juntament amb la temporalització.

3.1. Metodologia de treball

Per a desenvolupar qualsevol videojoc, existeixen tres fases bàsiques: el disseny, la implementació i el testeig. Aquestes fases inicialment es realitzen per ordre, però a mesura que avança el projecte es transformen en un procés iteratiu, ja que sovint cal modificar el disseny i/o la implementació d'algun element o elements.

Per al nostre projecte es comentaran breument que comporta cada una de les fases:

- **Disseny:** Diferenciarem les dues etapes de la fase de disseny. Primer, farem un disseny general per definir els conceptes bàsics dels quals partirem, i després, farem el disseny individual per a cada element concret.

- **Implementació:** Diferenciarem les dues vessants: la implementació de mecàniques i la implementació de l'estètica. La implementació de mecàniques és la programació d'aquestes dins el motor de jocs. La implementació de l'estètica és la creació dels diferents elements que componen un personatge, objecte, enemic... i d'entre altres són crear l'sprite inicial, fer-ne les animacions, afegir efectes...

- **Testeig:** Quan parlem de testeig ens referim la fase que està composta per dos processos que es realitzen en ordre: el testeig i la correcció d'errors.
  En el testeig, es comprova el funcionament de les mecàniques implementades. Aquest procés pot tenir tres resultats diferents: l'element funciona correctament i no cal modificar-ne res, l'element funciona correctament però cal modificar algun valor per ajustar el seu comportament o l'element no funciona i per tant cal corregir l'error.
  En la correcció d'errors, és quan es modifiquen els valors per ajustar el comportament o se solucionen els problemes perquè l'element tingui un funcionament correcte. Llavors és quan iterem el disseny o la implementació segons el cas.

A l'esquema de la Figura 15, es mostra com s'ha aplicat la metodologia.

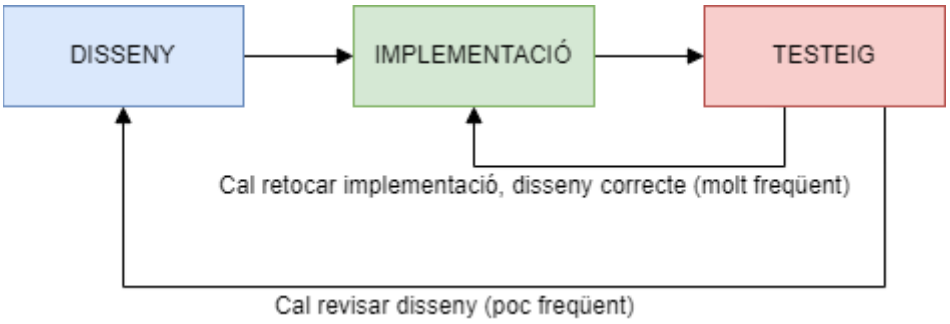

Figura 15: Esquema de la metodologia de treball

3.2. Pla de treball

S'han organitzat les tasques i subtasques que caldrà realitzar en aquest projecte en set categories: disseny, estètica, mecàniques, tecnologia, narrativa, integració i testeig i documentació. Una tasca pot estar formada per subtasques de diferents categories, però a l'hora de classificar-la, l'hem inclòs en la categoria més rellevant. Al diagrama de la Figura 16, es pot observar aquesta classificació:

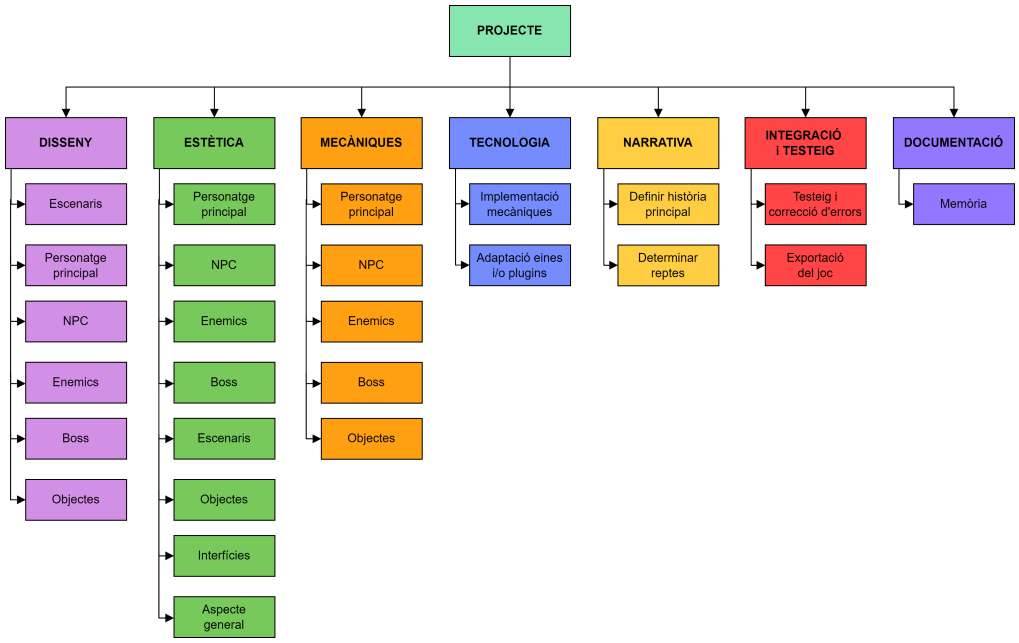

Figura 16: Divisió de les tasques

3.3. Tasques planificades

En aquest apartat, es farà una descripció de cada una de les tasques mencionades al diagrama de la Figura 16, juntament amb una estimació del temps necessari per implementar-la. Els temps estimats no impliquen que només s'avanci en aquella tasca i tenen en compte les diferents iteracions que calgui realitzar.

3.3.1. Disseny

- **Escenaris (2 mesos):** dissenyar l'ambientació de les zones, la distribució general del mapa, la distribució específica de cada sala i la posició de les trampes, dels objectes, dels obstacles i dels enemics dins les sales.
- **Personatge principal (1.5 mesos):** dissenyar les característiques, els atributs i les habilitats, les interaccions amb els escenaris, els objectes, els NPC i els enemics.
- **NPC (0.5 mesos):** dissenyar les interaccions amb el jugador i l'escenari.
- **Enemics (5 mesos):** dissenyar els atacs, les interaccions amb el jugador i l'escenari.
- **Boss (2 mesos):** dissenyar els diferents atacs, les interaccions amb el jugador i l'escenari.
- **Objectes (3 mesos):** dissenyar les interaccions amb el jugador i l'escenari.

3.3.2. Estètica

- **Personatge principal (1.5 mesos):** dissenyar l'aspecte, les animacions, crear els sprites i l'sprite sheet final.
- **NPC (0.5 mesos):** dissenyar l'aspecte i crear l'sprite.
- **Enemics (4.5 mesos):** dissenyar l'aspecte, les animacions, crear els sprites i l'sprite sheet final.
- **Boss (2.5 mesos):** dissenyar l'aspecte, les animacions, crear els sprites i l'sprite sheet final.
- **Escenaris (5 mesos):** dissenyar l'aspecte, crear els tiles, els tilesets, el tilemap i les decoracions.
- **Objectes (3 mesos):** crear l'sprite i les animacions.
- **Interfícies (3.5 mesos):** crear el HUD i els menús inicial i del jugador durant la partida.
- **Aspecte general (7.5 mesos):** crear la font pels textos, comprovar la coherència estètica dels diferents elements i les paletes de colors.

3.3.3. Mecàniques

- **Personatge principal (1.75 mesos):** determinar com s'aplicaran els dissenys creats anteriorment, definir els valors dels atributs, de les habilitats i de les interaccions amb els enemics, l'escenari i els objectes.
- **NPC (1 mes):** determinar com s'aplicaran els dissenys creats anteriorment, definir els valors dels atributs i de les interaccions amb el personatge principal i els objectes.
- **Enemics (6 mesos):** determinar com s'aplicaran els dissenys creats anteriorment, definir els valors dels atributs i de les interaccions amb els enemics i l'escenari.
- **Boss (3 mesos):** determinar com s'aplicaran els dissenys creats anteriorment, definir els valors dels atributs i de les interaccions amb els enemics i l'escenari.
- **Objectes (3.5 mesos):** determinar com s'aplicaran els dissenys creats anteriorment, definir els valors dels atributs i de les interaccions amb el personatge principal, els enemics i l'escenari.

3.3.4. Tecnologia

- **Implementació mecàniques (7.5 mesos):** implementar les mecàniques al motor Godot.
- **Adaptació eines i/o plugins (3 mesos):** adaptar els *plugins* per crear els sistemes que simplificaran el procés d'implementació i programació.

3.3.5. Narrativa

- **Definir història principal (2 mesos):** crear la història principal, l'ambientació del món i el rerefons dels personatges.
- **Determinar reptes (2 mesos):** definir els reptes i les tasques que haurà de superar el personatge principal.

3.3.6. Integració i Testeig

- **Testeig i correcció d'errors (8 mesos):** testejar el disseny, l'estètica, les mecàniques i les implementacions de cada element i iterar amb els resultats per anar polint cada part.
- **Exportació del joc (1 mes):** exportar el videojoc i comprovar que el funcionament esperat i provat dins el motor Godot és el mateix al qual s'ha exportat.

3.3.7. Documentació

- **Memòria (9.5 mesos):** escriure la memòria del projecte

3.4. Diagrama de Gantt

En el següent cronograma (Figura 17) s'observa detalladament la temporalització de cada una de les tasques segons els mesos que ha durat el projecte:

Figura 17: Diagrama de Gantt

4. Marc de treball i conceptes previs

Abans d'aprofundir més en aquest projecte, cal definir una sèrie de conceptes habituals del món dels videojocs o de les eines que s'han utilitzat, que ens permetran comprendre completament el que s'explicarà en els propers apartats.

4.1. Conceptes de videojocs i metroidvanias

- **Boss:** qualsevol enemic que es troba al final d'un nivell, fase o zona. És més difícil de derrotar que la resta d'enemics. Sol tenir patrons d'atac únics, punts febles i una banda sonora pròpia.
- **Boss fight:** lliuta contra qualsevol *Boss*. Se sol obtenir alguna recompensa o objecte important per avançar en la història.

A la figura 18 es pot observar una *boss fight*, entre *The Knight*, personatge de l'esquerra, i *Soul Master*, personatge de la dreta.

Figura 18: Captura Hollow Knight

- **NPC:** de les sigles en anglès Non-Player Character, és un personatge controlat pel propi joc que acostuma a ajudar, guiar o interactuar amb el jugador de manera no violenta. Alguna vegada un NPC pot convertir-se en enemic.

- **Sprite:** qualsevol imatge 2D que es presenta per pantalla. Alguns exemples serien un personatge, un enemic o un objecte.
- **Sprite sheet:** imatge formada per diferents imatges ordenades en una graella. Permet al motor obtenir diversos *sprites* per a generar animacions d'un element o per a representar diferents objectes només havent de carregar una imatge. Podem veure un exemple d'un sprite sheet complet a la Figura 19.

Figura 19: Sprite sheet protagonista de Metroid Fusion

- **Backtracking:** en el context dels videojocs, consisteix en tornar a zones o nivells ja superats amb l'objectiu d'explorar àrees o obtenir objectes desbloquejats que abans eren inaccessibles.
- **Checkpoint:** lloc o moment on es guarda la partida, des d'on, en cas que el personatge mori o sigui eliminat, es reiniciarà la partida.
- **Respawn:** lloc del joc on els enemics ja eliminats poden reaparèixer, ja sigui al cap d'un temps de ser eliminats o quan el jugador mor i reapareix. També seria aplicable al jugador quan reapareix a un checkpoint després de morir.

- **HUD:** de les sigles en anglès Head-Up Display, engloba tota la informació que es mostra per pantalla durant la partida, sovint en forma d'icones o nombres. Alguns exemples serien el nombre de vides o la quantitat d'energia restants. A la figura 20, es mostra un exemple.

Figura 20: Captura Have a nice death

- **Atributs:** capacitats bàsiques d'un personatge, que poden ser innates o haver-se desbloquejat, que defineixen les seves característiques físiques o mentals. En alguns casos poden millorar-se obtenint certs objectes durant la partida. Alguns exemples serien la velocitat, la resistència, la potència de salt...

- **Habilitats:** capacitat del personatge a realitzar una acció, que se sol tenir des de l'inici de la partida, si no que s'ha d'obtenir durant aquesta. Poden ser millorades durant la partida. Alguns exemples serien obrir candaus, resistència al verí o al foc...

  Atributs i Habilitats són conceptes similars, tot i que es poden diferenciar en el fet que els atributs són capacitats més generals i les habilitats són més concretes. En alguns jocs els atributs inicials del personatge determinen quines habilitats es podran adquirir durant la partida.

- **Mana:** font d'energia que s'utilitza com a recurs consumible quan usa una habilitat especial o màgica. També la podem trobar anomenada com a màgia o energia, entre d'altres.

4.2. Conceptes de Godot

A continuació es comentaran alguns termes específics relacionats amb el motor Godot:

- **Escena:** parts en les quals es pot desglossar el videojoc. Poden ser des d'un personatge, una arma, un menú, un nivell... Es poden inserir escenes dins escenes, per exemple, podem tenir un personatge dins una casa dins un nivell. En altres motors poden anomenar-se *prefabs* o escenes també.
- **Node:** elements que formen part d'una escena, són els elements més bàsics. Un personatge pot tenir de nodes un *sprite*, una càmera i un *collider*. El mateix motor Godot proporciona una gran quantitat de nodes que són suficients per a crear pràcticament qualsevol videojoc i també dóna llibertat per personalitzar-los. En altres motors poden anomenar-se classes.

Un exemple d'escenes i nodes seria el següent. Casa és l'escena actual. Dins aquesta escena trobem dues escenes instanciades, que són Persona i Televisió. Persona i Televisió poden tenir altres nodes. També trobem dos nodes, que són Camera2D i Sprite. Aquesta escena Casa podria ser instanciada diverses vegades en una altra escena que es digues Poble. Veure Figures 21 i 22.

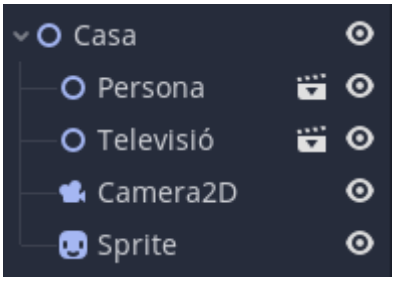

Figura 21: Nodes de Casa

Figura 22: Nodes de Poble

- **Signal:** o senyal, l'emet un node quan succeeix un event determinat prèviament. Permet als nodes comunicar-se amb més flexibilitat.
- **Collider:** node que pot tenir diferents formes que s'utilitza per detectar les col·lisions entre nodes.
- **Tile:** element que conté un *sprite* i opcionalment *colliders*.
- **Tileset:** llibreria formada per un conjunt de *tiles*.
- **Tilemap:** malla regular formada per diferents *tiles* que s'utilitza per determinar la disposició del nivell. Cada *tilemap* pot utilitzar un o diversos *tilesets*.

- **Collision layer:** defineix en quins *layers* es troba aquell node. El *layer* determina de quin tipus és el node.
- **Collision Mask:** defineix en quines *masks* es troba aquell node. La *mask* determina quins tipus d'objectes detectarà el node.

Un exemple de *collision layer* i *collision mask* seria el següent. El jugador estarà al *collision layer* 2, que s'anomenarà *Player.* El terra i les parets del nivell estaran al *collision layer* 1, que s'anomenarà *Map*. Els enemics estaran al *collision layer* 3, que s'anomenarà *Enemies*. Les monedes estaran al *collision layer* 4, que s'anomenarà *Coins.*

Les interaccions que volem que succeeixin són: jugador amb terra i parets, enemics amb terra i parets, jugador amb enemics, enemics amb jugador i jugador amb monedes. Llavors, segons la classificació anterior, haurem de definir les *collision masks* del jugador i dels enemics: Jugador tindrà les *collisions masks* 1 (terra i parets), 3 (enemics) i 4 (monedes). Enemics tindran les *collisions masks* 1 (terra i parets) i 2 (jugador).

- **Hitbox i Hurtbox:** Aquests dos nodes s'utilitzen per detectar col·lisions durant el combat. Les *hitboxes* actuen com a generadors de dany i les *hurtboxes* actuen com a receptors de dany. Per exemple, el jugador tindrà una *hitbox* que s'activarà quan realitzi un atac. També tindrà una *hurtbox* que detectarà quan un enemic l'ha colpejat en aquella zona. Les *hitboxes* i *hurtboxes* solen tenir *collision layers* i *collision masks* propis per simplificar les interaccions entre elles. Alguns exemples podrien ser *PlayerHitbox*, *PlayerHurtbox* o *EnemyHurtbox.*

4.3. Conceptes de la màquina d'estats

Per implementar la màquina d'estats que servirà de base per a tots els personatges que la necessitin, s'ha adaptat la implementació d'una màquina d'estats que forma part d'un projecte de GitHub anomenat GDQuest godot-demos. S'ha usat herència per a dissenyar els estats i la pròpia màquina. Tots els estats i màquines compartiran una estructura bàsica, que després es podrà ampliar o modificar, segons el cas.

En el cas dels estats, disposen d'un senyal (*signal finished*), que serveix per indicar que aquell estat ha acabat. També disposa de cinc funcions principals: *enter*, on s'inicialitzen les variables; *exit*, on es fa neteja de les variables o nodes (s'aturen els timers, es reinicialitzen variables, etc.); *handle_input*, on es gestionen els events d'entrada que realitza el jugador; *update,* és el *loop* de l'estat; i *_on_animation_finished,* per gestionar quan ha acabat una animació de l'estat. Veure Figura 23.

```
1  extends Node
2
3  signal finished(next_state_name)
4
5  # Initialize the state. E.g. change the animation
6  func enter():
7      return
8
9  # Clean up the state. Reinitialize values like a timer
10 func exit():
11     return
12
13 func handle_input(event):
14     return
15
16 func update(delta):
17     return
18
19 func _on_animation_finished(anim_name):
20     return
```

Figura 23: Codi de State

5. Disseny del videojoc

5.1. Pla de màrqueting

En aquest apartat es considera que el projecte s'ha realitzat per complet amb els costos suposats a l'apartat 2.1.3. Recursos econòmics, on s'ha determinat un cost de 139.800€.

5.1.1. Plataformes de distribució i consum

Per a determinar el preu de venda, s'ha tingut en consideració l'esforç i el cost de desenvolupar el projecte i la duració final del videojoc, que seran unes cinc hores. Tenint en compte aquests factors, s'ha decidit que el preu de venda serà de 10€.

Per distribuir el videojoc s'utilitzarà la plataforma *Steam.* S'ha escollit aquesta plataforma perquè és la més popular en quant a distribució i nombre d'usuaris actius. Per publicar el videojoc en aquesta plataforma, s'ha de pagar una tarifa de 100€ i per cada unitat venuda, *Steam* es queda un 30% dels beneficis.

Si el cost de produir el videojoc ha estat de 139.800€ i es ven a 10€, caldrà vendre unes 20.000 unitats per a recuperar l'inversió inicial.

5.1.2. Comunicació i publicitat

Per a la publicitat s'utilitzaran dues vies, que són:

- Anuncis a xarxes socials
- Lliurament de còpies gratuïtes a revistes especialitzades i a canals de videojocs de plataformes de contingut, com ara *Twitch* o *YouTube.*

L'objectiu d'aquesta publicitat és arribar al nombre més gran de jugadors possible que es troben dins el grup del nostre públic objectiu, i les dues vies són els canals que aquest tipus de jugadors consumeix en més quantitat.

5.2. Mecàniques

5.2.1. Jerarquia de reptes

El repte principal del videojoc és derrotar al *Boss* final. Per poder superar aquest repte primer s'han de completar els reptes de totes les zones, que tenen la mateixa estructura per cada una. L'objectiu final d'una zona és derrotar un *Boss*. Per poder derrotar-lo, cal superar un repte obligatòriament, obtenir els *upgrades* d'aquella zona, ja que sense els *upgrades* serà impossible accedir a la sala on es troba el *Boss*. Els altres reptes són opcionals, però completar-los facilita l'aprenentatge i el domini de les mecàniques desbloquejades en aquella zona i simplifica la lluita contra el *Boss*, ja que al disposar de més habilitats, es té més varietat d'estratègies per afrontar la lluita. Veure Figura 24.

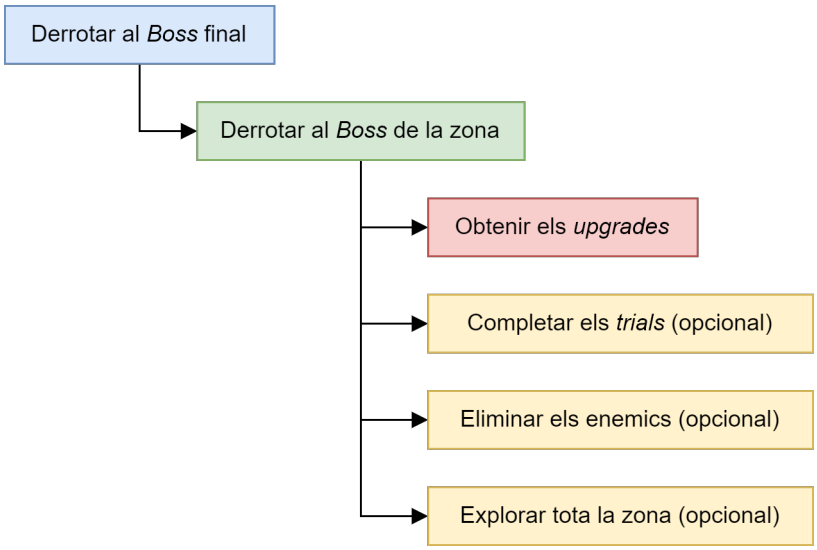

Figura 24: Jerarquia de reptes

5.2.2. Accions del jugador

El jugador pot realitzar un seguit d'accions per a completar els reptes, algunes d'aquestes accions estaran desbloquejades des de l'inici de la partida i d'altres s'aniran desbloquejant segons s'obtenen *upgrades* i *spells.* El llistat complet d'accions és el següent:

Desbloquejades des de l'inici:

- **Moviment:** desplaçar-se a la dreta, a l'esquerra o caure.
- **Saltar:** desplaçar-se cap amunt. Només es pot realitzar si s'està en contacte amb el terra.
- **Atacar:** realitzar un atac que danya als enemics. Es pot executar cap als costats, cap a dalt o cap a baix. Té un temps de recàrrega molt baix.
- ***Wall slide:*** desplaçar-se lliscant per una paret, caient més lentament.
- **Interactuar:** interactuar amb un objecte. Les possibles interaccions són:
    - Obrir una porta
    - Obrir un cofre
    - Activar un *checkpoint*
    - Recollir un *upgrade, spell, boss essence*
    - Activar una palanca
    - Destruir una paret

Desbloquejades recollint un *upgrade*:

- ***Wall jump:*** mentre s'està fent *wall slide*, permet realitzar un salt en sentit contrari a la paret.
- ***Dash:*** moviment lateral molt ràpid en sentit en el qual mira el personatge.
- ***Double jump:*** mentre s'està a l'aire, permet realitzar un salt extra després de saltar.

Desbloquejades recollint un *spell* (encanteri màgic que permet al jugador realitzar certes accions especials).

- ***Heal:*** permet recuperar vides.
- ***Shield:*** protegeix al jugador de qualsevol atac enemic durant uns pocs segons.
- ***Fireball:*** dispara una bola de foc que danya als enemics, en direcció a la qual apunta el joystick del comandament, excepte cap a baix.
- ***Plunge:*** moviment cap a baix molt ràpid, que danya als enemics i pot destruir algunes parets.

5.2.3. Recursos, objectes i atributs

Recursos

- Vides: cops que pot rebre dany el jugador abans de morir
- Manà: energia que s'usa per llançar *spells*
- Scrolls recollits: objectes que el jugador ha de trobar per poder completar un trial i desbloquejar un nou spell

Objectes i atributs:

- *Boss essence:* recollit, no recollit
- *Checkpoint*: actiu, no actiu
- Cofre: obert, no obert
- Porta: oberta, tancada, bloquejada, desbloquejada
- *Fireball cannon*: disparant, recarregant
- Palanca: activada, desactivada
- Plataforma: (sense atributs)
- *Scroll:* recollit, no recollit
- *Shield obstacle*: actiu, no actiu
- Punxes: (sense atributs)
- *Upgrade*: recollit, no recollit
- Parets: destruïda, no destruïda

5.2.4. Economia del joc

Les vides són un comptador numèric amb un rang entre zero i cinc. És intangible perquè no ocupa espai físic dins el videojoc. És concret perquè existeix dins el videojoc.

El manà és un comptador numèric amb un rang entre zero i cent. És intangible perquè no ocupa espai físic dins el videojoc. És concret perquè existeix dins el videojoc

Els *scrolls* són un comptador numèric amb un rang entre zero i tres per cada tipus d'*scroll*, perquè cada tipus està relacionat amb un *spell* diferent. És tangible perquè ocupa espai físic dins el videojoc. És concret perquè existeix dins el videojoc

Recursos i entitats

- Vides: Intangible / Concret -> Comptador de vides (numèric)
- Manà: Intangible / Concret -> Comptador quantitat de manà (numèric)
- *Scrolls*: Tangible / Concret -> Comptador d'*scrolls* (numèric)

Funcions

Sources:

- Vides:
  - El jugador utilitza l'*spell heal* (recupera una vida cada cert temps que usa *heal*)
  - El jugador respawneja després de morir (recupera totes les vides)
- Manà: jugador colpeja a un enemic

Drains:

- Vides: jugador rep atac d'un enemic
- Manà: jugador utilitza un *spell*

Converters:

- *Scrolls*: jugador intercanvia tres *scrolls* per un *upgrade* d'un *spell* nou

Traders:

Per aquest projecte no s'ha implementat cap *trader*, però s'han plantejat diverses idees que es podran implementar en el futur. S'explicaran més endavant a l'apartat 9, Treball futur.

5.3. Narrativa

5.3.1. Sinopsi

El videojoc segueix la història d'un jove mag que retorna al seu antic món després d'alliberar-se de l'encanteri que el tenia apressat. Descobreix que el món també ha estat víctima d'un encanteri que ha contaminat les ments dels habitants. Haurà de descobrir la forma de destruir l'encanteri i al conjurador i retornar el món a la normalitat.

5.3.2. Estructura narrativa

L'estructura narrativa del videojoc es dividirà en quatre parts principals: Introducció, desenvolupament, clímax i desenllaç.

La introducció transcorre a les zones de tutorial i *Rotting Castle*. Durant aquesta part, es presenta al protagonista i es descobreix la història del món. Al final d'aquesta part s'expandeix la narrativa a les diferents possibilitats de les noves zones a explorar.

Al desenvolupament, cada una de les zones esdevé un acte separat. Com que el jugador pot explorar les zones en diferent ordre, els actes no segueixen un ordre cronològic, si no cada un d'ells aportarà nova informació de la història i caldrà completar-los tots per recopilar tot el conjunt de la narrativa.

Un cop completades totes les zones, la narrativa s'integra i avança al clímax. Aquest transcorre a l'última zona on el protagonista haurà de lluitar amb el *boss* final, que explicarà les seves motivacions i tancarà el misteri que s'arrossega des de la introducció. Un cop acabada la lluita i havent guanyat el protagonista, es passarà al desenllaç.

Al desenllaç es mostrarà com el món torna a la normalitat després que el protagonista hagi derrotat al *boss* final i després, es veurà el començament de la nova aventura que espera al protagonista.

5.3.3. Plot points i Granularity

Els plots points principals seran els combats contra el *boss* de cada zona. Serviran per desvelar informació important sobre què ha succeït en aquell tros de món.

La freqüència amb la qual succeixen aquests serà baixa, ja que solament hi ha cinc *bosses* i quatre es troben distribuïts al final de cada zona i un al clímax del joc.

Altres plot points menys importants seran les trobades amb alguns NPC, que aportaran detalls menys rellevants, però que enriqueixen la història del món.

La freqüència d'aquests dependrà del jugador i de la seva habilitat explorant, ja que, per cada zona, hi haurà diferents NPC que caldrà trobar per recopilar nova informació.

5.3.4. Estudi del món

El món en el qual basem el projecte està format per sis zones diferents interconnectades entre elles. Per aquest projecte, ja que creem un prototip més petit, només desenvoluparem i implementarem la primera zona. Tot i així, s'ha creat un esquema de la disposició general de les zones (veure Figura 25). El tutorial no s'ha comptat com a zona, ja que només es pot explorar a l'inici del videojoc i no s'hi pot tornar un cop s'accedeix a la zona 1.

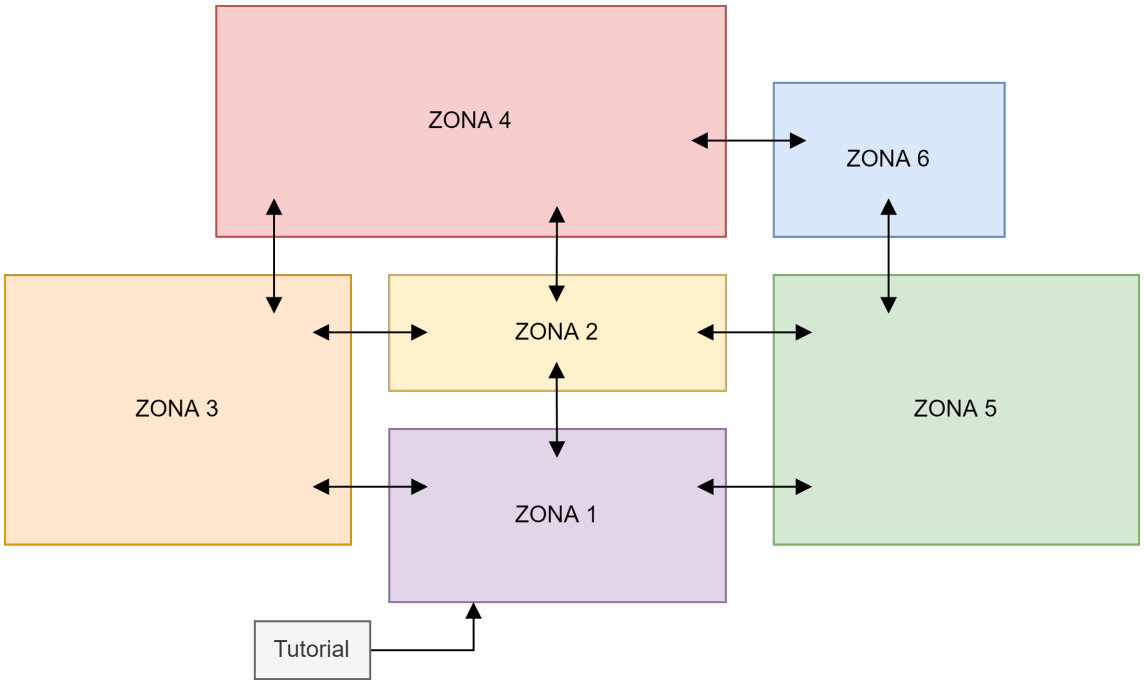

Figura 25: Esquema zones

A continuació, s'explica de manera resumida l'ambientació de cada zona. Cada zona tindrà: enemics i *bosses* propis que estaran fortament relacionats amb les característiques de la zona, i *upgrades* i *spells* nous que permetran al jugador anar augmentant el seu poder.

- **Zona 1 – Rotting Castle:** estarà ambientada en un antic castell abandonat, consumit per la vegetació amb sales vigilades per enemics infectats.

- **Zona 2 – The Town:** petit poble que no s'ha vist afectat per l'encanteri. Actua de *Hub World*, on el jugador pot interactuar amb diferents NPC i intercanviar objectes.

- **Zona 3 – Endless Mines:** estarà ambientada en unes coves i mines amb rius de lava i passadissos laberíntics.

- **Zona 4 – Hidden Mountain:** estarà ambientada en les parts superiors d'una muntanya nevada amb salts desafiants.

- **Zona 5 – Corrupted Lands:** estarà ambientada en les proximitats d'una ciutat, consumides per l'ambient verinós de la següent zona.

- **Zona 6 – Floating City:** estarà ambientada en una ciutat amb alts edificis, que emana el verí que intoxica al món.

5.4. Dissenys dels espais

Com s'ha comentat anteriorment, en aquest projecte ens hem centrat en la primera zona i n'hem dissenyat la disposició. S'ha començat amb un disseny inicial, que després s'ha revisat dues vegades fins a arribar a la versió final que s'implementarà. Els primers dos dissenys s'han fet a mà, amb llapis i paper, i l'últim ja s'ha fet digitalment. Els dissenys són els mostrats a les Figures 26, 27 i 28.

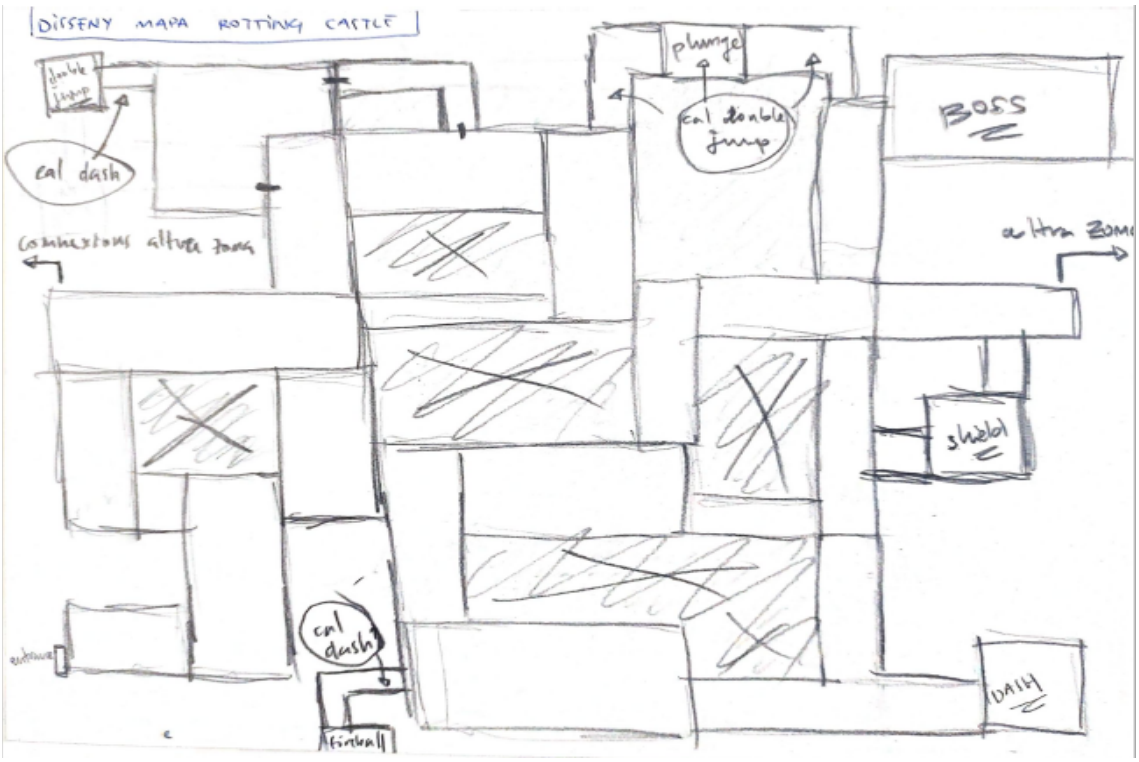

Figura 26: Disseny inicial

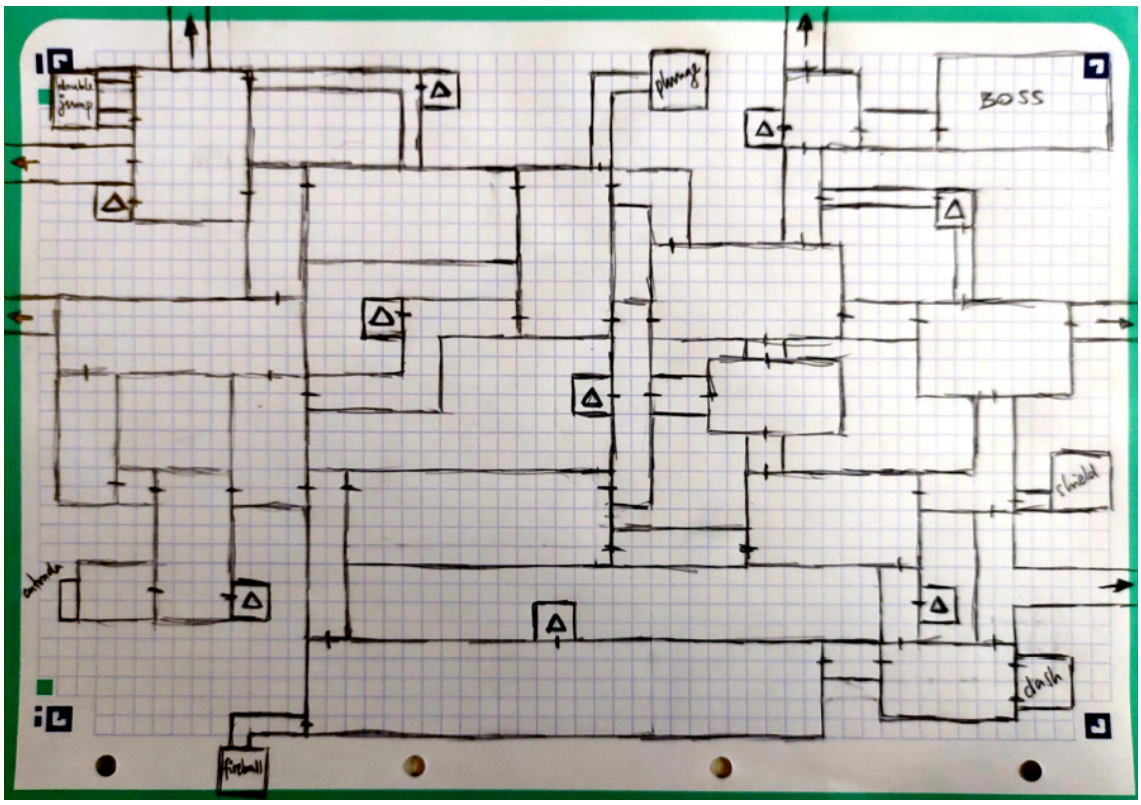

Figura 27: Primera revisió disseny

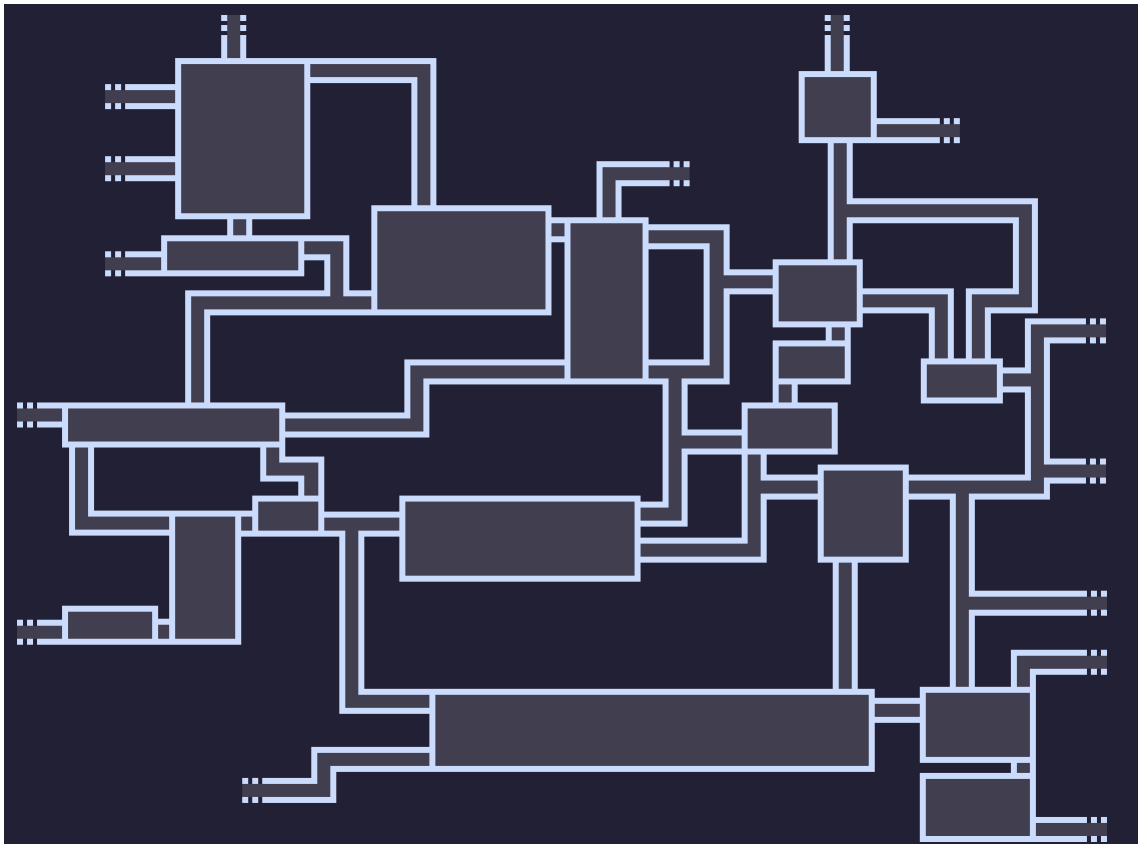

Figura 28: Segona revisió disseny

Per a crear el disseny i les revisions, s'ha tingut en compte les diferents característiques dels nivells dels videojocs del gènere Metroidvania.

Un dels trets principals és que els nivells estan compostos per zones no accessibles inicialment, que requereixen d'habilitats desbloquejables. Aquestes zones caldrà tornar a visitar-les després d'aconseguir l'habilitat necessària per avançar la història o obtenir altres habilitats que es requereixen en altres zones del nivell. Aquest procés és el que s'anomena *backtracking*, i incita a l'exploració exhaustiva del nivell. Per bloquejar les zones s'han col·locat trampes o obstacles que no permeten passar sense l'habilitat necessària.

La disposició de cada un dels elements del nivell està pensada amb un objectiu concret, com veurem seguidament:

- La dels enemics permet anar introduint els enemics de manera gradual en ambients poc hostils, perquè quan el jugador se'ls torni a trobar ja conegui els atacs respectius i sigui un combat just, que faci sentir poderós al jugador.

- La de les trampes busca bloquejar les zones a les quals no es pot accedir sense alguna habilitat, o augmentar la dificultat d'una zona del nivell i aconseguir que aquesta sigui més entretinguda i desafiant.

- La dels *scrolls* incita a l'exploració completa i atenta del mapa, ja que es troben repartits per passadissos o sales de tot el nivell, amagats darrere de parets destruïbles.

- La de les sales d'*upgrades* dins el nivell està pensada perquè el jugador les pugui descobrir quan disposa de les habilitats necessàries. Primer només podrà trobar la sala on obtindrà el *dash.* Un cop obtingut podrà trobar la següent on aprendrà el *double jump.*

- La de les sales de *trials* dins el nivell està pensada perquè el jugador pugui trobar-les segons va explorant i si segueix un sentit d'exploració lògic les trobarà en l'ordre esperat: *shield*, *fireball* i *plunge.*

5.5. Visió global

El diagrama de flux de les Figures 29 i 30 mostra la visió global del videojoc considerant el nivell bàsic i que es juga per primera vegada:

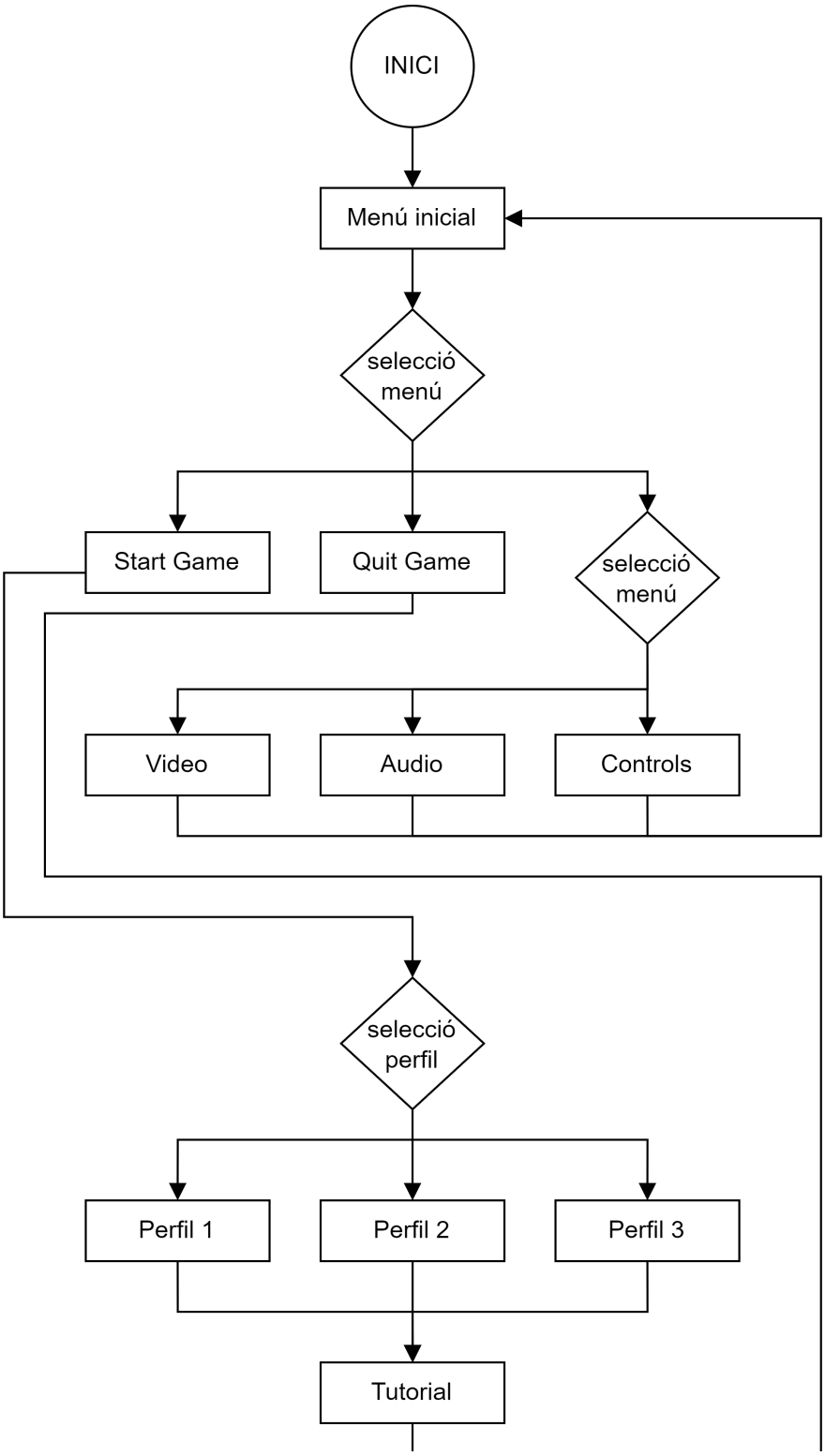

Figura 29: Diagrama de flux (part 1)

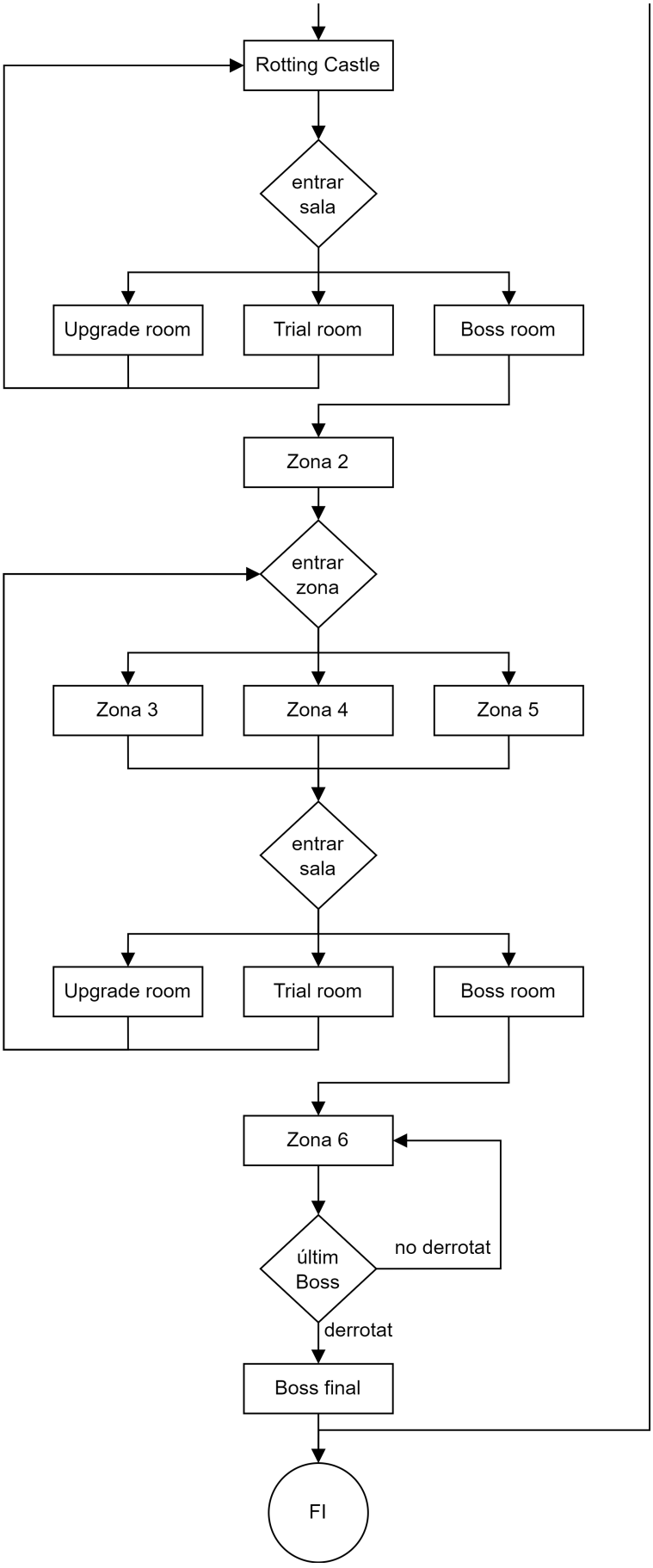

Figura 30: Diagrama de flux (part 2)

5.6. Disseny personatges

El disseny dels personatges seguirà un estil artístic comú, amb l'objectiu d'integrar els diferents personatges i mantenir una coherència estètica entre ells, els nivells i els objectes. Com ja s'ha explicat, aquest estil serà píxel art amb tocs retro, dissenys simples i paletes de pocs colors.

Les paletes de colors agrupen els personatges en dos grups: protagonista / NPC i enemics / Boss, mantenint les paletes similars per a què el jugador pugui entendre de manera ràpida i senzilla si un personatge amb el qual no ha interactuat encara en el videojoc és amic o enemic.

A continuació es descriuran els personatges, es comentarà el seu pes narratiu dins la història i es mostraran els esbossos i paletes de colors usats per dissenyar-los.

5.6.1. Protagonista

És el personatge controlat pel jugador. És un aprenent de mag, força baixet i que no domina gaire la màgia encara.

Té el pes narratiu més important, perquè és el que mou la història i la fa progressar o aturar.

En quant a l'estètica, és el personatge amb el qual s'han fet més proves. Primer es va iterar un disseny, però es va decidir fer-lo més amigable i reduir-ne l'alçada. Es van fer més proves amb el segon disseny fins a arribar al final. Veure Figures 31, 32 i 33.

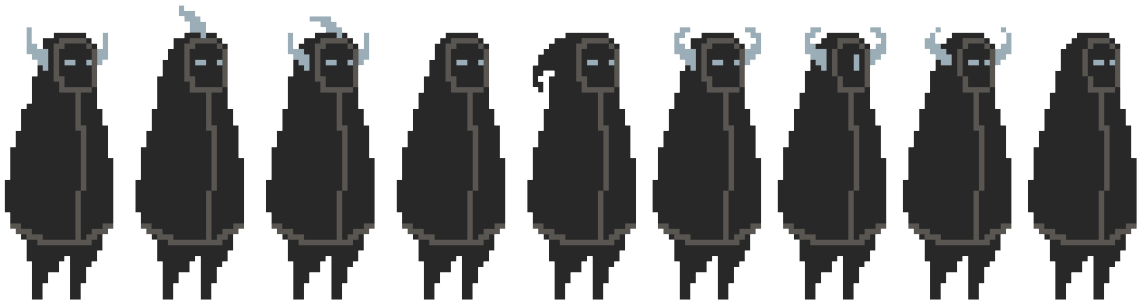

Figura 31: Primers dissenys protagonista

Figura 32: Segons dissenys protagonista

Figura 33: Tercers dissenys protagonista

La paleta del disseny final és la següent (Figura 34):

Figura 34: Paleta protagonista

5.6.2. Enemics

El pes narratiu dels enemics és força baix, ja que no aporten novetats a la història ni nova informació. Tot i així, contribueixen a l'ambientació i enriqueixen el món, fent que se senti més viu i real.

Roamer

És l'enemic més feble. No persegueix al jugador, simplement patrulla una zona d'un costat a l'altre. Tot i que és fàcil d'eliminar, no és estrictament necessari fer-ho, ja que no suposa una gran amenaça.

En quant a l'estètica, es va dissenyar un personatge senzill en forma de bolet i es van anar afegint diferents detalls fins a arribar al resultat final. Veure Figura 35.

Figura 35: Dissenys Roamer

La paleta del disseny final és la següent (Figura 36):

Figura 36: Paleta Roamer

Bomber

Aquest enemic patrulla una zona i quan detecta al jugador el persegueix mentre es mantingui dins el rang de visió i li va disparant projectils cada cert temps.

En quant a l'estètica, es va trobar un disseny inicial que després es va anar retocant la silueta i la forma i afegint els detalls finals. Veure Figura 37.

Figura 37: Dissenys Bomber

La paleta del disseny final és la següent (Figura 38):

Figura 38: Paleta Bomber

Charger

Aquest enemic patrulla una zona i quan detecta al jugador augmenta la velocitat i intenta envestir-lo.

En quant a l'estètica, a partir del disseny inicial es van afegir detalls per fer l'enemic més interessant i detallat. Veure Figura 39.

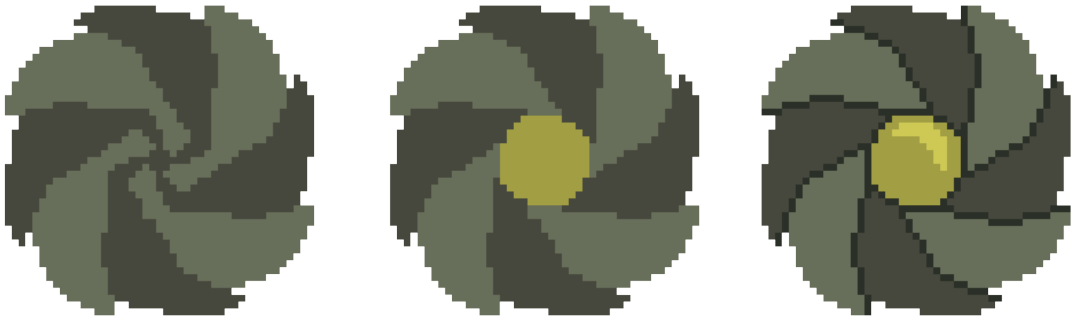

Figura 39: Dissenys Charger

La paleta del disseny final és la següent (Figura 40):

Figura 40: Paleta Charger

5.6.3. Boss

És l'enemic més poderós. Té molta més vida que la resta i disposa de tres atacs diferents. El primer atac, colpeja el terra amb el final del mànec de la destral que porta i cauen roques del sostre de la sala. El segon, ataca amb la destral i genera una ona expansiva d'energia. Finalment, el tercer és un atac bàsic amb la destral.

Té un pes narratiu molt alt, ja que és un plot point important de la història que aporta nova informació i fa avançar la narrativa.

En quant a l'estètica, es buscava que fos amenaçant i poderós. Primer tenia una maça, però es va optar per una destral. Es va retocar la forma base i es van afegir els detalls al personatge i a l'arma. Veure Figura 41.

Figura 41: Disseny Boss

La paleta del disseny final és la següent (Figura 42):

Figura 42: Paleta Boss

5.6.4. NPC

Es troba a les sales de *trials.* Custodia els cofres que proporcionen nous *spells* al jugador. Cal que el jugador hagi trobat els *scrolls* corresponents perquè aquest li permeti obrir el cofre al completar el *trial.*

Té un pes narratiu més important que els enemics, perquè proporciona la possibilitat d'aconseguir noves habilitats, però menys important que un *boss*, perquè no fa avançar la història ni aporta nova informació a la narrativa.

En quant a l'estètica, es van aporfitar els dissenys inicials que es van descartar del protagonista i es van fer unes iteracions més fins arribar al disseny final. Veure Figura 43.

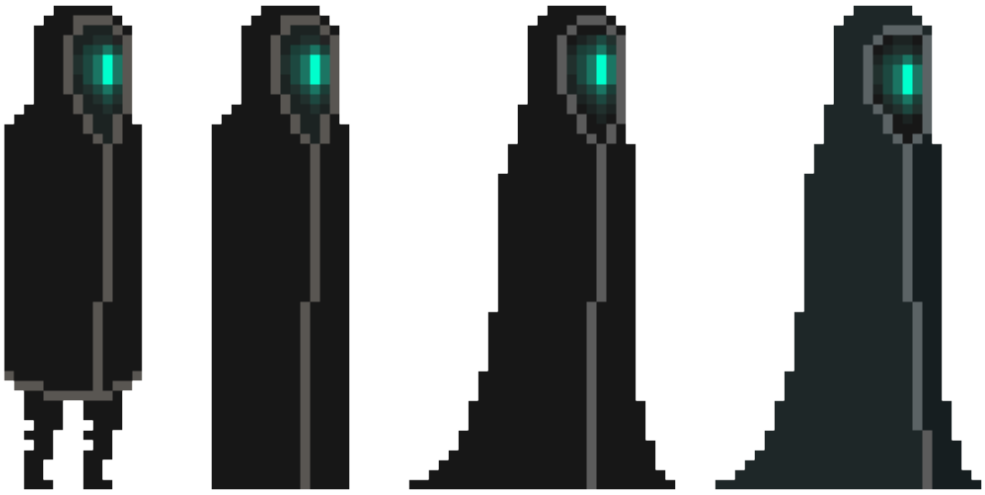

Figura 43: Dissenys NPC

La paleta del disseny final és la següent (Figura 44):

Figura 44: Paleta NPC

5.7. Disseny objectes

El disseny estètic dels objectes s'ha fet tenint en compte el disseny de l'escenari, perquè mantinguin una coherència estètica amb aquest i entre ells.

5.7.1. *Boss essence*

El desprenen els *bosses* quan són derrotats. El jugador els ha de recollir durant el videojoc per anar avançant a la història.

Figura 45: Boss essence

5.7.2. *Checkpoint*

El jugador els pot activar, però només hi pot haver un d'actiu. Quan el jugador es mor, apareix en l'últim activat amb totes les vides recuperades.

Figura 46: Checkpoint

5.7.3. Cofre

El jugador els ha d'anar trobant pels diferents nivells i els pot obrir. Sempre contenen un *upgrade* que proporcionarà al jugador un nou *spell* o una nova habilitat.

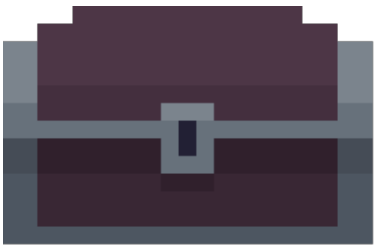

Figura 47: Cofre

5.7.4. Porta

Separen certes sales i poden ser bloquejades per restringir l'accés a zones no desbloquejades o a les quals no es pot tornar, i per tancar al jugador en aquella sala fins que faci certa acció, com per exemple en una sala d'un *boss*.

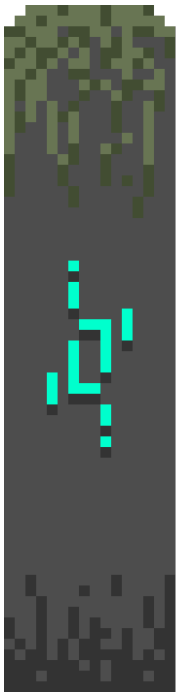

Figura 48: Porta

5.7.5. *Fireball cannon*

Es troben en sales de *trials.* Dispara boles de foc i si el jugador és danyat per aquesta per una vida.

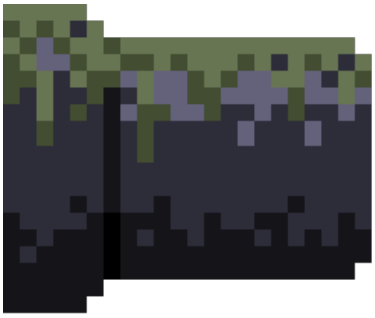

Figura 49: Fireball cannon

5.7.6. Palanca

Es troben en sales que també hi ha *shield obstacles.* Permeten desactivar-los per poder avançar per la sala.

Figura 50: Palanca

5.7.7. Plataformes

Superfícies sobre les que el personatge pot posicionar-se i utilitzar per accedir a les zones més remotes.

Figura 51: Plataformes

5.7.8. *Scroll*

Es troben distribuïts pels nivells i el jugador els ha de trobar per poder completar els *trials* i obtenir nous *spells.* Cada color representa amb quin *spell* estan relacionats.

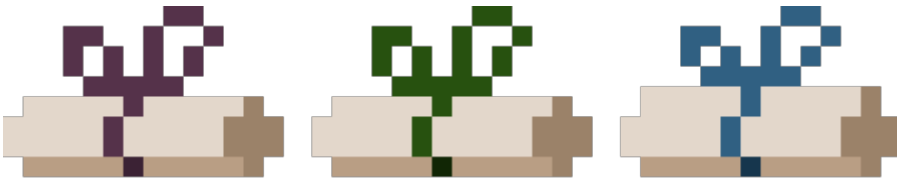

Figura 52: Scrolls

5.7.9. *Shield obstacle*

Barrera màgica que bloqueja el camí, cal activar una palanca per desactivar-la un cert temps.

Figura 53: Shield obstacle

5.7.10. Punxes

Trampa que resta una vida i reposiciona al jugador que aquest la toca. També serveix per bloquejar alguns camins fins que el jugador no hagi desbloquejat l'habilitat necessària.

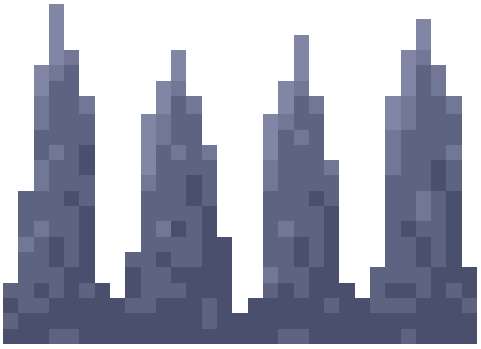

Figura 54: Punxes

5.7.11. *Upgrade*

Es troben dins els cofres, permeten al jugador desbloquejar habilitats o *spells* nous, que s'identifica segons el color de l'aura que els envolta.

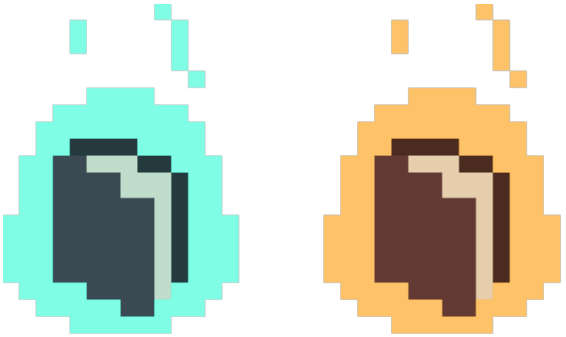

Figura 55: Upgrades

5.7.12. Parets

Són parets destruïbles que bloquegen el camí o amaguen un camí secret. El jugador les pot destruir per revelar els camins.

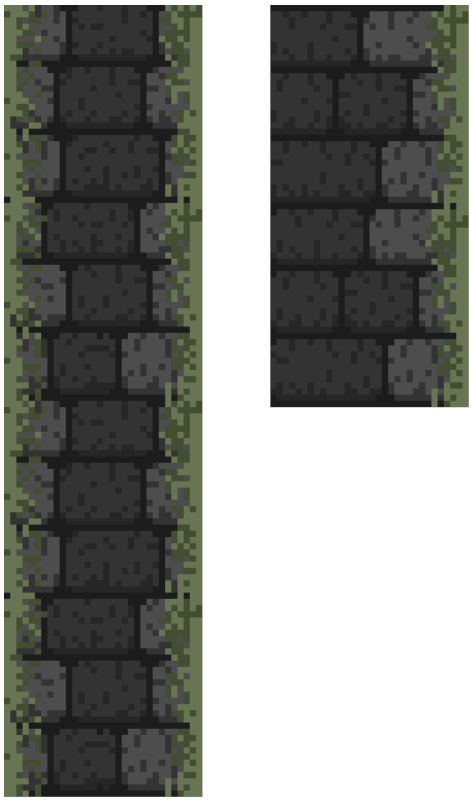

Figura 56: Parets

5.8. Interaccions

En aquest apartat es mostraran les interaccions entre elements que poden succeir durant una partida del videojoc. Aquestes interaccions es poden classificar en tres categories: personatge-personatge, objecte-objecte i personatge-objecte. Veure Figura 57.

Els elements que poden interactuar són:

- **Personatges:** Protagonista (jugador), Boss, Enemics (Roamer, Bomber i Charger) i NPC
- **Objectes:** BossEssence, Checkpoint, Cofre, Porta, Fireball Cannon, Palanca, Scroll, Shield Obstacle, Punxes, Upgrade i Paret.

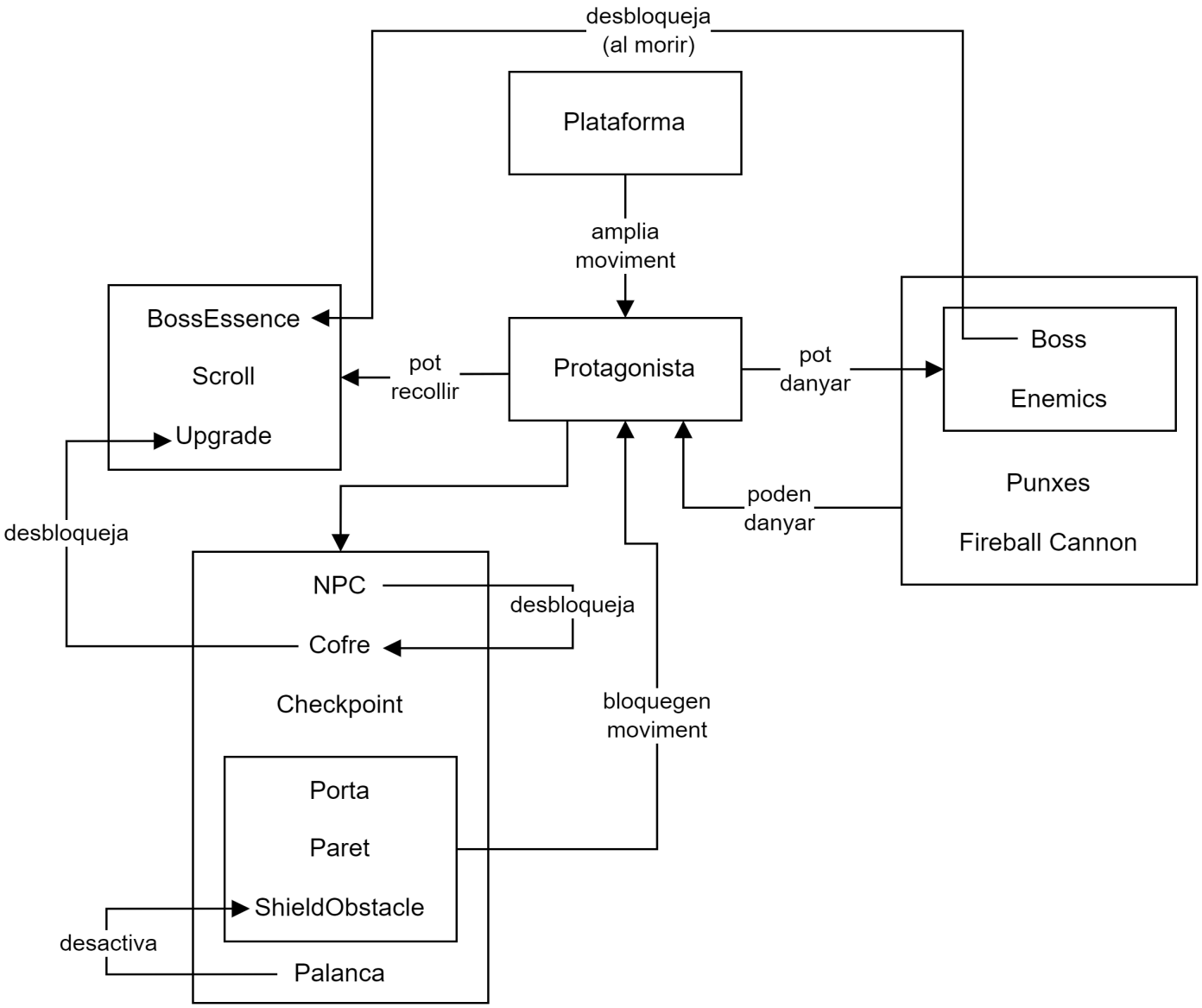

Figura 57: Interaccions entre elements del videojoc

5.9. Interfícies

Al videojoc s'han implementat dos tipus d'interfícies: el HUD, que es troba present durant la partida, i els menús, que es divideixen entre el del jugador, al que s'accedeix durant la partida, i el principal, al que s'accedeix a l'inici de la partida.

5.9.1. HUD

El HUD mostra el nombre de vides màxim i actual (triangles superiors grisos o verdosos), la quantitat de manà disponible (barra lila) i els *spells* actius per a cada acció (quadrats de l'esquerra). Veure Figura 58.

Figura 58: Captura Awaken nº1

També mostra el missatges de tutorial per aportar consells al jugador sobre les accions que pot realitzar. Veure Figura 59.

Figura 59: Captura Awaken nº2, missatge de tutorial

Quan s'obté un upgrade, mostra el nom del que s'ha desbloquejat i una petita descripció del que fa. Veure Figura 60.

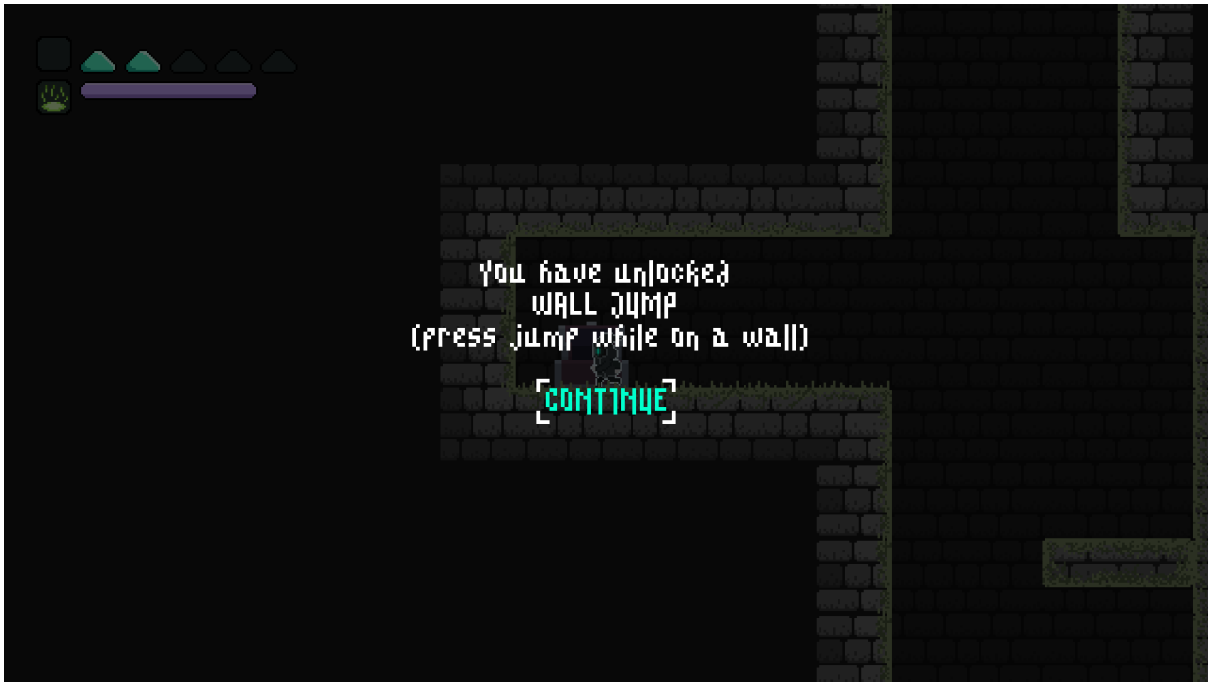

Figura 60: Captura Awaken nº3, descripció i consell d'upgrade desbloquejat

Quan el jugador s'apropa a un objecte amb el que pot interactuar, es mostra *Interact* sobre l'objecte en qüestió. Veure Figura 61.

Figura 61: Captura Awaken nº4, missatge interacció amb un cofre

5.9.2. Menú inicial

És el primer menú que trobem a l'iniciar el joc, conté tres opcions: *Start game*; *Options*, que porten als seus respectius submenús; i *Quit game*, que tanca el videojoc. Veure Figura 62.

Figura 62: Menú inicial

Perfils del jugador

Aquest submenú mostra els perfils disponibles i la zona en el que es troba el jugador. També permet esborrar la partida d'un perfil per començar de nou. Veure Figura 63.

Figura 63: Menú perfils del jugador

Options

Permet accedir a diferents submenús. *Video* (veure Figura 65) i *Audio* (veure Figura 66), on es poden modificar els paràmetres respectius, *Controls* (veure Figura 67), que mostra els controls del jugador i *Reset defaults*, que retorna els paràmetres de vídeo i audio als valors per defecte. Veure Figura 64.

Figura 64: Menú opcions

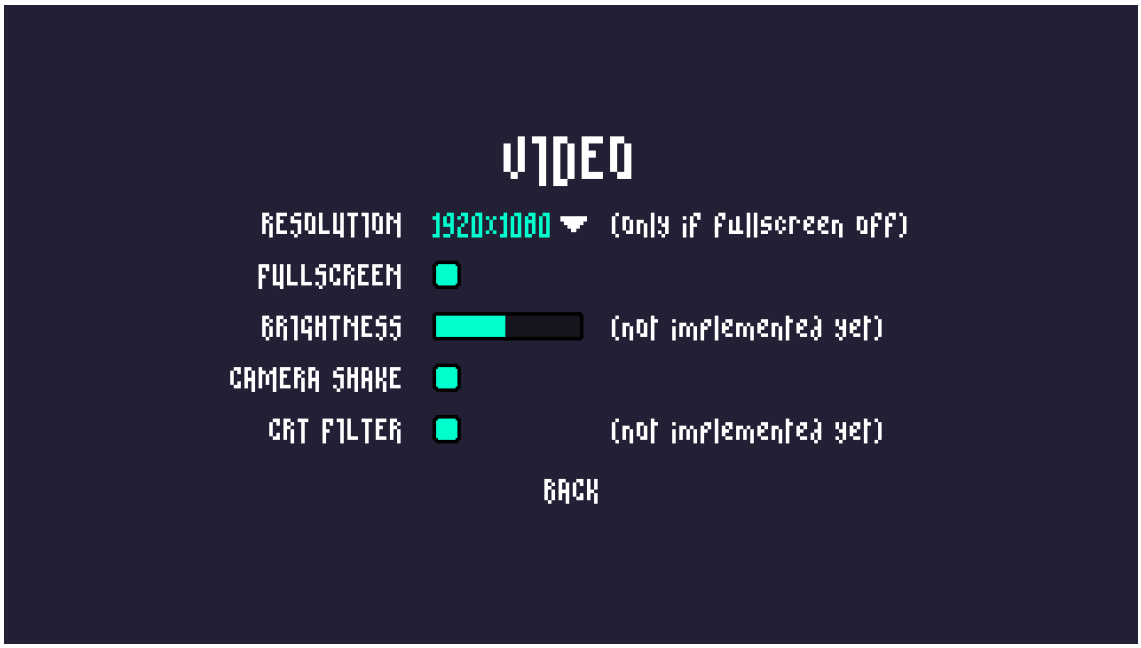

Figura 65: Menú vídeo

Figura 66: Menú audio

Figura 67: Menú controls

5.9.3. Menú jugador

El menú del jugador està format per tres submenús: *Inventory*, *Settings* i *Map.*

Inventory

Mostra els objectes que ha anat obtenint el jugador. A la part esquerra es mostren les *Boss essences*, que s'obtenen al derrotar un *Boss*, al centre es mostren els *upgrades* i a la part dreta superior es mostren els *spells* que s'han desbloquejat. A la part esquerra inferior es mostra el nom de l'objecte que està seleccionat, el seu tipus i una petita descripció. Veure Figura 68.

Figura 68: Menú inventori

Settings

Conté les mateixes opcions que es troben al menú inicial. Veure Figura 69.

Figura 69: Menú opcions

Map

Mostra el mapa del videojoc. Conforme es van arribant a les noves zones, aquestes s'afegeixen al mapa. En el cas de la Figura 70, només s'ha explorat la primera zona.

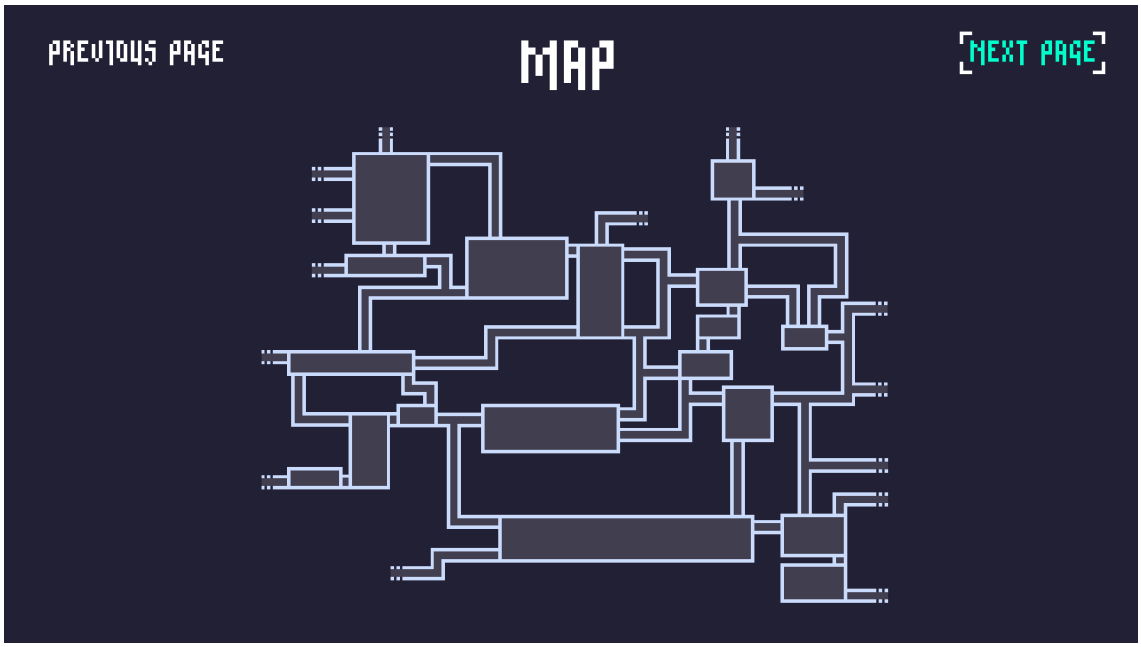

Figura 70: Menú mapa

5.9.4. Font

Per aquest projecte s'ha decidit dissenyar una font pixel art que tingués coherència amb l'estètica de la resta del videojoc perquè les fonts d'aquest estil que s'han trobat no s'ajustaven al resultat que es buscava.

Per a fer aquesta font s'ha utilitzat la web *Pixel Font Converter* que a partir d'una imatge de la font genera el fitxer ttf.

Per dissenyar la font s'han usat com a inspiració les fonts medievals tradicionals. Veure Figures 71 i 72.

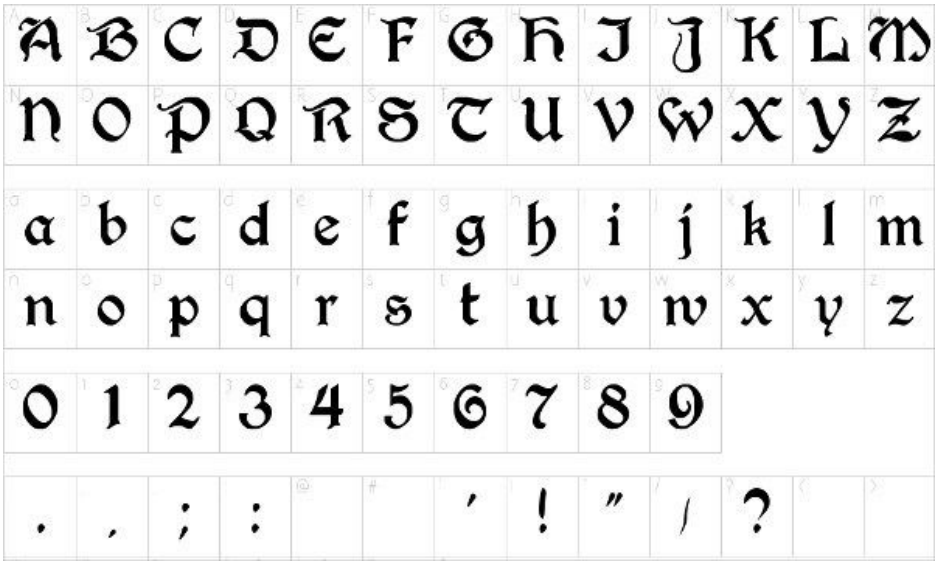

Figura 71: Font medieval inspiració 1

Figura 72: Font medieval inspiració 2

El resultat ha sigut una font que manté l'estètica de píxel art però amb característiques medievals, que s'ha utilitzat en tots els elements de text del videojoc. Veure Figura 73.

Figura 73: Font del projecte

6. Implementació i proves

En aquest apartat es comentarà l'estructura de les escenes principals, quins nodes contenen i quina funció tenen, i s'analitzaran els scripts més rellevants.

6.1. Nivells

6.1.1. Game

És l'escena principal del videojoc. Conté el *HUD*, el *PlayerMenu*, dos nodes de transicions i el *Level* (veure Figura 74). També gestiona el *CameraShake* (s'explicarà a la secció 6.2.2).

- *HUD* i *PlayerMenu* són instàncies de les respectives escenes. La implementació es comentarà més endavant a l'apartat 6.3.
- *Level* és l'escena que conté les escenes i nodes d'un nivell i que es va modificant segons el nivell al qual es troba el jugador, primer s'elimina el node i després s'instancia la nova escena amb el mateix nom.
- *TransitionCanvas* s'activa quan es canvia d'escena i es manté mentre s'elimina i s'instancia la nova escena.
- *DeathCanvas* s'activa quan el jugador mor i es manté mentre es gestiona el reposicionament del jugador i el respawn dels enemics.

Els dos nodes de transició (*TransitionCanvas* i *DeathCanvas*) fan un fos a negre, es mantenen en negre uns segons i després fan un fos a transparent.

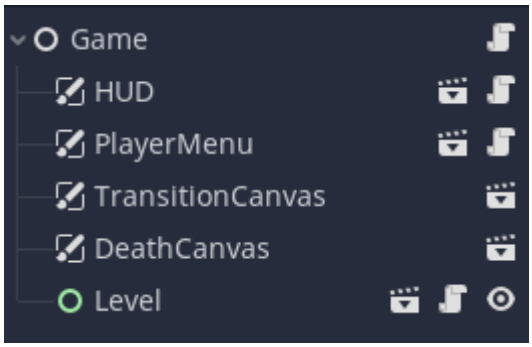

Figura 74: Nodes de Game

Les dues funcions importants d'aquesta escena són les següents (Els paràmetres que utilitzen s'explicaran més endavant a la secció 6.1.2, Elements comuns, quan es comenti *SceneChange).*

- ***change_scene:*** Aquesta funció es cridarà des d'altres escenes quan es fa el canvi d'escena. Primer de tot, inicia la transició de canvi d'escena a fos a negre. Un cop aquesta transició ha acabat, crida la funció *change* i inicia la transició de canvi d'escena a transparent. Activarà o desactivarà el HUD segons si l'escena nova serà el menú principal o no. Veure Figura 75.

```
18 func change_scene(next_scene, is_main, position_id):
19     main_menu = is_main
20     transition_animator.play("Alpha1")
21     yield(transition_animator, "animation_finished")
22     call_deferred("change", next_scene, position_id)
23     transition_animator.play("Alpha0")
24     if main_menu:
25         $HUD/MainContainer.visible = false
26     else:
27         $HUD/MainContainer.visible = true
28
29 func change(next_scene, position_id):
30     # Remove the current level
```

Figura 75: Funció change_scene

- ***change:*** Aquesta funció elimina l'escena actual (*Level*), crea una instància de l'escena nova, li canvia el nom a *Level* i l'afegeix a *Game.* Si la posició és diferent a zero, posicionarà al jugador a la posició indicada de la nova escena. Veure Figura 76.

```
27         $HUD/MainContainer.visible = true
28
29 func change(next_scene, position_id):
30     # Remove the current level
31     current_scene.free()
32     # Add the next level
33     var next_s = load(next_scene)
34     current_scene = next_s.instance()
35     current_scene.set_name("Level")
36     get_tree().get_root().get_node("Game").add_child(current_scene)
37     if position_id != 0:
38         $Level.player_to_position(position_id)
```

Figura 76: Funció change

6.1.2. Elements comuns

Primer de tot, es comentaran els elements comuns que tenen les següents escenes. Aquests elements són els bàsics perquè l'escena tingui l'aspecte correcte i pugui funcionar.

SceneChange

Aquest node gestiona el canvi d'escenes. Té 3 variables que s'han de definir quan s'instancia una escena:

- next_scene: escena a la qual es vol canviar
- is_main: si l'escena nova és el main menú
- position_id: posició dins l'escena següent a la qual ha d'aparèixer el jugador. Serà 0 si només hi ha una posició possible

La principal funció és *_on_SceneChangeArea2D_body_entered.* Aquesta funció detecta quan el jugador entra a una àrea. Llavors, desactiva has_camera del *CameraShake* (s'explicarà més endavant perquè es fa això) i crida la funció *change_scene()* de l'escena *Game*, amb les variables que ja s'han definit. Veure Figura 77.

```
11 func _on_SceneChangeArea2D_body_entered(body):
12     CameraShake.has_camera = false
13     var root = get_tree().get_root()
14     root.get_node("Game").change_scene(next_scene, is_main, position_id)
```

Figura 77: Funció _on_SceneChangeArea2D_body_entered

BaseTileMap

Aquest node conté el Tilemap que delimita la zona per on es podrà moure el jugador. És l'únic tilemap que té col·lisions.

GrassTileMap

Aquest node és de decoració i per tant, no té col·lisions. Afegeix un efecte de gespa sobre el BaseTileMap.

BackgroundTileMap

Aquest node és de decoració i per tant no té col·lisions. Afegeix el fons darrere el BaseTileMap.

HiddenPaths

Aquest node conté els tilemaps que serveixen per amagar els passadissos secrets. Veure Figura X.

HiddenWalls

Aquest node conté les instàncies de les parets destruïbles. Veure Figura 78.

Figura 78: Nodes de HiddenPaths i HiddenWalls

Player

Aquest node és el que controlarà el jugador (s'explicarà detalladament a l'apartat 6.4.2).

Enemies

Aquest node conté les diferents instàncies dels enemics. Veure Figura 79.

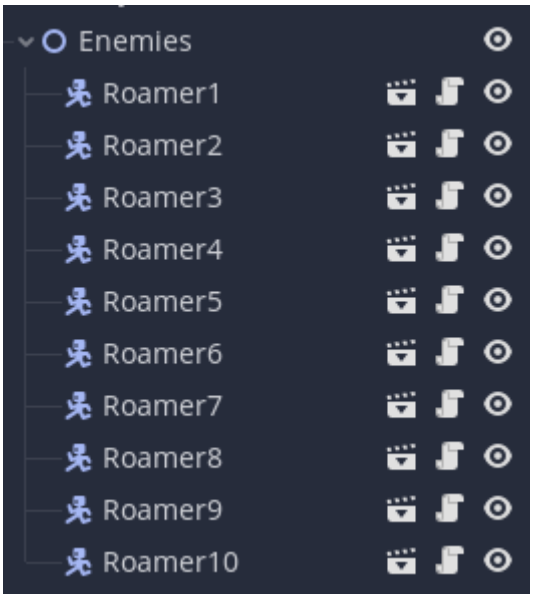

Figura 79: Nodes de Enemies

Spikes

Aquest node conté les diferents instàncies dels *spikesGroups* (s'explicaran detalladament a l'apartat 6.5.10). Veure Figura 80.

Figura 80: Nodes de Spikes

Checkpoints

Aquest node conté les diferents instàncies dels checkpoints. Veure Figura 81.

Chests

Aquest node conté les diferents instàncies dels chests. Veure Figura 81.

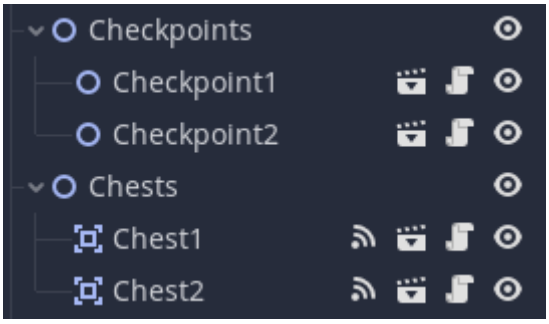

Figura 81: Nodes de Checkpoints i Chests

Platforms

Aquest node conté les diferents instàncies de les plataformes.

6.1.3. Tutorial

És l'escena del tutorial del videojoc. A part de tenir tots els elements comuns ja comentats, també té un node anomenat *TutorialAreas. TutorialAreas* conté les diferents àrees que mostren un missatge de tutorial quan el jugador hi entra. Veure Figura 82.

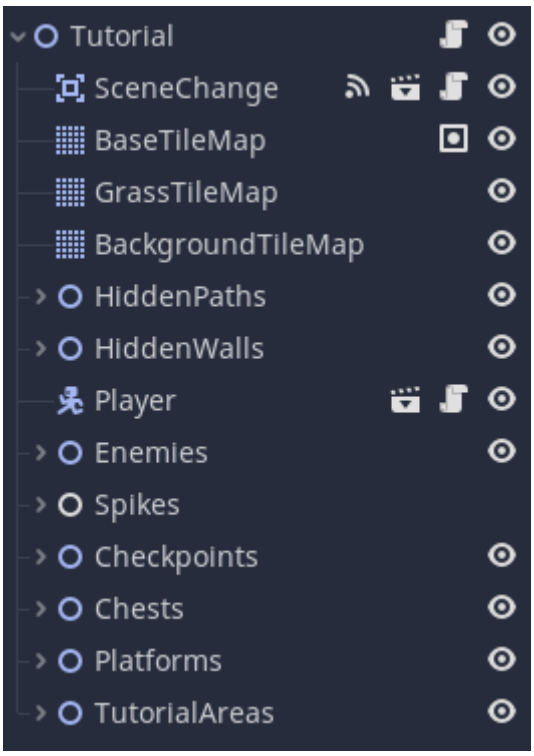

Figura 82: Nodes de Tutorial

6.1.4. Rotting Castle

És l'escena de la primera zona principal del videojoc. A part de tenir tots els elements comuns ja comentats, també té un node anomenat *Scrolls,* que conté els diferents tipus d'*scrolls* que es troben repartits per l'escenari. Veure Figura 83.

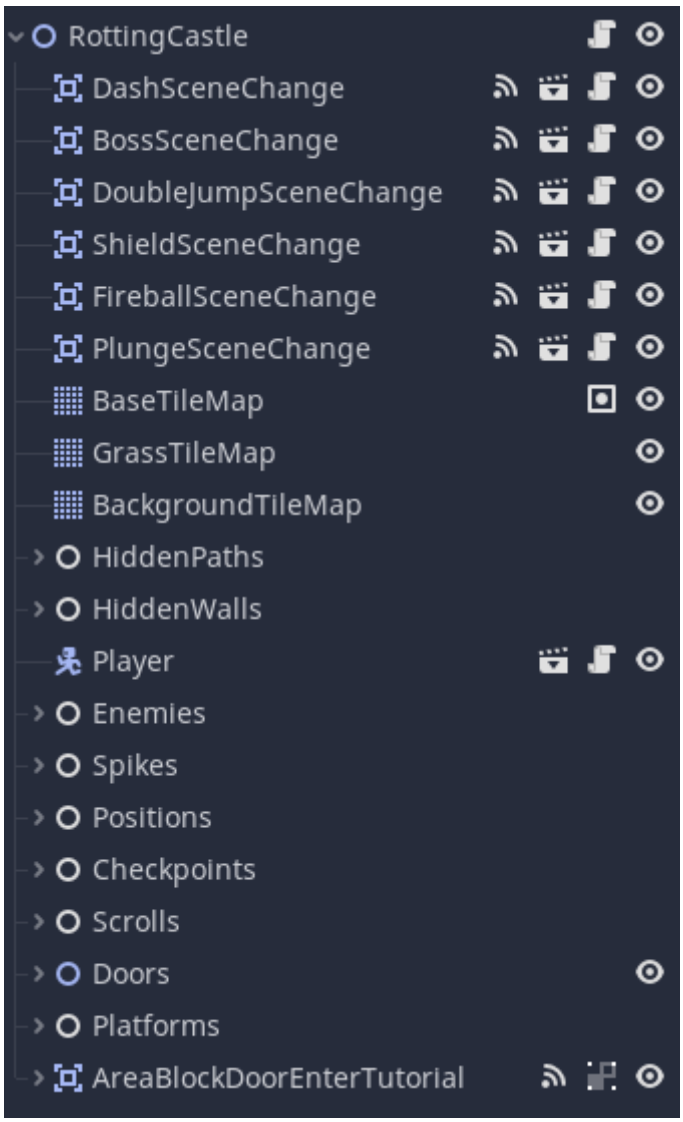

Figura 83: Nodes de Rotting Castle

6.1.5. Sales d'upgrades

Són les escenes de les sales on el jugador pot desbloquejar *l'upgrade* del *dash* i del *double jump*. Només contenen alguns dels elements comuns, tal com es pot observar en les Figures 84 i 85.

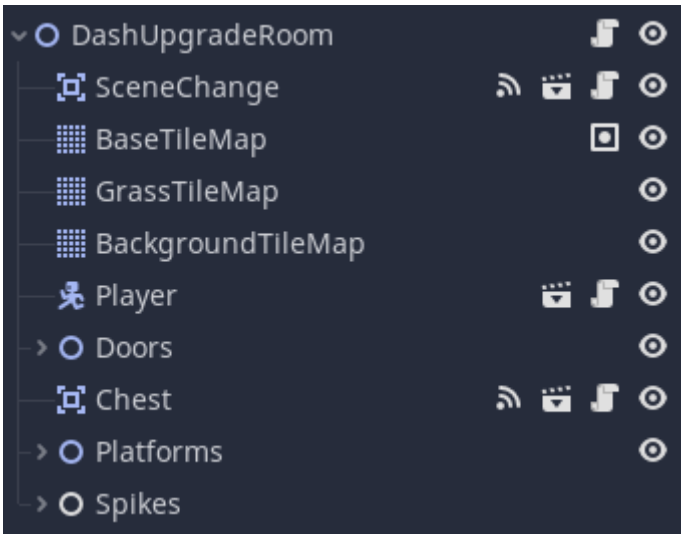

Figura 84: Nodes de DashUpgradeRoom

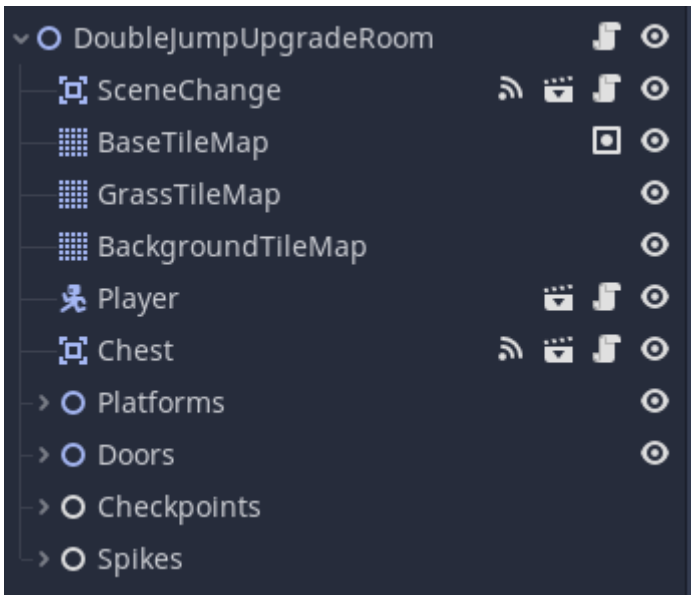

Figura 85: Nodes de DoubleJumpUpgradeRoom

6.1.6. Sales de trials

Són les escenes de les sales on el jugador pot desbloquejar els *spells Shield, Fireball* i *Plunge.* A part de contenir alguns dels elements comuns, tenen nodes que només es troben a les sales de *trials.* Aquests nodes són:

- *TrialCompletedArea*: la zona que detecta quan el jugador ha completat el *trial.*
- *MageNPC*: NPC que dóna indicacions al jugador.
- *Chest*: *chest* que proporcionarà el nou *spell* al jugador. Només es fa visible un cop s'ha completat el *trial.*

A part dels nodes anteriors, cada sala té uns nodes únics, que són:

- La sala del *trial* del *shield* té el node *Shields,* que conté les instàncies dels *shield obtacles,* i el *Levers,* que conté les instàncies dels *levers.* Veure Figura 86.

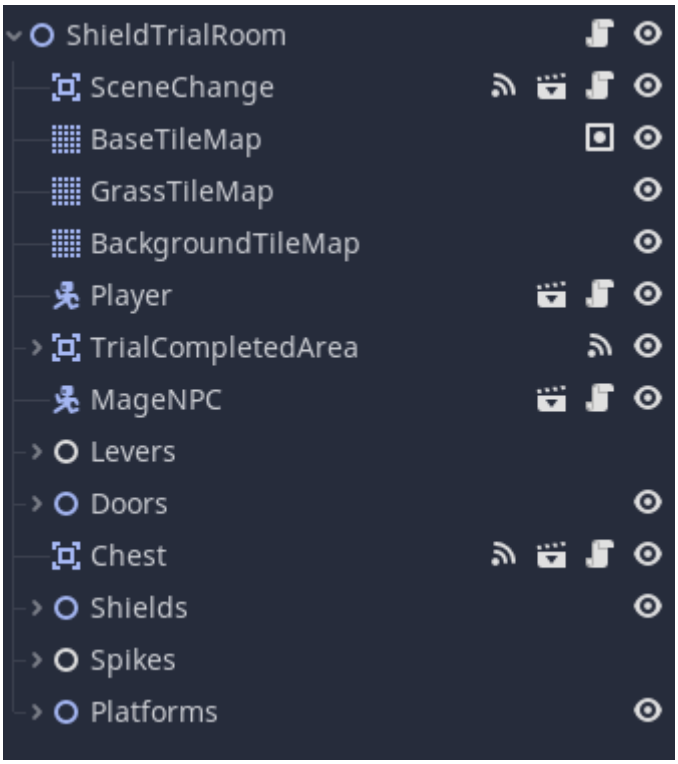

Figura 86: Nodes de ShieldTrialRoom

- La sala del *trial* del *fireball*, té el node *FireballCanons*, que conté les instàncies dels *fireball canons*, i el *CanonTimer,* que gestiona quan han de disparar els canons. Veure Figura 87.

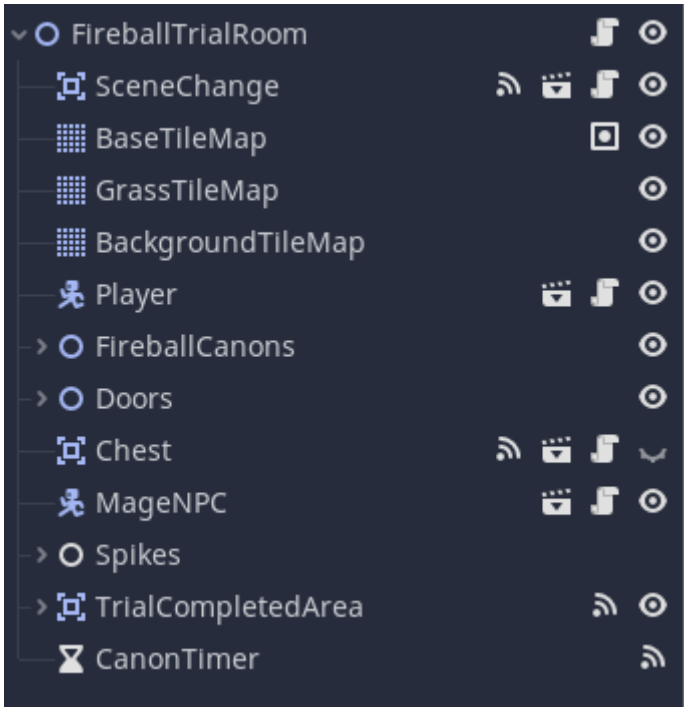

Figura 87: Nodes de FireballTrialRoom

- La sala del *trial* del *plunge*, té dos nodes pel MageNPC, un per l'entrada i un per la zona on es desbloqueja el chest, a causa de la disposició de la sala. També té el node *BlockDoorArea*, que al detectar el jugador bloqueja una porta que li impedeix sortir de la sala sense completar el trial. Veure Figura 88.

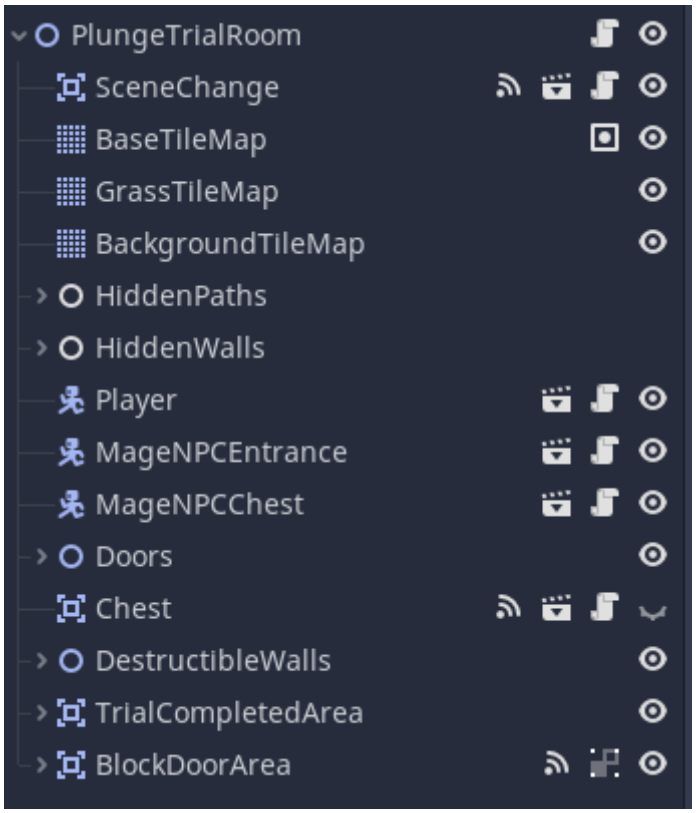

Figura 88: Nodes de PlungeTrialRoom

6.1.7. Sala del *boss*

És l'escena de la sala del boss de la primera zona. A part de tenir tots els elements comuns ja comentats, també té nodes únics, que són els següents. Veure Figura 89.

- Boss1: la instància del personatge del *boss*
- BossLifeBar: HUD que conté la barra de vida del *boss*
- CameraBoss: càmera secundària, la qual s'activa quan el jugador entra a la sala
- CamerasTween: node que gestiona la transició entre la càmera del jugador i la *CameraBoss*
- BossRoomArea2D: àrea que detecta quan el jugador ha entrat i inicia la transició entre les càmeres
- BossRoomCameraPosition: posició a la qual es troba la *CameraBoss*
- BossRockSpawnPosition1, 2, 3, 4 i 5: posicions des d'on spawnegen les roques que cauen després de l'atac del *boss*

Figura 89: Nodes de Boss1Room

6.2. Singletons

6.2.1. PlayerInfo

Guarda tota la informació del jugador: atributs, habilitats i *spells* desbloquejats.

Els atributs se separen segons si estan relacionats amb el moviment, amb el salt, amb el dash, amb el plunge, amb moviments de les parets o amb el knockback de quan es rep dany. Els atributs són els següents:

Moviment:

- var ACCELERATION : int = 50
- var MAX_MOVEMENT_SPEED : int = 200
- var SLOW_DOWN_FLOOR : float = 0.4
- var MAX_JUMPS : int = 2

Salt:

- var JUMP_HEIGHT : int = 400
- var ATTACK_JUMP_HEIGHT : int = 100
- var LOW_JUMP_MULTIPLIER : int = 250
- var DOUBLE_JUMP_HEIGHT : int = 350
- var GRAVITY : int = 15
- var MAX_FALL_SPEED : int = 600
- var COYOTE_TIME : float = 0.1

Dash:

- var DASH_DISTANCE : int = 1000
- var DASH_TIME : float = 0.15
- var DASH_RECHARGE : float = 0.6

Plunge:

- var PLUNGE_TIME : float = 0.5
- var PLUNGE_SPEED : float = 1500

Moviments de paret:

- var WALL_JUMP_PUSH : int = 250
- var WAIT_TIME : float = 0.5
- var WALL_SLIDE_GRAVITY : int = 2
- var MAX_FALL_SPEED_WALL : int = 200
- var WALL_SLIDE_MARGIN : float = 0.2

Knockback:

- var KOCKBACK_SPEED : int = 300
- var KOCKBACK_DURATION : float = 0.2

Les característiques són les següents:

- var MAX_LIFES : int = 5
- var MAX_MANA : float = 100.0
- var LIFES : int = 3
- var MANA : float = 100.0
- var MELEE_POWER : int = 1
- var SPELL_POWER : int = 2
- var SPELL_UP : int = 0
- var SPELL_DOWN : int = 0
- var LAST_CHECKPOINT = Vector2.ZERO

Les habilitats són les següents:

- var WALL_JUMP : bool = false
- var DASH : bool = false
- var DOUBLE_JUMP : bool = false

L'inventori conté les següents variables:

- var INVENTORY_UNLOCKED : bool = false
- var boss_essences_collected = [false, false, false, false]
- var scrolls_shield : int = 0
- var scrolls_fireball : int = 0
- var scrolls_plunge : int = 0
- var spells_up_learned = [false, false]
- var spells_down_learned = [false, false]

Aquest singleton també gestiona la connexió i desconnexió del comandament. Per fer-ho, s'utilitzen les següents tres funcions. Veure Figura 90.

- **_init():** connecta el senyal *joy_connection_changed* a l'*Input*. Aquest senyal es generarà quan detecti que s'ha connectat o desconnectat un comandament. També comprova si hi ha un controller connectat amb la funció *check_controller().*

- **_joy_controller_changed():** quan es detecta el senyal *joy_connection_changed,* es crida a aquesta funció amb el mateix nom perquè comprovi si hi ha comandaments connectats amb la funció *check_controller().*

- **check_controller():** per comprovar si hi ha un comandament connectat, es consulta la mida de la taula que s'obté d'*Input* amb la funció *get_connected_joypads().* Si el resultat és més gran que zero, hi haurà un comandament connectat; contrariament, no hi haurà cap de connectat.

```
75  ## CONTROLLER CONNECTION ##
76  var controller_connected : bool
77
78  func _init():
79      Input.connect("joy_connection_changed",self,"_joy_connection_changed")
80      check_controller()
81
82  func _joy_connection_changed(_id, _connected):
83      check_controller()
84
85  func check_controller():
86      if Input.get_connected_joypads().size() > 0:
87          controller_connected = true
88      else:
89          controller_connected = false
```

Figura 90: Funcions de PlayerInfo, gestió del commandament

6.2.2. Camera shake

El *camera shake* sacseja la càmera del jugador quan executa un atac o quan rep dany. Per implementar-ho, s'ha adaptat una implementació d'un projecte de Github anomenat *Camera Shake Tutorial*.

Té quatre variables:

- **shake_intensity:** Determina la intensitat de la vibració de la càmera
- **shake_duration:** Determina la duració de la vibració de la càmera
- **camera:** Defineix la càmera sobre la qual aplicar la vibració
- **has_camera:** Assegura que es disposa d'una càmera activa

Té dues funcions (veure Figura 91):

- **shake:** Primer es comprova si l'usuari té activada la vibració de la càmera, que està guardat a *Settings.* Després, si la intensitat o la duració que s'han passat com a paràmetres són més grans que les variables respectives, en substitueix els valors.
- **_process:** Primer comprova si hi ha una càmera activa. Després comprova si encara queda duració de la vibració. Finalment, aplica la vibració afegint un offset a la càmera de manera aleatòria.

```
 3  var shake_intensity : float = 0.0
 4  var shake_duration : float = 0.0
 5  var camera
 6  var has_camera : bool
 7
 8  func shake(intensity, duration):
 9      if not Settings.camera_shake:
10          intensity = 0.0
11
12      if intensity > shake_intensity and duration > shake_duration:
13          shake_intensity = intensity
14          shake_duration = duration
15
16  func _process(delta):
17      if has_camera:
18          if shake_duration <= 0.0:
19              camera.offset = Vector2.ZERO
20              shake_intensity = 0.0
21              shake_duration = 0.0
22              return
23
24          shake_duration = shake_duration - delta
25          var offset = Vector2(randf(), randf() * shake_intensity)
26          camera.offset = offset
```

Figura 91: Singleton Camera Shake

6.2.3. Settings

Guarda els settings del jugador. Les variables es separen segons vídeo o audio:

Vídeo:

- var resolution : Vector2
- var fullscreen : bool
- var brightness : int
- var camera_shake : bool
- var crt_filter : bool

Audio:

- var master_volume : int
- var sound_volume : int
- var music_volume : int

Té un setter per cada una de les variables, que s'usa quan es modifica algún valor dels del menú inicial o del jugador. Les principals funcions són set_defaults i set_all. Veure Figura 92.

- **set_defaults:** assigna els valors per defecte a les variables.
- **set_all:** crida a tots els setters amb els nous valors.

```
18 func set_defaults():
19     resolution = Vector2(1920, 1080)
20     fullscreen = true
21     brightness = 50
22     camera_shake = true
23     crt_filter = true
24 
25     master_volume = -40
26     sound_volume = -40
27     music_volume = -40
28     set_all()
29 
30 func set_all():
31     set_resolution(resolution)
32     set_fullscreen(fullscreen)
33 #   set_brightness(brightness)
34     set_camera_shake(camera_shake)
35 #   set_crt_filter(crt_filter)
36     set_master_volume(master_volume)
37     set_sound_volume(sound_volume)
38     set_music_volume(music_volume)
```

Figura 92: Funcions de Settings

6.2.4. Integer resolution handler

Un problema que s'ha trobat durant el desenvolupament és que l'aspecte dels píxels es distorsionava, és a dir, no es mantenien quadrats. Això succeeix al modificar la resolució del videojoc durant una partida o al moure's el jugador i amb ell la càmera que mostra el videojoc per pantalla, perquè els valors que s'usen per modificar la resolució són del tipus *float.* Per solucionar aquest problema s'ha utilitzat un plugin anomenat Integer resolution handler. Aquest plugin restringeix els valors que modifiquen la resolució a *integers.* Veure Figura 93.

Utilitzant aquest plugin i ajustant les opcions de Godot que controlen com es comporta la resolució de la pantalla, s'ha pogut corregir aquest problema. Les opcions es troben a Project Settings → Display → Window i s'han ajustat a 2d (Mode) i keep_height (Aspect). S'han triat aquests valors perquè, si es juga al videojoc en una pantalla que no té un format de 16:9, s'afegeixen barres negres a les parts inferior i superior en lloc de barres laterals.

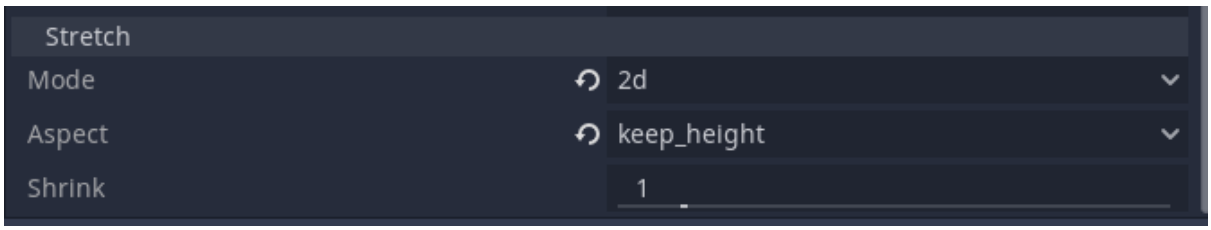

Figura 93: Opcions Stretch

6.3. HUD i Menús

6.3.1. HUD

El HUD està compost per dos nodes principals: *MainContainer*, que conté la informació que sempre es troba present al HUD, i *PopUps*, que conté els elements que es mostren per pantalla durant uns segons en certs moments. Veure Figura 94.

Dins el *MainContainer*, trobem dos *sprites* per gestionar quins *spells* estan actius (*SpellUp* i *SpellDown*), una barra de progress que indica quant manà li queda al jugador (*ManaBar*) i cinc *sprites* per les vides, que poden tenir dues textures diferents segons si aquella vida està disponible o si ja s'ha perdut (*Life1, 2, 3, 4* i *5*). Veure Figura 95.

Dins el *PopUps,* trobem tres nodes que gestionen els missatges segons si es tracta d'un *upgrade* (veure Figura 96), d'una *boss essence* (veure Figura 97) o d'un de tutorial (veure Figura 98).

Per *upgrades* i *boss essence*, hi ha un text que varia segons quin objecte ha recollit el jugador (*Text*) i un botó per continuar la partida (*ContinueButton*). Per als tutorials trobem els textos que indiquen l'acció que cal fer (*Text* i *Text2*) i un *sprite* que mostra la tecla o botó que cal apretar per realitzar l'acció (*ButtonTexture*).

Els nodes *VboxContainer* i *HboxContainer* s'utilitzen per organitzar els nodes fills de manera vertical o horitzontal, respectivament.

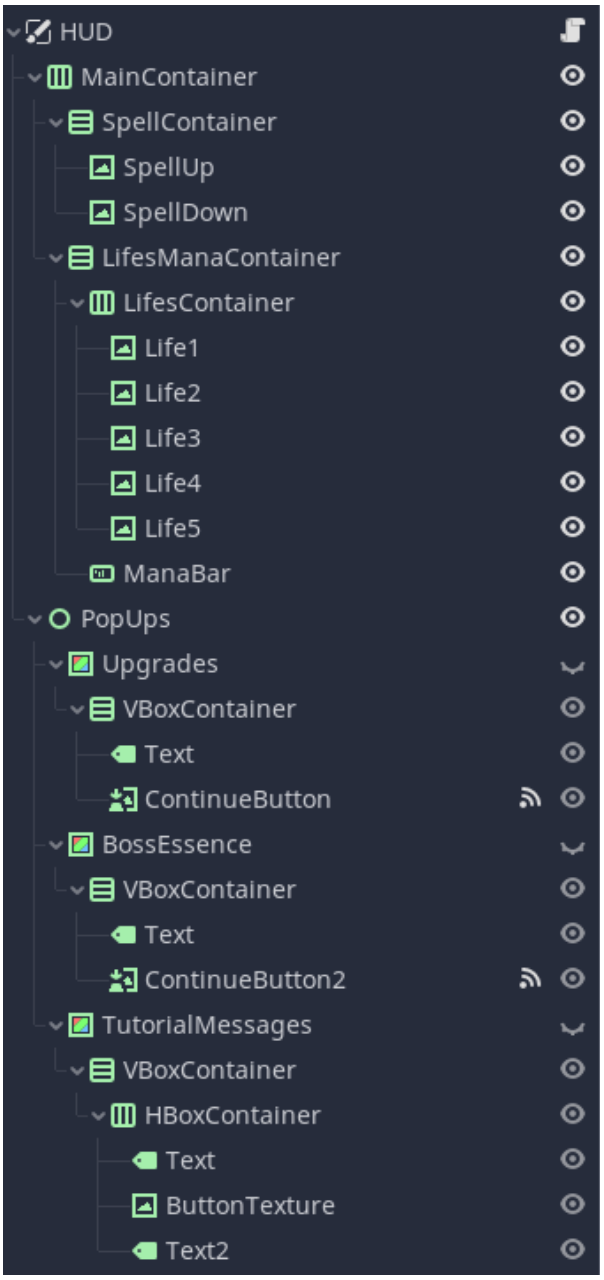

Figura 94: Nodes de HUD

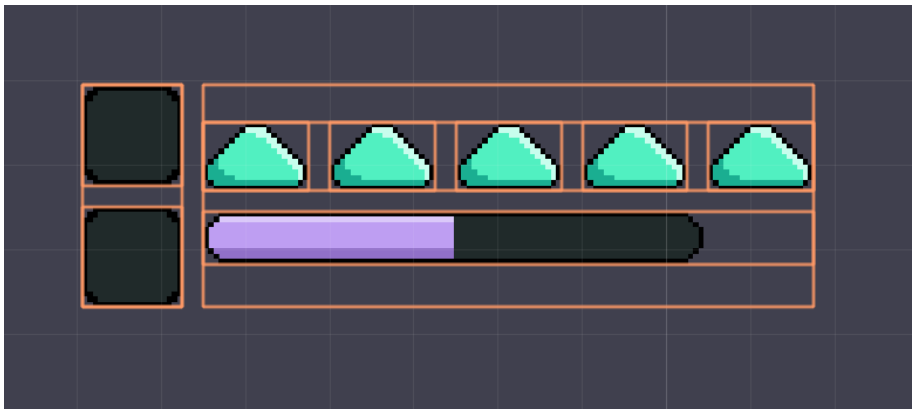

Figura 95: Components de HUD

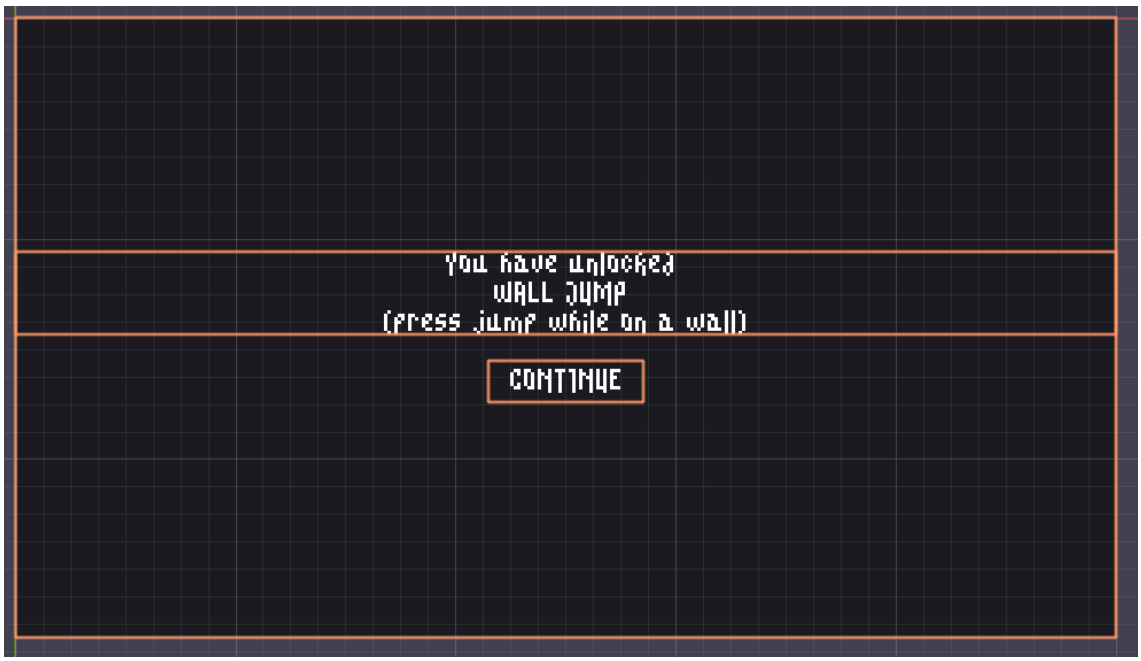

Figura 96: Components de Popups Upgrade

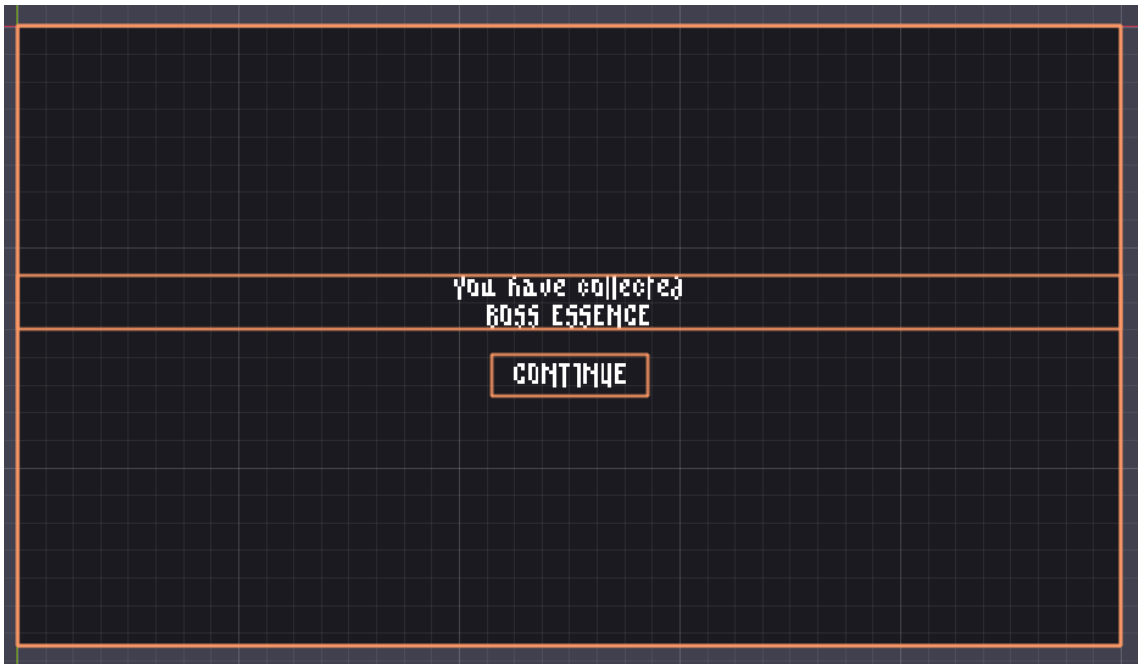

Figura 97: Components de Popups BossEssence

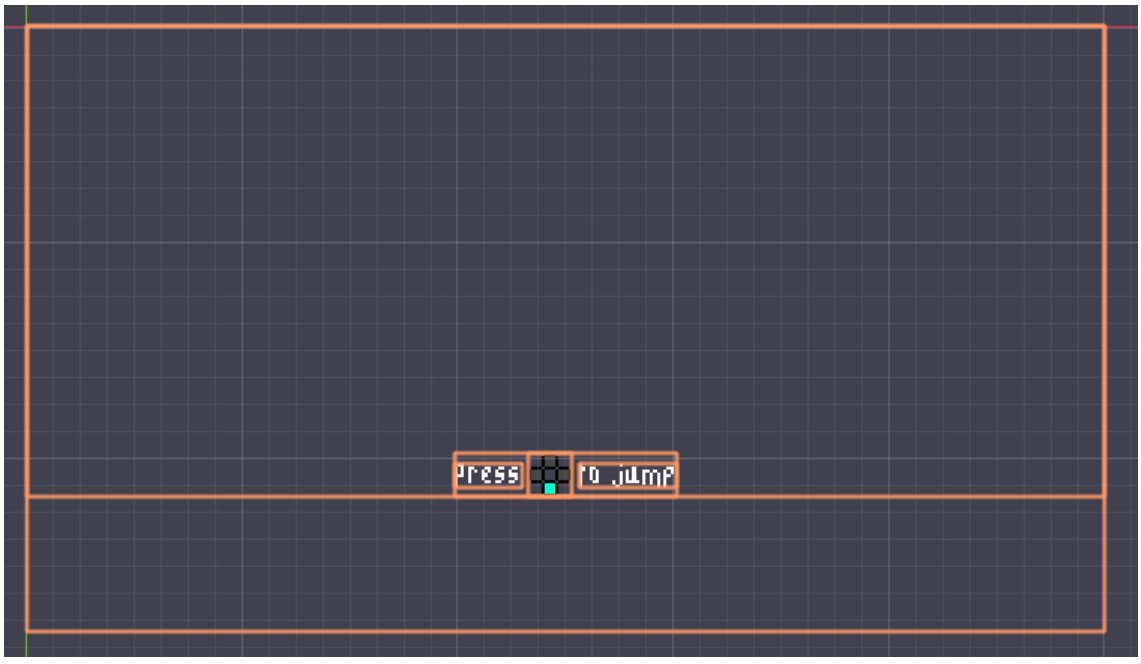

Figura 98: Components de Popups TutorialMessages

6.3.2. Menú inicial

El menú inicial està estructurat en set nodes principals: *Main, Games, Options, VideoOptions, AudioOptions, ControlsOptions* i *Quit.* Veure Figura 99.

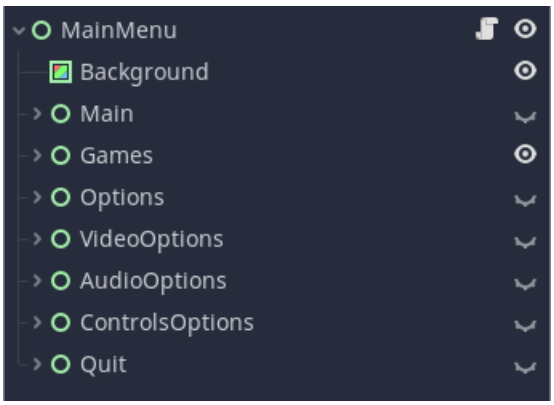

Figura 99: Nodes principals Main Menu

Dins el node *Main*, els princials nodes que trobem són els tres botons que permeten passar als altres submenús. Quan un d'aquests botons és premut, el submenú *Main* es posa no visible i el respectiu submenú es posa visible. Veure Figures 100 i 101.

Figura 100: Nodes del Main

Figura 101: Components de Main

Dins el node *Games* trobem quatre *containers* diferents. Els tres primers (*HboxContainer1, 2* i *3*) són iguals i contenen dos botons, un per iniciar la partida d'aquell perfil i un altre (*HBoxContainer4*) per eliminar la partida guardada d'aquell nivell. Cal comentar que no s'ha implementat el sistema de guardat, però s'han implementat aquests botons pels avanços futurs. L'últim *container* conté el botó per anar enrere i tornar al submenú *Main*. Veure Figures 102 i 103.

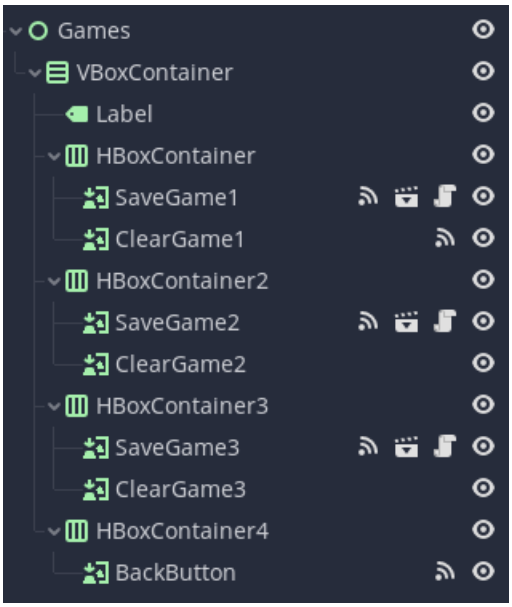

Figura 102: Nodes del Games

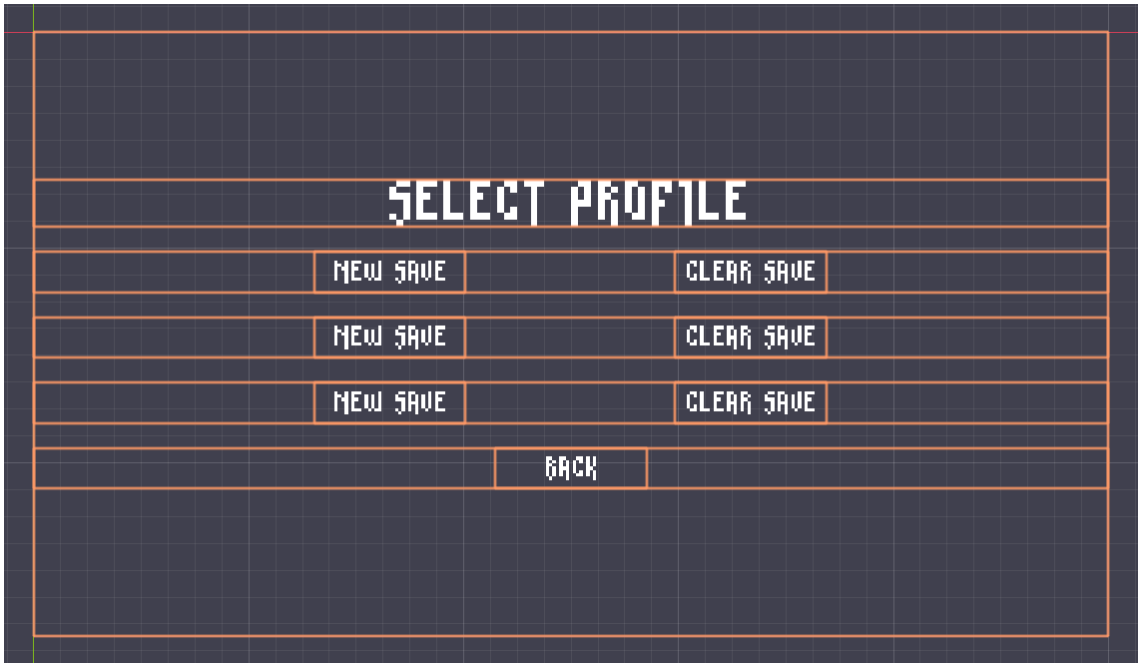

Figura 103: Components de Games

Dins el node *Options*, trobem cinc botons diferents. Tres d'aquests (*VideoButton*, *AudioButton* i *ControlsButton*) porten als submenús per editar els ajustaments de vídeo i audio i als submenús per veure els controls. El quart botó (*ResetButton*), torni als valors per defecte dels ajustaments de vídeo i audio. El cinquè botó (*BackButton*), torna enrere al submenú *Main.* Veure Figures 104 i 105.

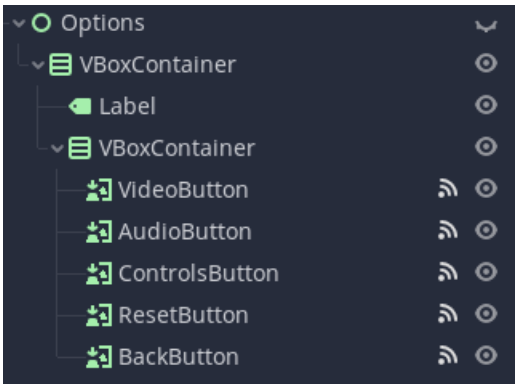

Figura 104: Nodes de Options

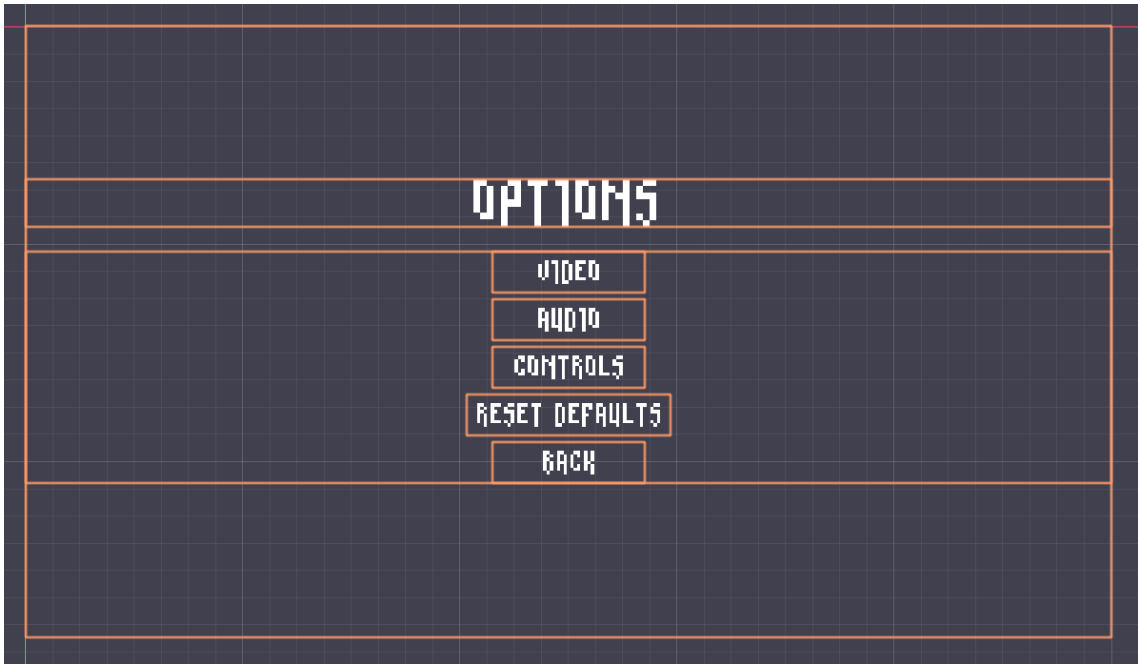

Figura 105: Components de Options

Dins el node *VideoOptions*, els principals nodes són els que permeten modificar els ajustaments de vídeo. Aquests nodes són: *OptionsButton*, que permet seleccionar a partir d'un desplegable la resolució a la qual es vol jugar, *CheckBox1,* que activa o desactiva el mode pantalla completa, *HSlider,* que permet ajustar la brillantor, *CheckBox2,* que permet activar o desactivar la vibració de la càmera, i *CheckBox3,* que permet activar o desactivar el fitre CRT. Cal comentar que les opcions de brillantor i filtre CRT no s'han implementat en aquest projecte, però s'ha implementat els botons pels avanços futurs. Veure Figures 106 i 107.

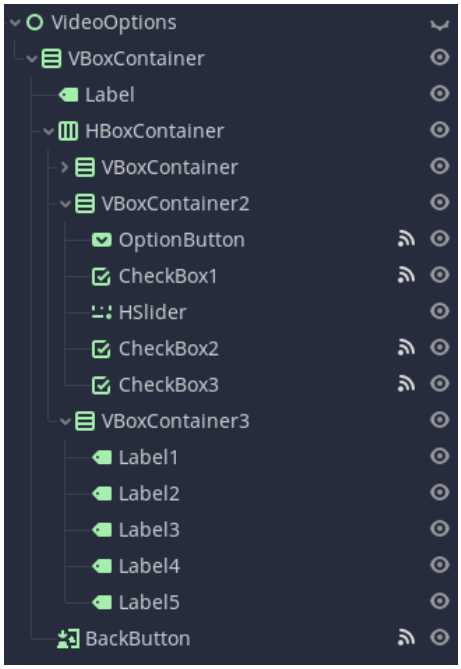

Figura 106: Nodes de VideoOptions

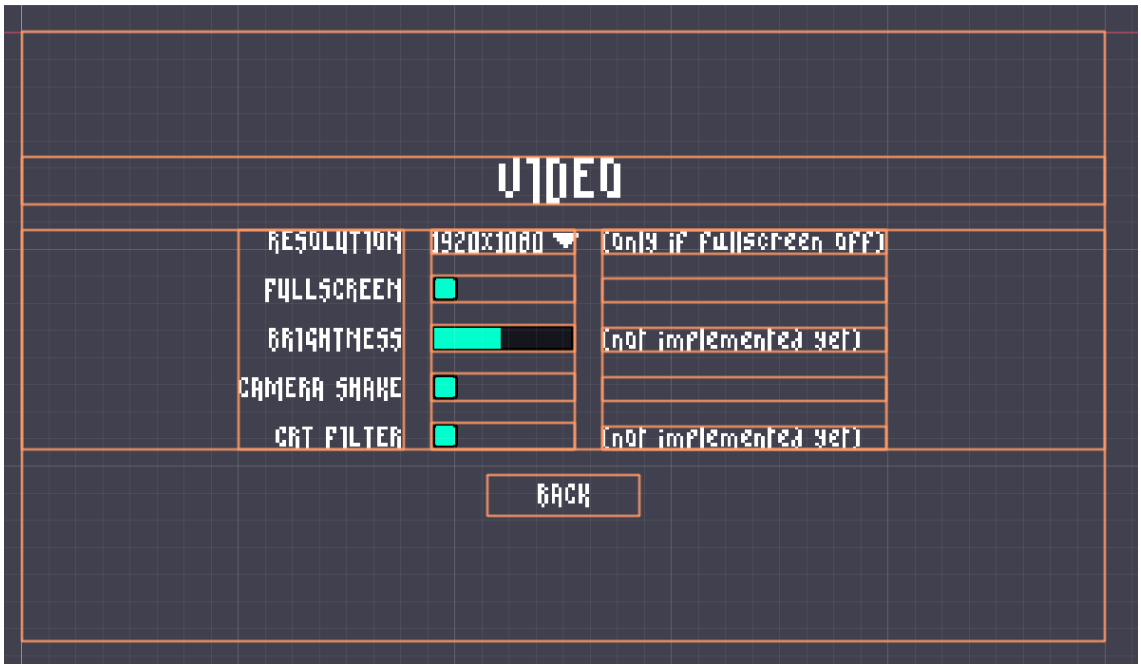

Figura 107: Components de VideoOptions

Dins el node *AudioOptions*, els principals nodes són els que permeten modificar els ajustaments d'audio. Aquests nodes són: *HSLider1, HSLider2* i *HSLider3.* Aquests nodes són sliders que permeten seleccionar el nivell de so general, el nivell dels efectes de so i el nivell de la música. Tot i que per aquest projecte no s'han creat sons o música, s'han implementat aquestes opcions pels avanços futurs. Veure Figures 108 i 109.

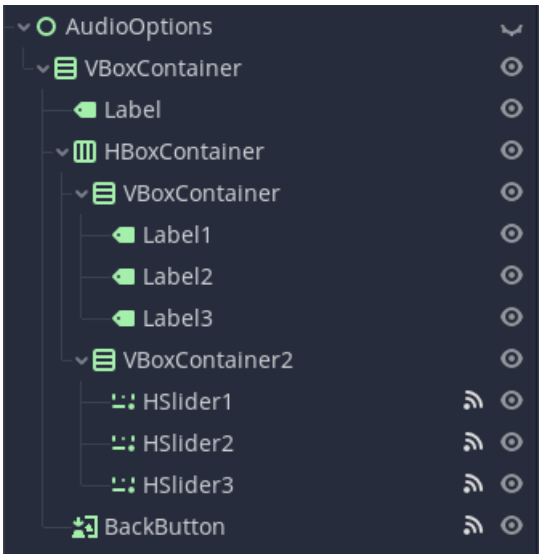

Figura 108: Nodes de AudioOptions

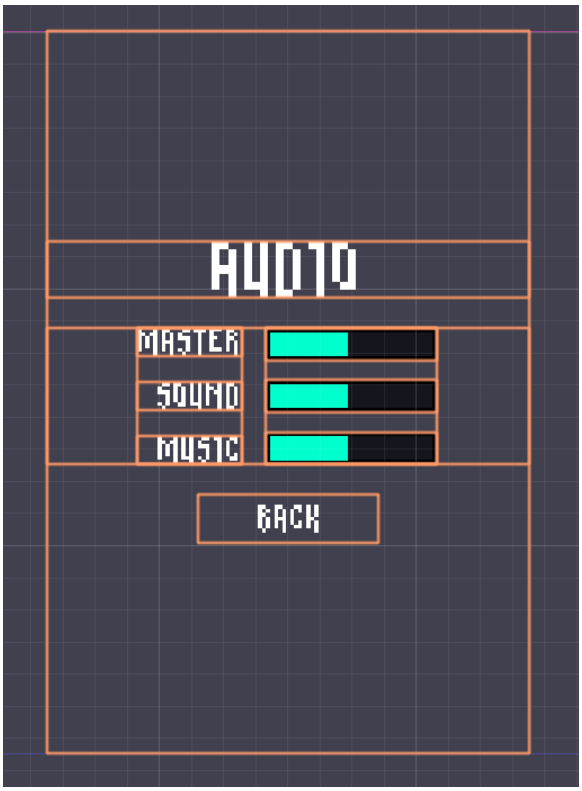

Figura 109: Components d'AudioOptions

Dins el node *Quit*, trobem els dos botons si (*YesButton*) i no (No*Button*). Aquest submenú es mostra per confirmar que el jugador vol sortir del videojoc. Veure Figures 110 i 111.

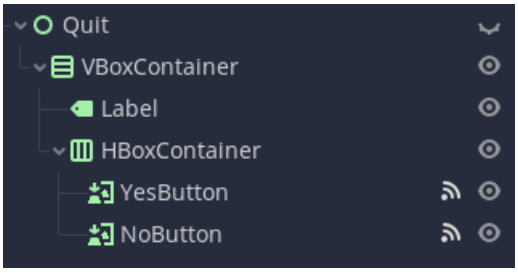

Figura 110: Nodes de Quit

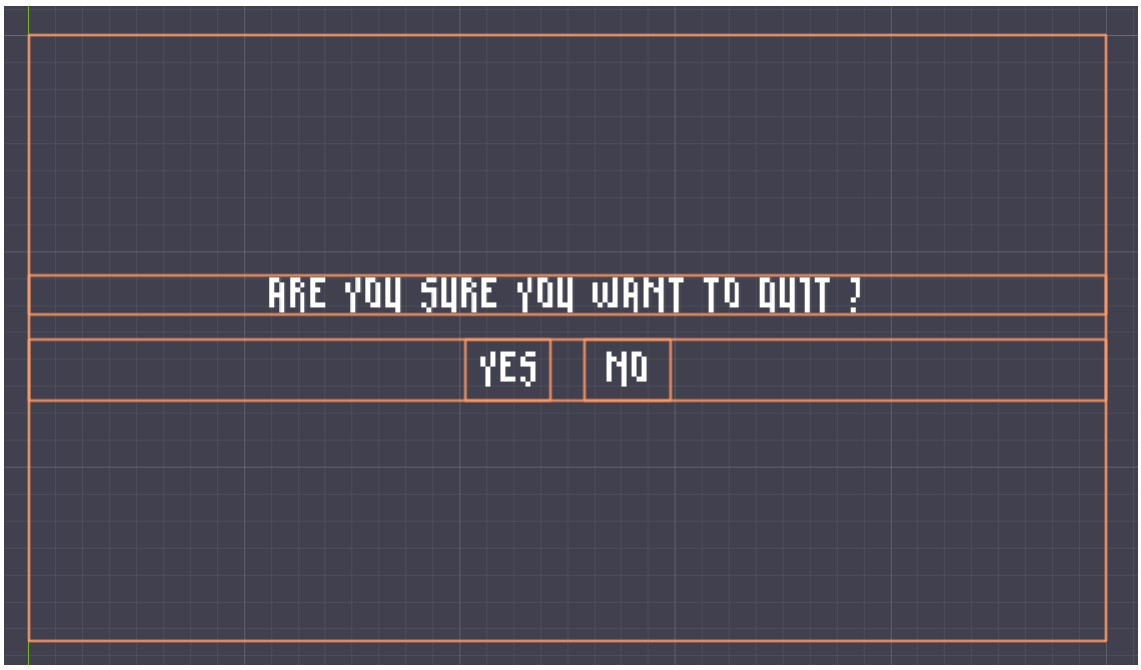

Figura 111: Components de Quit

Dins el node *ControlsOptions*, trobem diferents *containers* que serveixen per organitzar els diferents *labels* i *sprites* que mostren el nom de l'acció i el botó que cal prémer per executar aquesta acció amb comandament i amb teclat. Veure Figures 112 i 113.

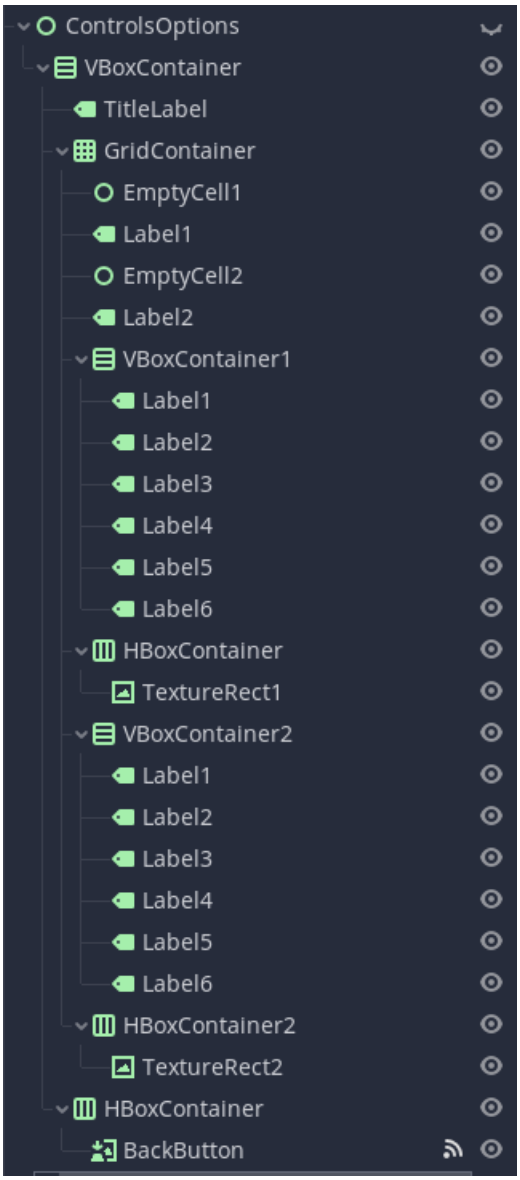

Figura 112: Nodes de ControlsOptions

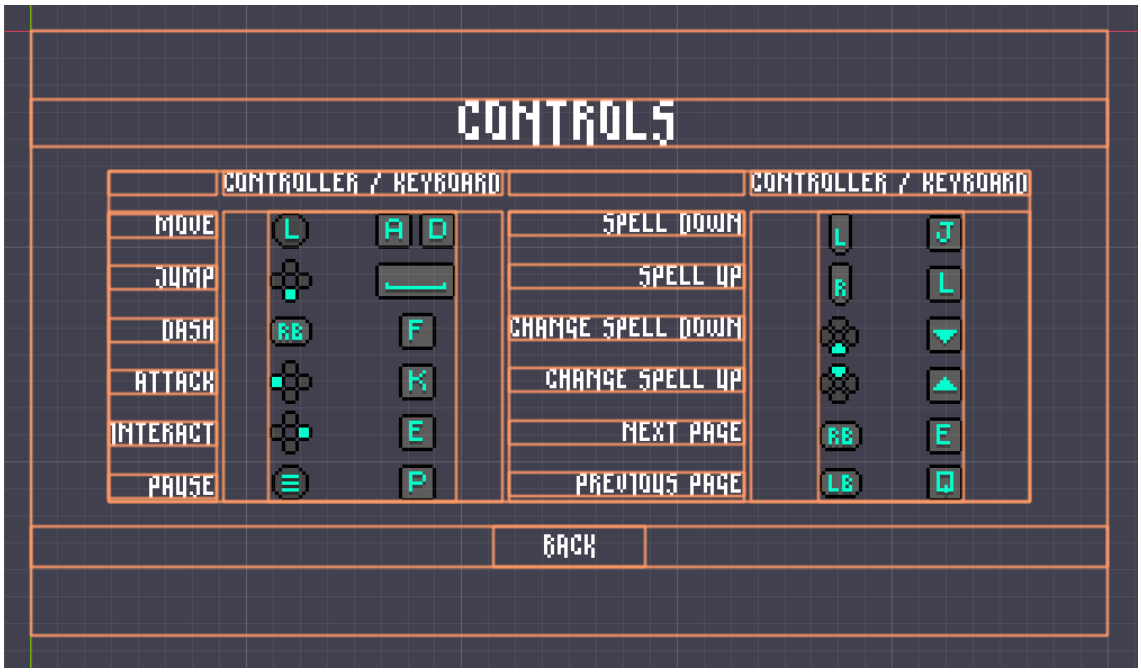

Figura 113: Components de Controls

6.3.3. Menú del jugador

El menú del jugador està estructurat en dos botons (*ButtonLeft* i *ButtonRight*) i set nodes principals: *Map, Inventory, Settings, VideoOptions, AudioOptions, ControlsOptions* i *Quit.* Veure Figura 114. Els dos botons s'usen per navegar entre els tres submenús principals (*Map, Inventory* i *Settings*).

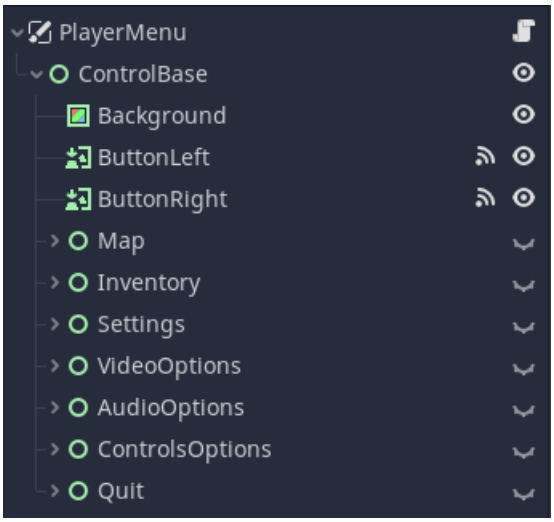

Figura 114: Nodes de PlayerMenu

Els nodes *Settings, VideoOptions, AudioOptions, ControlsOptions* i *Quit* són els mateixos que els que ja s'han comentat a la Secció 6.3.2. El node *Map* conté una imatge esquemàtica del mapa dels nivells desbloquejats (veure Figures 115 i 116). El mapa que es mostra en la implementació d'aquest projecte és el que es mostra a la Figura 66 de l'Apartat 5.7.3.

Figura 115: Nodes de Map

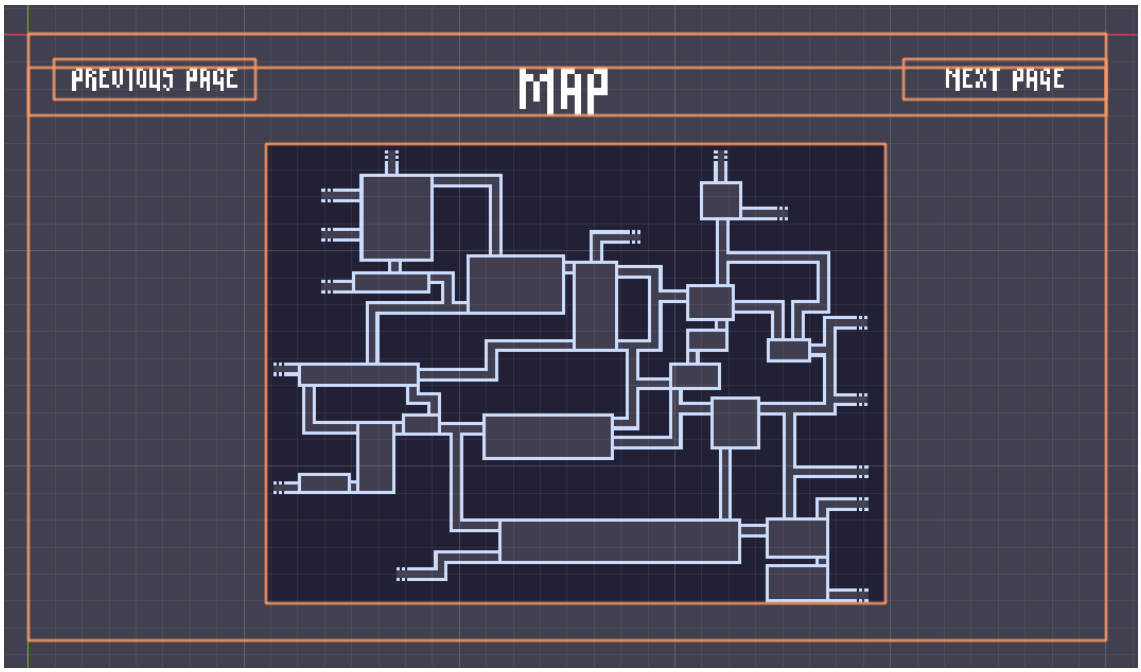

Figura 116: Components de Map

El node *Inventory* té quatre nodes principals: *BossEssences, Spells, Upgrades* i *Description.* Veure Figures 117 i 118.

El node BossEssences és un *grid* (taula on els nodes s'ordenen segons les columnes predefinides). Conté quatre *sprites* que quan la *boss essence* no s'ha obtingut, l'*sprite* està buit, i quan el jugador l'obté l'*sprite,* mostrarà la icona d'aquella *boss essence.*

El node Spells conté dos *containers* iguals. Cada un d'aquests té cinc *sprites*, que mostraran la icona d'aquell *spell* si el jugador l'ha desbloquejat.

El node Upgrades conté un *sprite* per cada un dels possibles upgrades que hi haurà (*Upgrade1, 2, 3, 4* i *5)*. Quan l'*upgrade* no s'ha desbloquejat, l'*sprite* està buit, i quan el jugador el desbloqueja, l'*sprite* mostrarà la icona d'aquell *upgrade.*

El node *Description* conté nodes amb text en els quals es mostra el nom de l'objecte seleccionat (*Title*), de quin tipus és aquest objecte (*Type*) i una petita descripció (*Text*).

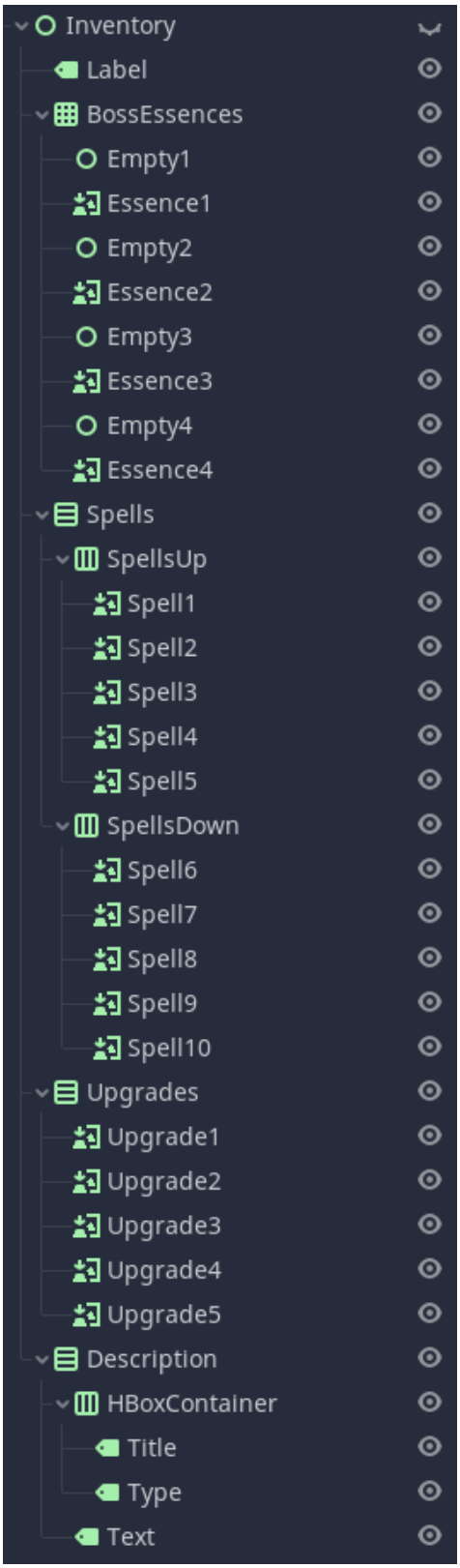

Figura 117: Nodes de Inventory

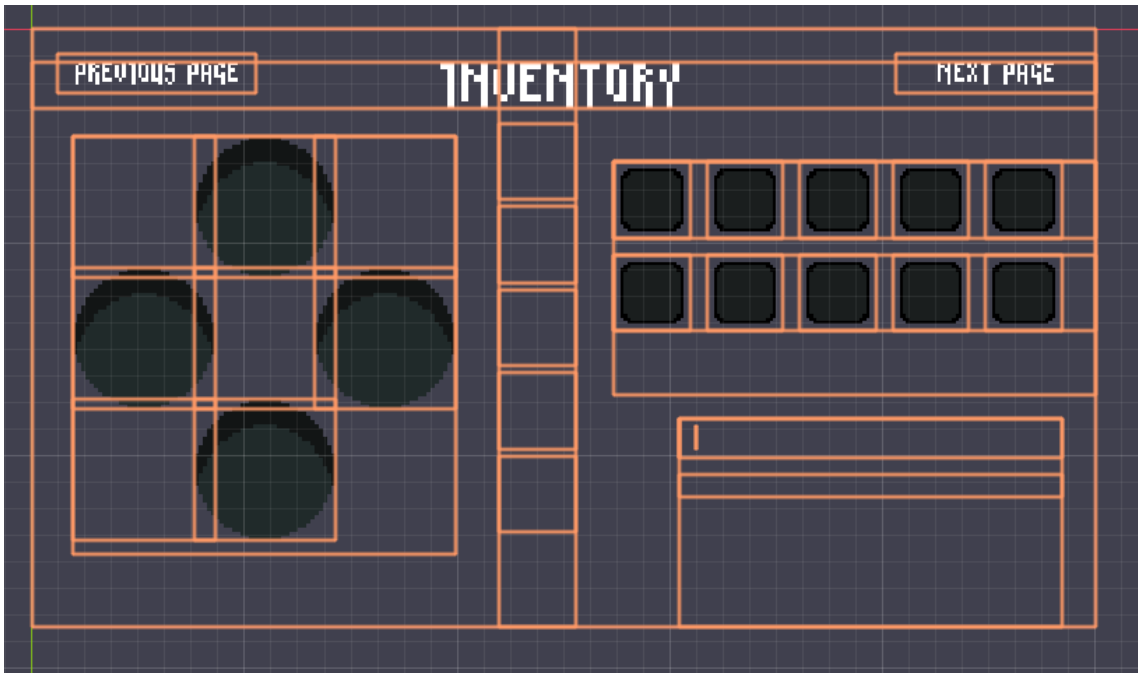

Figura 118: Components de Inventory

6.4. Personatges

6.4.1. Màquines d'estats

Per implementar les màquines d'estats que es trobaran al protagonista, als enemics i al *boss* s'ha usat la implementació que ja s'ha comentat a l'Apartat 4. Aquestes màquines, disposen d'un senyal (*signal state_changed*) per indicar que s'ha canviat d'estat. Les variables principals són l'estat en el que s'ha d'iniciar, que s'ha d'indicar a cada instància d'una màquina (*START_STATE*); un *map* dels estats que té el node que gestionaran (*states_map*); una pila d'estats que s'usarà per apilar estats quan se sap que al canviar a un estat nou s'haurà de tornar a l'anterior (*state_stack*) i l'estat actual (*current_state*). Veure Figura 119.

```
3  signal state_changed(current_state)
4
5  export(NodePath) var START_STATE
6
7  var states_map = {}
8  var states_stack = []
9  var current_state = null
```

Figura 119: Variables de StateMachine

La funció important de la màquina d'estats i la que es cridarà més sovint és *change_state,* que és la funció que es crida per canviar d'un estat a un altre. Primer comprova si hi ha un estat actiu (*_active).* Després surt de l'estat actual i comprova si el nou estat, definit pel *state_name,* és el *previous.* Si és el cas, treu l'estat que està acabant de la pila. Si no, posarà al primer estat de la pila el nou estat. Llavors, assigna el primer estat de la pila com a estat actual i emet el senyal d'estat canviat (*state_changed*). Veure Figura 120.

```
42 func _change_state(state_name):
43     if not _active:
44         return
45     current_state.exit()
46
47     if state_name == "previous":
48         states_stack.pop_front()
49     else:
50         states_stack[0] = states_map[state_name]
51
52     current_state = states_stack[0]
53     emit_signal("state_changed", current_state)
54
```

Figura 120: Funció change_state de StateMachine

A continuació es comentaran les diferents implementacions de la màquina d'estats per a cada un dels personatges que l'utilitzen, explicant cada un dels estats i les transicions entre aquests.

6.4.1.1. Protagonista

Els estats del protagonista són els següents (veure Figura 121 per observar les transicions entre els estats).

- **Idle:** Quan no es detecta cap input. El personatge es queda quiet.
- **Run:** Quan es detecta input lateral i el jugador es troba a terra. El personatge es mou en aquell sentit.
- **Jump:** Quan es detecta un input de salt i el jugador està caient o a terra. El jugador es desplaça cap amunt.
- **Dash:** Quan es detecta un input de dash i el jugador no està fent un *wall slide.*
- **Fall:** Quan el jugador no es troba en contacte amb el terra o amb una paret.
- **WallJump:** Quan el jugador està en un *wall slide* i es detecta un input de salt. El jugador es desplaça cap amunt en diagonal en sentit contrari a la paret.
- **WallSlide:** Quan es detecta input lateral en el mateix sentit de la paret. Ralentitza la caiguda del jugador.
- **DoubleJump:** Quan el jugador es troba a l'aire, ja ha realitzat un salt, ha desbloquejat l'habilitat i es detecta un input de salt. El jugador es desplaça cap amunt, però en menor quantitat que en un salt normal.
- **Attack:** Quan es detecta un input d'atac.
- **Stagger:** Quan el jugador rep dany.
- **Die:** Quan el jugador es queda sense vides.
- **NoMovement:** Serveix per bloquejar el moviment del personatge. S'utilitza quan s'activa un checkpoint, en una transició entre escenes, durant cinemàtiques, etc.
- **Plunge:** Quan es detecta un input de *plunge* i el jugador està a l'aire i no en un *wall slide.*

Els estats Attack, Stagger, NoMovement i Die poden ser entrats des de qualsevol altre estat i són els casos en els quals s'afegirà l'estat a la pila perquè un cop s'acabi es pugui accedir a l'estat anterior desempilant-lo.

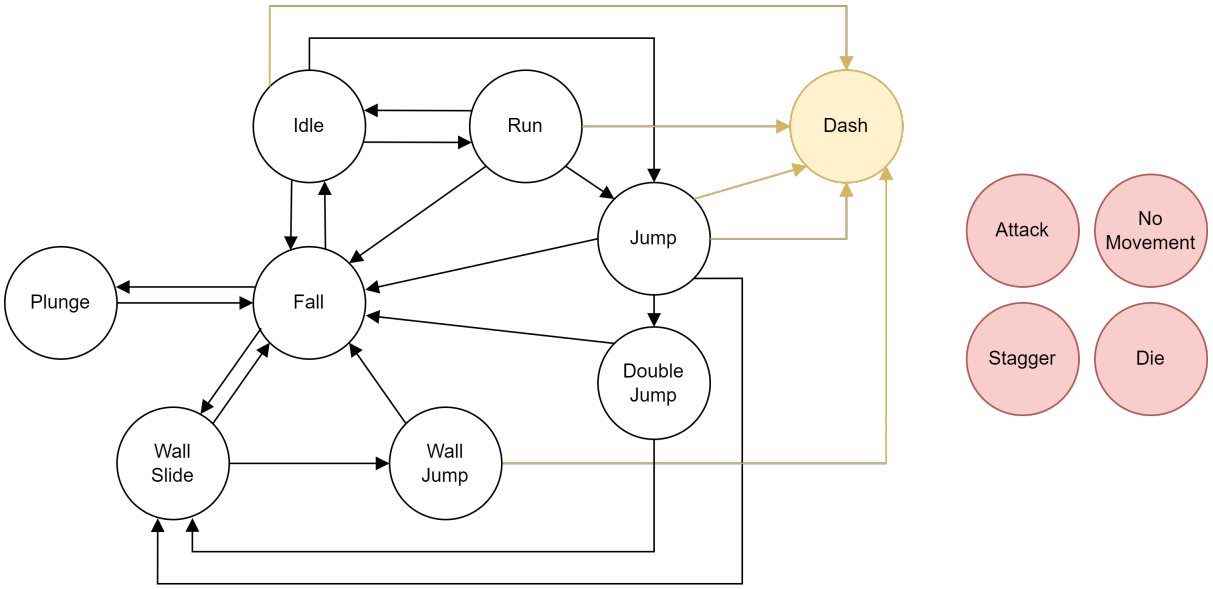

Figura 121: Estats del Protagonista

6.4.1.2. Roamer

Els estats d'aquest personatge són els següents. Veure Figura 122 per observar les transicions entre els estats.

- **Roam:** El personatge es mou durant uns segons predefinits i després canvia de sentit.

Quan el personatge rep dany perd una vida i pot passar a un dels dos estats següents:

- **Stagger:** Si el personatge encara té vides restants, es queda aturdit uns segons i torna a l'estat de *Roam*.
- **Die:** Si el personatge es queda sense vides, es desactiven les col·lisions i al cap d'uns segons s'elimina.

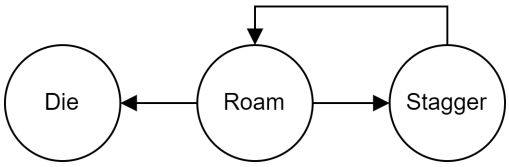

Figura 122: Estats de Roamer

6.4.1.3. Bomber

Els estats d'aquest personatge són els següents. Veure Figura 123 per observar les transicions entre els estats.

- **Roam:** El personatge es mou durant uns segons predefinits i després canvia de sentit.
- **Chase:** El personatge a detectat al jugador, llavors comença a perseguir-lo i cada pocs segons li dispara un projectil.

Quan el personatge rep dany perd una vida i pot passar a un dels dos estats següents:

- **Stagger:** Si el personatge encara té vides restants, es queda aturdit uns segons i torna a l'estat anterior.
- **Die:** Si el personatge es queda sense vides, es desactiven les col·lisions i al cap d'uns segons s'elimina.

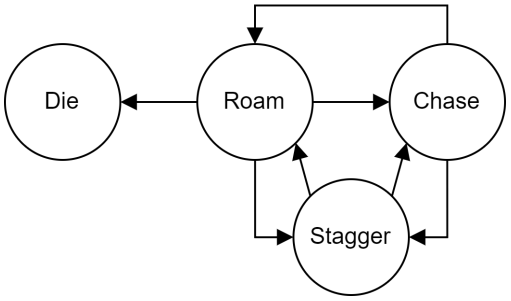

Figura 123: Estats de Bomber

6.4.1.4. Charger

Els estats d'aquest personatge són els següents. Veure Figura 124 per observar les transicions entre els estats.

- **Roam:** El personatge es mou durant uns segons predefinits i després canvia de sentit.
- **Charge:** El personatge ha detectat al jugador, llavors augmenta la velocitat de moviment i es mou cap al jugador.

Quan el personatge rep dany perd una vida i pot passar a un dels dos estats següents:

- **Stagger:** Si el personatge encara té vides restants, es queda aturdit uns segons i torna a l'estat anterior.
- **Die:** Si el personatge es queda sense vides, es desactiven les col·lisions i al cap d'uns segons s'elimina.

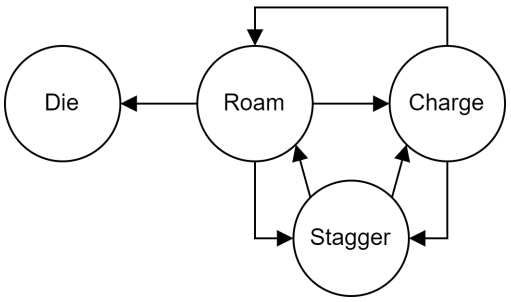

Figura 124: Estats de Charger

6.4.1.5. Boss

Els estats d'aquest personatge són els següents (veure Figura 125 per observar les transicions entre els estats).

- **Prefight:** Estat inicial del personatge. En aquest, es mostra el diàleg de presentació cap al protagonista. (No s'ha implementat)
- **Idle:** Estat en el qual el personatge no realitza cap acció, es succeeix entre estats d'atac. (Implementat parcialment)
- **BaseAttack:** S'activa la *HitboxBasicAttack* mentre es troba en aquest estat. (Implementat completament)
- **Attack1:** El personatge es fa invulnerable mentre dura aquest estat, fins que cau l'última pedra del sostre. (Implementat completament)
- **Attack2:** S'activa la *HitboxAttack2,* es mou cap al sentit on mira el personatge i augmenta la seva alçada. (Implementat completament)
- **Attack3:** Serà un nou atac diferent als ja dissenyats. (No s'ha implementat)

Quan el personatge rep dany perd una quantitat de la barra de vida i pot passar a un dels dos estats següents:

- **Stagger:** Si el personatge encara té barra de vida restant, es queda estabornit uns segons i torna a l'estat anterior. (Implementat parcialment)
- **Die:** Si el personatge es queda sense barra de vida, es desactiven les col·lisions i al cap d'uns segons s'elimina. (Implementat parcialment)

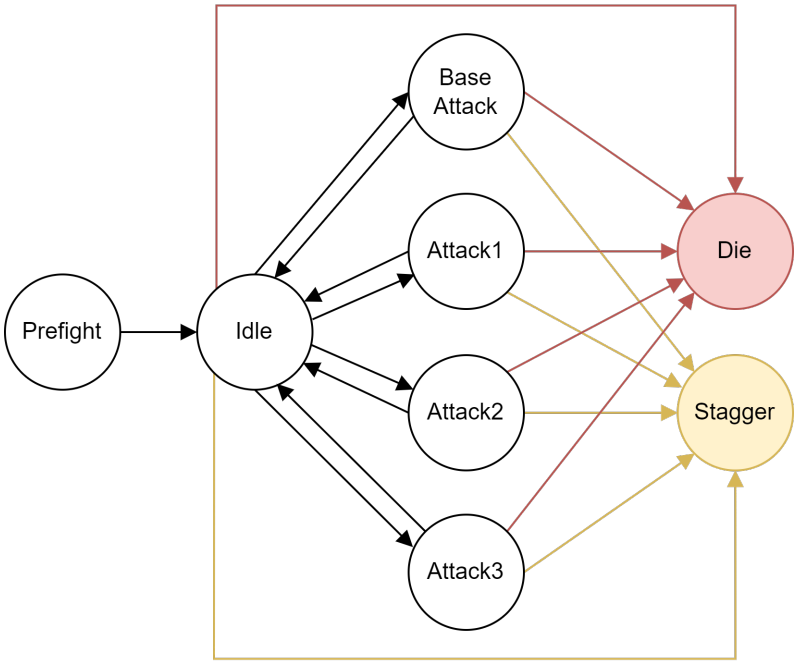

Figura 125: Estats de Boss1

6.4.2. Protagonista

El personatge conté divuit nodes principals (veure Figures 126 i 127), que podem agrupar en les següents categories:

Debug:

- **StateLabel:** Node utilitzat per debugar. Mostrar l'estat actual del personatge.

Player:

- **AnimatedSprite:** Node que gestiona les animacions que cal reproduir en cada moment segons l'estat actual.
- **Spells:** Conté les escenes de cada un dels *spells* (*HealSpell, ShieldSpell, PlungeSpell* i *FireballSpell*), un node que gestiona els *spawns* de les *fireballs* i dos nodes que gestionen des d'on s'han d'*spawnejar* les *fireballs* segons si es disparen cap als costats (*FireballPositionSide*) o cap amunt (*FireballPositionUp*), cap abaix no es poden disparar.
- **PlayerCollision:** Detecta les col·lisions del personatge amb altres personatges o objectes. Treballa conjuntament amb els raycasts.
- **Camera2D:** Càmera principal del videojoc. Segueix al protagonista.
- **StateMachine:** Màquina d'estats pel protagonista. Conté els nodes de tots els estats i gestiona els canvis entre aquests.

Timers:

- **DashTimer:** *Timer* que, un cop s'ha usat el *dash,* s'inicia i envia un senyal quan es pot tornar a utilitzar un altre cop.
- **HurtTimer:** *Timer* que s'inicia quan el jugador rep dany i, mentre està actiu, el jugador és invulnerable. Indica quan pot tornar a rebre dany.
- **SpellTimer:** *Timer* que un cop s'ha usat un *spell,* s'inicia i envia un senyal quan es pot tornar a utilitzar un altre cop.

Raycasts:

- **RayCastTop:** Raycast que detecta les col·lisions superiors. Aquestes col·lisions poden ser amb enemics, parets, plataformes, etc.
- **RayCastGround:** Raycast que detecta si el personatge està en contacte amb el terra (*BaseTileMap* o plataformes) o si ha xocat contra un enemic que es troba a sota. Indica si es pot realitzar un salt.
- **RayCastLeft:** Raycast que detecta les col·lisions laterals de la part esquerra. Aquestes col·lisions poden ser amb enemics, parets, plataformes, etc. Indica si es pot fer *wall slide.*

- **RayCastRight:** Raycast que detecta les col·lisions laterals de la part dreta. Aquestes col·lisions poden ser amb enemics, parets, plataformes, etc. Indica si es pot fer *wall slide.*

Hitboxes i Hurtboxes:

Els quatre nodes següents són els que detecten les col·lisions amb les *hurtboxes* dels enemics o objectes. Quan en detecten una, indiquen quantes vides cal restar si és un enemic i si s'ha de destruir si és un objecte.

- **PlungeHitbox:** Es troba a la part inferior del personatge. S'activa quan realitza l'*spell plunge.*
- **MeleeHitboxSides:** Es troben als laterals del personatge. S'activen quan realitza un atac cap al costat corresponent.
- **MeleeHitboxUp:** Es troba a la part superior del personatge. S'activa quan realitza un atac cap a dalt.
- **MeleeHitboxDown:** Es troba a la part inferior del personatge. S'activa quan realitza un atac cap a baix.
- **Hurtbox:** Node que detecta les col·lisions amb les *hitboxes* dels enemics. Quan en detecta una, indica al jugador que s'ha de restar una vida.

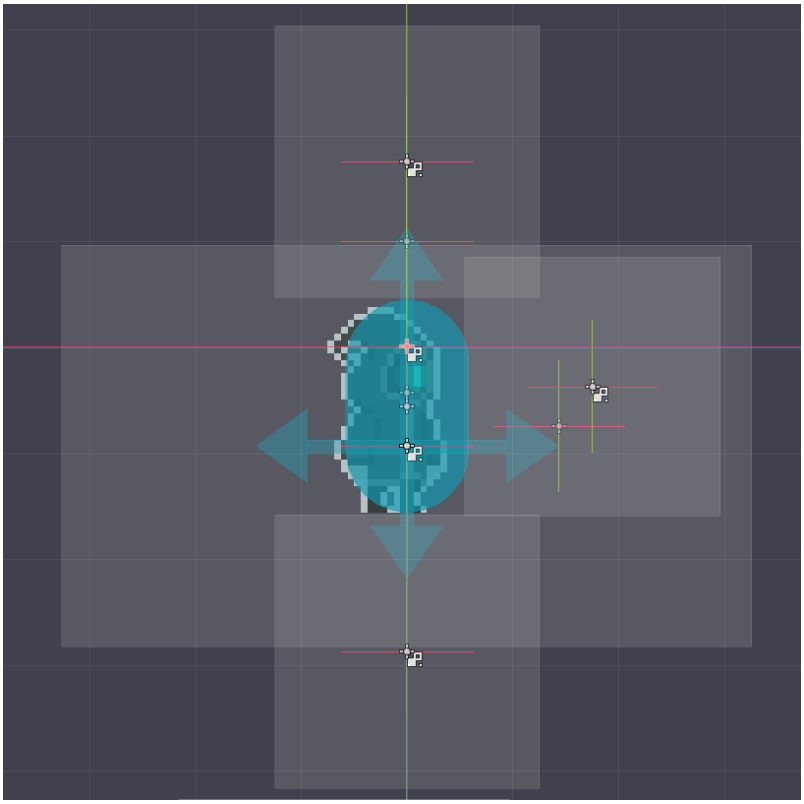

Figura 126: Components de Player

Figura 127: Nodes de Player

Les funcions principals del protagonista són **_input(event)**, **check_abilities()** i **hurt(lifes)**.

A **_input(event)**, primer es comprova si s'està usant un *spell*. Si no s'està usant, hi ha quatre opcions d'input que gestionarà (veure Figura 128).

- ***spell_up:*** segons quin *spell_up* (variable de PlayerInfo) hi ha actiu, i si el jugador l'ha desbloquejat, es llança l'*spell* de *fireball* (*FireballSpell.fire()*) o l'*spell* de plunge (*PlungeSpell.plunge()*).
- ***spell_down:*** segons quin *spell_down* (variable de PlayerInfo) hi ha actiu, i si el jugador l'ha desbloquejat, es llança l'*spell* de *heal* si el jugador té menys vides que el valor màxim possible (*HealSpell.ActivationTimer.start()*) o l'*spell* de *shield* si l'està en contacte amb el terra (*ShieldSpell.shield_up()*).
- ***change_spell_up:*** canvia el *spell_up* actiu, tenint en compte quins ha desbloquejat el jugador
- ***change_spell_down:*** canvia el *spell_down* actiu, tenint en compte quins ha desbloquejat el jugador

```
67  func _input(event):
68      if not using_spell:
69          if event.is_action_pressed("spell_up"):
70              match(PlayerInfo.SPELL_UP):
71                  1: if PlayerInfo.spells_up_learned[PlayerInfo.SPELL_UP-1] == true:
72                      $Spells/FireballSpell.fire()
73                  2: if PlayerInfo.spells_up_learned[PlayerInfo.SPELL_UP-1] == true:
74                      $Spells/PlungeSpell.plunge()
75
76          elif event.is_action_pressed("spell_down"):
77              match(PlayerInfo.SPELL_DOWN):
78                  1:
79                      if (PlayerInfo.spells_down_learned[PlayerInfo.SPELL_DOWN-1] == true
80                      and PlayerInfo.LIFES < 5
81                      and raycast_ground.is_colliding()):
82                          $Spells/HealSpell/ActivationTimer.start()
83                          $StateMachine._change_state("no_movement")
84                  2:
85                      if (PlayerInfo.spells_down_learned[PlayerInfo.SPELL_DOWN-1] == true
86                      and $Spells/ShieldSpell.can_shield
87                      and raycast_ground.is_colliding()):
88                          $Spells/ShieldSpell.shield_up()
89
90          elif event.is_action_pressed("change_spell_up"):
91              PlayerInfo.SPELL_UP += 1
92              if (PlayerInfo.SPELL_UP > PlayerInfo.spells_up_learned.size()
93              or not PlayerInfo.spells_up_learned[PlayerInfo.SPELL_UP-1]):
94                  PlayerInfo.SPELL_UP = 0
95
96          elif event.is_action_pressed("change_spell_down"):
97              PlayerInfo.SPELL_DOWN += 1
98              if (PlayerInfo.SPELL_DOWN > PlayerInfo.spells_down_learned.size()
99              or not PlayerInfo.spells_down_learned[PlayerInfo.SPELL_DOWN-1]):
100                 PlayerInfo.SPELL_DOWN = 0
```

Figura 128: Funció input

A **check_abilities()**, es comproven les variables que determinen si el jugador ha desbloquejat cada una de les habilitats disponibles. Aquesta funció es crida cada cop que es canvia d'escena. Veure Figura 129.

```
 95 func check_abilities():
 96     # CHECK UNLOCKED ABILITIES
 97     if PlayerInfo.DOUBLE_JUMP:
 98         double_jump_unlocked = true
 99     if PlayerInfo.DASH:
100         dash_unlocked = true
101     if PlayerInfo.WALL_JUMP:
102         wall_jump_unlocked = true
```

Figura 129: Funció check_abilities

A **hurt(lifes)**, primer s'assigna *false* a *can_get_hurt*, per a que el jugador no pugui rebre dany. Llavors s'inicia el *hurtTimer* i es resta una vida al jugador. Si aquest s'ha quedat sense vides, es canvia a l'estat *die.* Si encara li queden vides, es comprova si estava usant l'*spell* heal. Si és així, l'atura. Finalment, canvia a l'estat *stagger.* Veure Figura 130.

```
105 func hurt(lifes):
106     can_get_hurt = false
107     hurtTimer.start()
108     PlayerInfo.LIFES = PlayerInfo.LIFES - lifes
109     if PlayerInfo.LIFES < 1:
110         $StateMachine._change_state("die")
111     else:
112         if $Spells/HealSpell.healing:
113             $Spells/HealSpell.end_healing()
114         $StateMachine._change_state("stagger")
```

Figura 130: Funció hurt

6.4.3. Roamer

El personatge conté vuit nodes principals. Veure Figures 131 i 132.

- **StateLabel:** Node utilitzat per debugar. Mostrar l'estat actual del personatge.
- **HurtBox:** Node que detecta les col·lisions amb les *hitboxes* del protagonista. Quan en detecta una, indica al personatge que s'ha de restar una vida.
- **Hitbox:** S'activa quan es detecta una col·lisió amb el *hurtbox* del protagonista. Indiquen quantes vides cal restar-li.
- **CollisionShape2D:** Serveix per detectar les col·lisions amb el *BaseTileMap* i amb els objectes.
- **StateMachine:** Màquina d'estats pel personatge. Conté els nodes de tots els estats i gestiona els canvis entre aquests.
- **MoveTimer:** *Timer* que gestiona els canvis de sentit. Quan s'esgota el temps, canvia el sentit de moviment del personatge i es reinicia el *timer.*
- **DeadTimer:** *Timer* que gestiona el temps que el personatge existeix després de morir. Un cop esgotat el temps, s'elimina.
- **AnimatedSprite:** Node que gestiona les animacions que cal reproduir en cada moment segons l'estat actual.
-

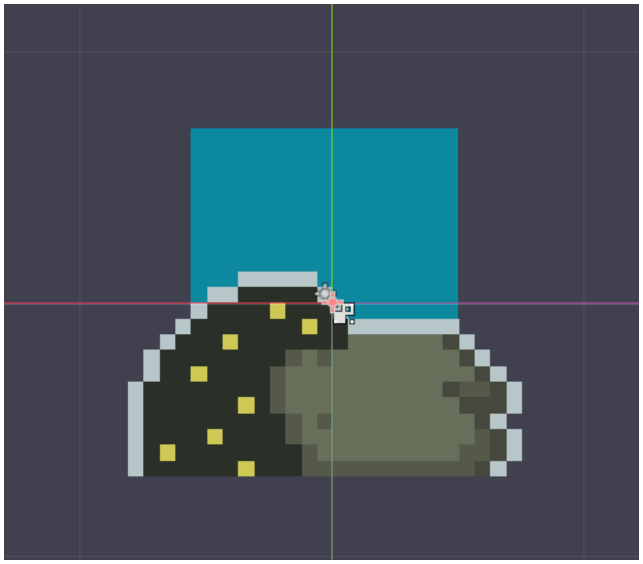

Figura 132: Components de Roamer

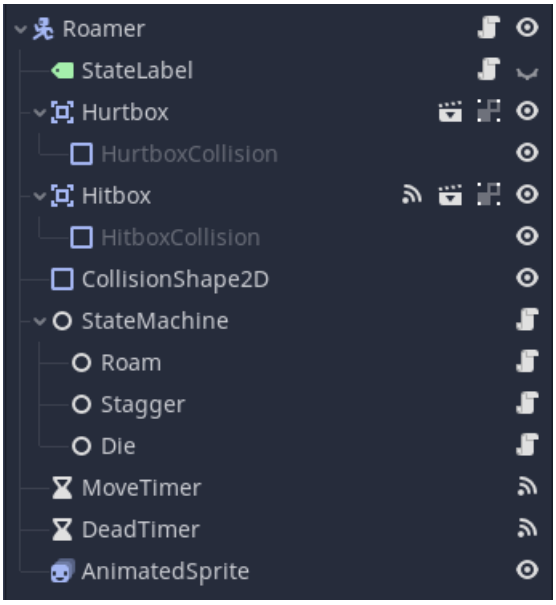

Figura 131: Nodes de Roamer

6.4.4. Bomber

El personatge conté tretze nodes principals. Veure Figures 133 i 134.

- **StateLabel:** Node utilitzat per debugar. Mostrar l'estat actual del personatge.
- **AnimatedSprite:** Node que gestiona les animacions que cal reproduir en cada moment segons l'estat actual.
- **HurtBox:** Node que detecta les col·lisions amb les *hitboxes* del protagonista. Quan en detecta una, indica al personatge que s'ha de restar una vida.
- **Hitbox:** S'activa quan es detecta una col·lisió amb el *hurtbox* del protagonista. Indiquen quantes vides cal restar-li.
- **CollisionShape2D:** Serveix per detectar les col·lisions amb el *BaseTileMap* i amb els objectes.
- **StateMachine:** Màquina d'estats pel personatge. Conté els nodes de tots els estats i gestiona els canvis entre aquests.
- **DetectionArea:** Detecta quan el jugador es troba dins el rang en el qual se'l començarà a perseguir.
- **MoveTimer:** *Timer* que gestiona els canvis de sentit. Quan s'esgota el temps canvia el sentit de moviment del personatge i es reinicia el *timer.*
- **AttackTimer:** Si el personatge està perseguint al jugador, s'inicia i quan s'esgota es dispara un projectil.
- **DeadTimer:** *Timer* que gestiona el temps que el personatge existeix després de morir. Un cop esgotat el temps, s'elimina.
- **ChaseTimer:** Gestiona els canvis de sentit del moviment mentre s'està perseguint al personatge.
- **ProjectileSpawner:** S'encarrega d'spawnejar els projectils
- **Position2D:** Posició en la qual s'spawnegen els projectils.

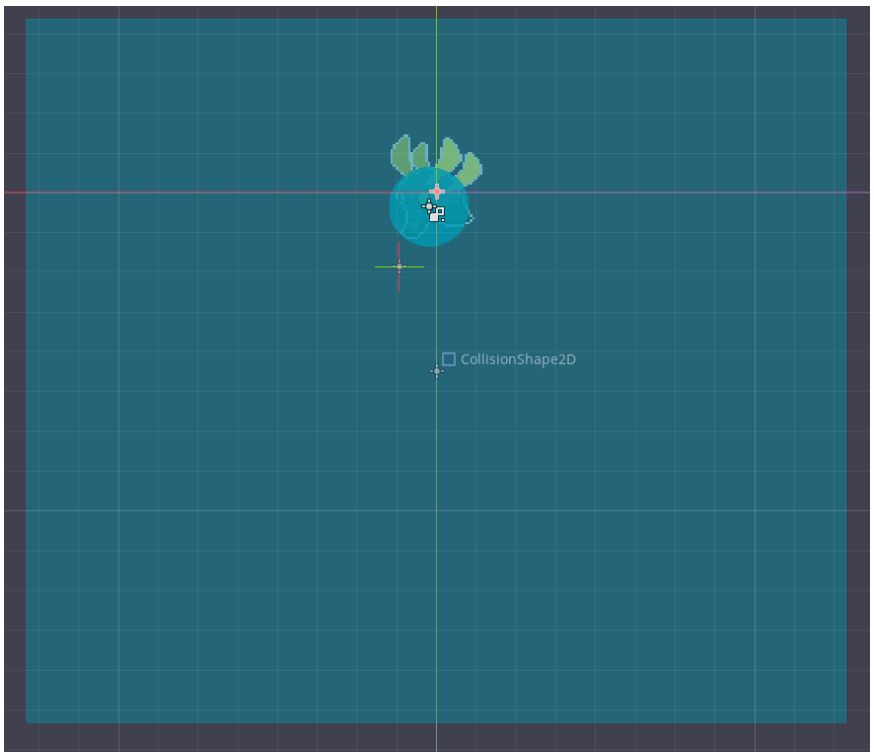

Figura 133: Components de Bomber

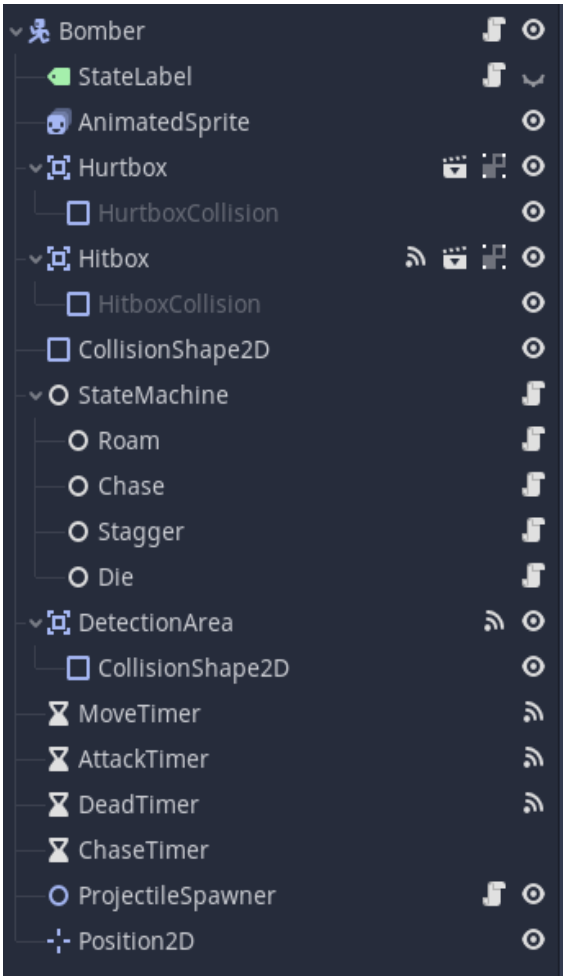

Figura 134: Nodes de Bomber

6.4.5. Charger

El personatge conté tretze nodes principals. Veure Figures 135 i 136.

- **RayCastRight:** Detecta les col·lisions per la part dreta del personatge. Quan en detecta una, canvia el sentit del moviment.
- **RayCastLeft:** Detecta les col·lisions per la part esquerra del personatge. Quan en detecta una, canvia el sentit del moviment.
- **StateLabel:** Node utilitzat per debugar. Mostrar l'estat actual del personatge.
- **AnimatedSprite:** Node que gestiona les animacions que cal reproduir en cada moment segons l'estat actual.
- **CollisionShape2D:** Serveix per detectar les col·lisions amb el *BaseTileMap* i amb els objectes.
- **HurtBox:** Node que detecta les col·lisions amb les *hitboxes* del protagonista. Quan en detecta una, indica al personatge que s'ha de restar una vida.
- **HitBox:** S'activa quan es detecta una col·lisió amb el *hurtbox* del protagonista. Indiquen quantes vides cal restar-li.
- **StateMachine:** Màquina d'estats pel personatge. Conté els nodes de tots els estats i gestiona els canvis entre aquests.
- **DeadTimer:** *Timer* que gestiona el temps que el personatge existeix després de morir. Un cop esgotat el temps, s'elimina.
- **ChargeTimer:** Quan es comença a càrregar contra el jugador s'inicia i quan s'esgota el personatge torna a l'estat de *roam* fins a tornar a detectar al jugador.
- **DetectionArea:** Detecta quan el jugador es troba dins el rang en el qual se'l començarà a perseguir.

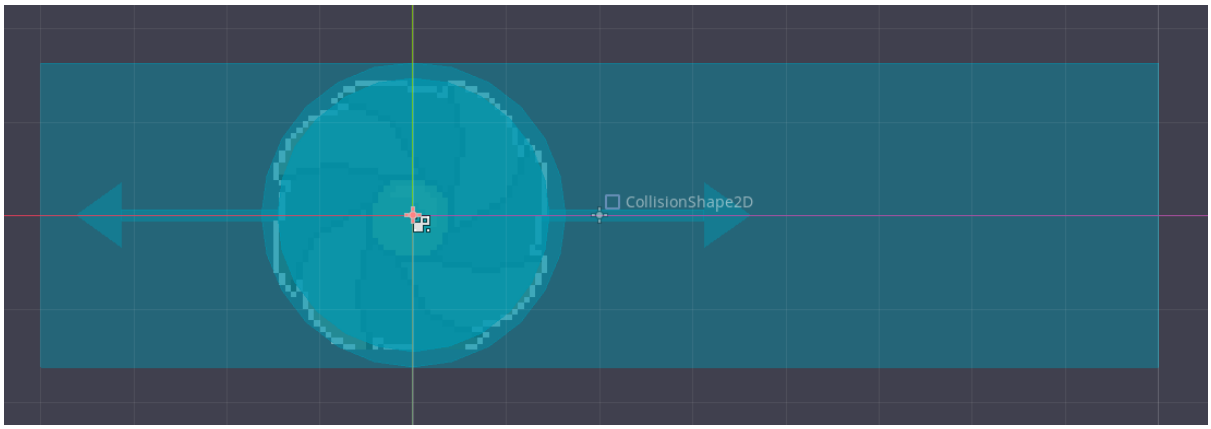

Figura 135: Components de Charger

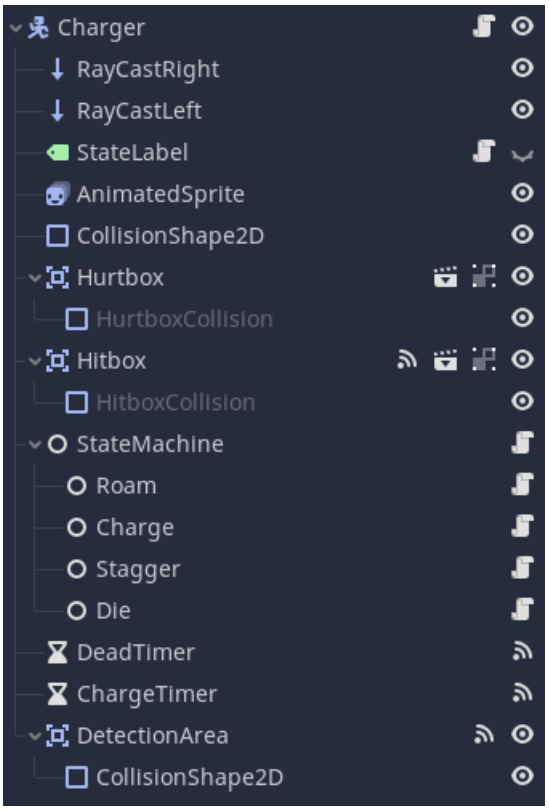

Figura 136: Nodes de Charger

6.4.6. Boss

Cal mencionar que aquest personatge no s'ha pogut implementar completament. Per aquest motiu, en aquesta secció es comentarà si el que s'està explicant ha estat implementat completament, parcialment o s'implementarà en el futur.

El personatge conté deu nodes principals. Veure Figures 137 i 138.

- **StateLabel:** Node utilitzat per debugar. Mostrar l'estat actual del personatge.
- **AttackingTimer:** Quan s'entra a l'estat *idle*, s'inicia el *timer*. Quan s'esgota es canvia d'estat a un dels quatre possibles atacs.
- **AnimationPlayer:** Node que gestiona les animacions especials dels atacs (mou *colliders* o *hitboxes*).
- **AnimatedSprite:** Node que gestiona les animacions que cal reproduir en cada moment segons l'estat actual.
- **CollisionShape2D:** Serveix per detectar les col·lisions amb el *BaseTileMap* i amb els objectes.
- **StateMachine:** Màquina d'estats pel personatge. Conté els nodes de tots els estats i gestiona els canvis entre aquests.
- **HurtBox:** Node que detecta les col·lisions amb les *hitboxes* del protagonista. Quan en detecta una, indica al personatge que s'ha de restar una vida.
- **DetectionArea2D:** Detecta quan el jugador es troba dins l'àrea d'acció del personatge.
- **HitBoxBasicAttack:** S'activa quan el *boss* realitza un atac bàsic (estat *BaseAttack*) i detecta una col·lisió amb el *hurtbox* del protagonista. Indiquen quantes vides cal restar-li.
- **HitBoxAttack2:** S'activa quan el *boss* realitza un atac dos (estat *Attack2*) i detecta una col·lisió amb el *hurtbox* del protagonista. Indiquen quantes vides cal restar-li.

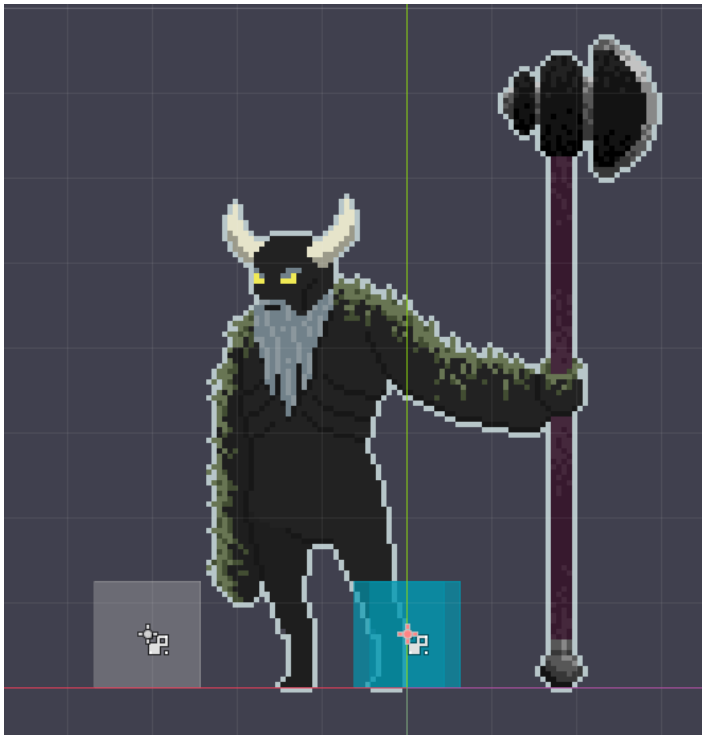

Figura 137: Components de Boss1

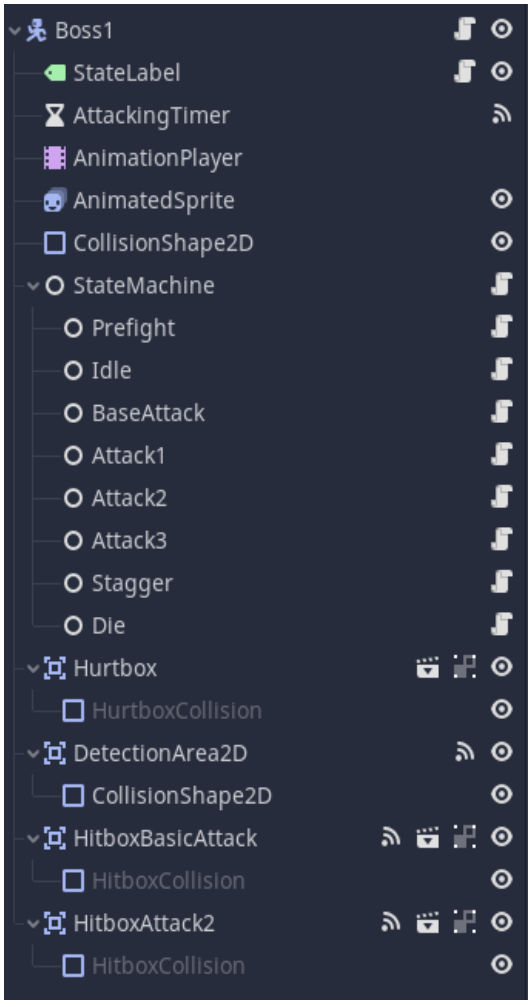

Figura 138: Nodes de Boss1

Els tres enemics (*Roamer*, *Bomber* i *Charger*) i el *Boss* comparteixen les mateixes funcions principals, que són **hurt(damage)** i **_on_Hitbox_area_entered(area)**.

A **hurt(damage)**, que es crida quan el jugador danya a un d'aquests personatges. Resta el *damage* que s'ha passat com a paràmetre i llavors, segons el valor de vida restant, poden succeir dues coses. Si la vida és igual o menor que zero el personatge passarà a l'estat *die*. Si la vida restant és més gran que zero, el personatge passarà a l'estat de *stagger,* i si el jugador no té el manà, al valor màxim se li suma una petita quantitat. L'única diferència entre les dels enemics i la del *Boss* és que a la del *Boss* s'actualitza la vida a la barra de vida d'aquest també. Veure Figura 139.

```
32  func hurt(damage):
33      HEALTH = HEALTH - damage
34      health_bar.value = HEALTH
35      if HEALTH <= 0:
36          $StateMachine._change_state("die")
37      else:
38          if PlayerInfo.MANA < PlayerInfo.MAX_MANA:
39              PlayerInfo.MANA += MANA_RECOVER
40          $StateMachine._change_state("stagger")
```

Figura 139: Funció hurt d'enemics i Boss

A **_on_Hitbox_area_entered(area)** es comprova si l'àrea que ha col·lisionat amb la *hitbox* pot rebre dany (podrà si és el jugador). Si és així, es la seva funció de *hurt* amb el paràmetre *damage* segons el personatge. S'inicialitza el *knockback_vector* segons la direcció entre el personatge i el jugador i el passa a la variable *knockback_direction* del jugador, que farà que el jugador es mogui en sentit contrari al que ha rebut el dany. La variable *hurt_by_enemy* del jugador a true indica que l'ha ferit un enemic. Veure Figura 140.

```
42  func _on_Hitbox_area_entered(area):
43      if area.get_parent().can_get_hurt:
44          area.get_parent().hurt(DAMAGE)
45          knockback_vector = global_position.direction_to(area.get_parent().global_position)
46          area.get_parent().knockback_direction = knockback_vector
47          area.get_parent().hurt_by_enemy = true
```

Figura 140: Funció _on_Hitbox_area_entered

6.4.7. NPC

El personatge conté quatre nodes principals. Veure Figures 141 i 142.

- **Sprite:** Mostra l'sprite del personatge.
- **CollisionShape2D:** Gestiona les col·lisions amb altres objectes.
- **Label:** Mostra el diàleg corresponent segons el nombre d'*scrolls* que té el jugador i si ha completat o no el *trial.*
- **DetectionZone:** Gestiona la detecció del jugador per part del personatge.

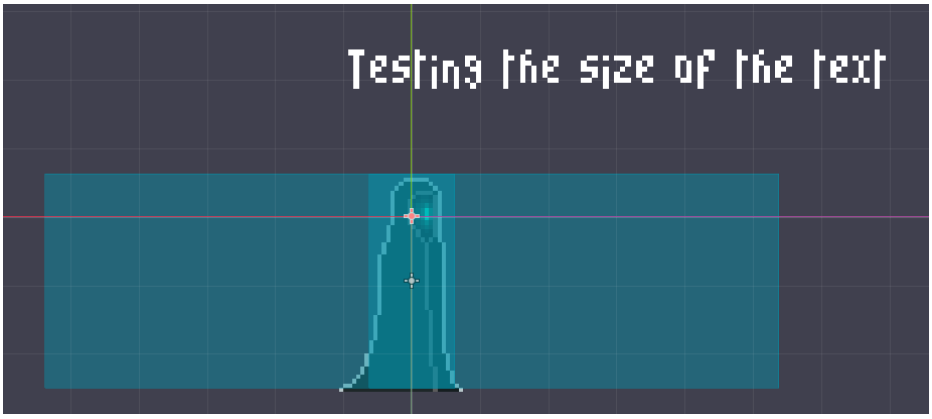

Figura 141: Components de NPC

Figura 142: Nodes del MageNPC

Aquest personatge no utilitza la màquina d'estats comentada a l'Apartat 4.3, ja que només té un estat i solament serveix per orientar al jugador dins les sales de *trials.*

La principal funció d'aquest personatge és **show_text()**. Aquesta funció pot mostrar tres tipus de missatge al *label*. El primer és quan el jugador encara no ha aconseguit els tres *scrolls* necessaris per poder completar el *trial*, el segon quan ja els ha aconseguit i el tercer quan s'ha completat el *trial.*

6.5. Objectes

6.5.1. Boss essence

Aquest objecte conté els següents nodes. Veure Figures 143 i 144.

- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.
- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).

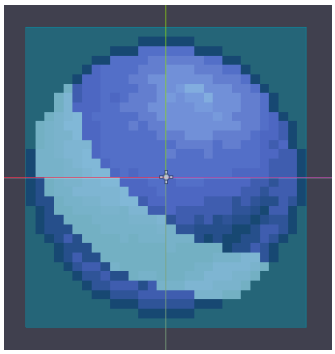

Figura 143: Components de BossEssence

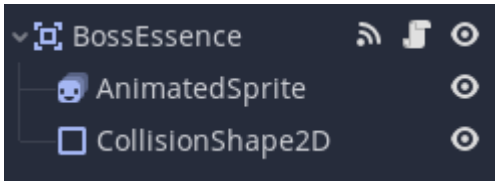

Figura 144: Nodes de BossEssence

La funció principal d'aquest objecte és **_on_BossEssence_body_entered(body)**. En aquesta funció s'actualitza l'array de *PlayerInfo* i el player_menu, i després es mostra per pantalla el pop-up de l'objecte, que en aquest cas mostrarà el nom del Boss i una petita descripció. Veure Figura 145.

```
8  func _on_BossEssence_body_entered(body):
9      PlayerInfo.boss_essences_collected[id-1] = true
10     root = get_tree().get_root()
11     hud = root.get_node("Game").get_node("HUD")
12     player_menu = root.get_node("Game").get_node("PlayerMenu")
13     hud.show_boss_essence_text(id)
14     player_menu.update_boss_essence(id)
15     queue_free()
```

Figura 145: Funció _on_BossEssence_body_entered

6.5.2. Checkpoint

Aquest objecte conté els següents nodes. Veure Figures 146 i 147.

- **CheckpointArea2D:** Detecta quan el jugador entra al rang en el qual pot interactuar amb l'objecte (rectangle blau clar).
- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.
- **Label:** Mostra *Interact* quan el jugador es troba dins el rang d'interacció.

Figura 146: Components de Checkpoint

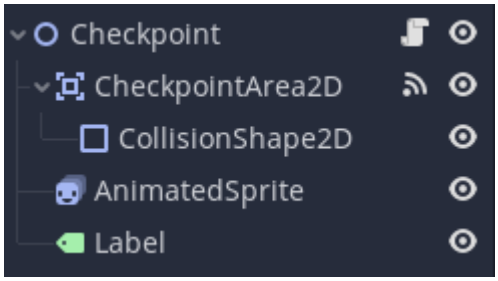

Figura 147: Nodes de Checkpoint

La funció principal d'aquest objecte és **set_checkpoint()**. En aquesta funció, primer s'atura el joc i s'espera un segon. Després, posen tots els checkpoints a no actius, la posició de l'últim checkpoint a *PlayerInfo* i les vides al màxim. Llavors, es reprodueix l'animació del checkpoint omplint-se i ,un cop acabada, es reprèn el joc. Finalment, es posa a actiu el checkpoint actual. Veure Figura 148.

```
22 func set_checkpoint():
23     get_tree().paused = true
24     yield(get_tree().create_timer(1), "timeout")
25     for c in checkpoints.get_children():
26         c.unset_checkpoint()
27     PlayerInfo.LAST_CHECKPOINT = global_position + Vector2(20,0)
28     PlayerInfo.LIFES = PlayerInfo.MAX_LIFES
29     animated_sprite.play("Filling")
30     yield(animated_sprite, "animation_finished")
31     get_tree().paused = false
32     animated_sprite.play("Full")
33     active = true
```

Figura 148: Funció set_checkpoint

6.5.3. Cofre

Aquest objecte conté els següents nodes. Veure Figures 149 i 150.

- **Sprite:** Mostra un dels dos possibles *sprites* de l'objecte, pel cofre obert o tancat.
- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **Upgrade:** Instància de l'*upgrade* que aconseguirà el jugador a l'obrir el cofre. Es troba com a no visible fins que s'obre el cofre.
- **Label:** Mostra *Interact* quan el jugador es troba dins el rang d'interacció.

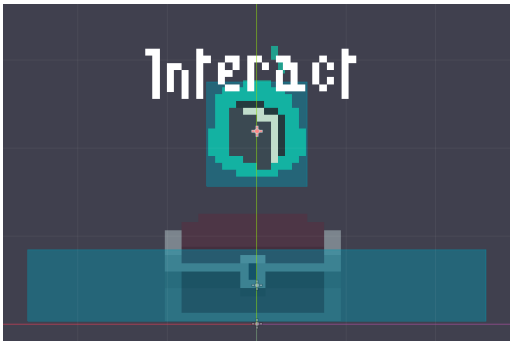

Figura 149: Components de Chest

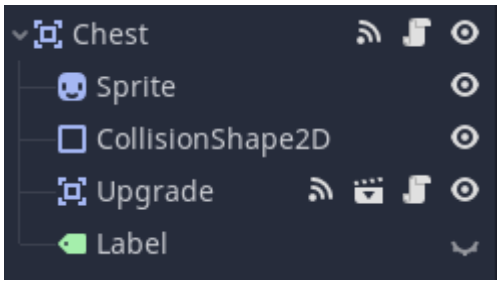

Figura 150: Nodes de Chest

Les funcions principals d'aquest objecte són **_ready()** i **open()**. A **_ready()** s'inicialitzen les variables, es determina el tipus d'*upgrade* que contindrà el cofre i es fa l'*upgrade* no visible. A **open()**, es fa visible l'*upgrade* i es marca el cofre com obert, això fa que el jugador ja no hi pugui interactuar més.

6.5.4. Porta

Hi ha dos tipus de portes: *DoorTall* i *DoorSmall.* L'única diferència entre les dues és la mida. Aquests objectes contenen els següents nodes. Veure Figures 151 i 152.

- **Sprite:** *Sprite* de l'objecte.
- **DoorCollision:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **DoorTriggerLeft:** Detecta quan el jugador pot interactuar amb la porta per obrir-la des de la part esquerra (un dels rectangles blau clar).
- **DoorTriggerRight:** Detecta quan el jugador pot interactuar amb la porta per obrir-la des de la part dreta (un dels rectangles blau clar).
- **AnimationPlayer:** Mou l'*sprite* de l'objecte per simular que s'obre i es tanca la porta.
- **LabelLeft:** Mostra *Interact* a la part esquerra quan el jugador es troba dins el rang d'interacció.
- **LabelRight:** Mostra *Interact* a la part dreta quan el jugador es troba dins el rang d'interacció.

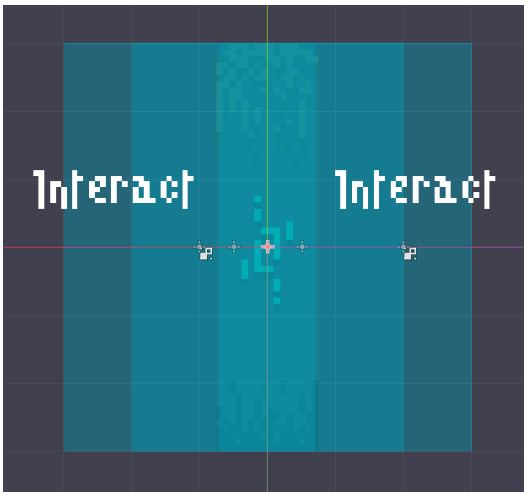

Figura 151: Components de DoorTall

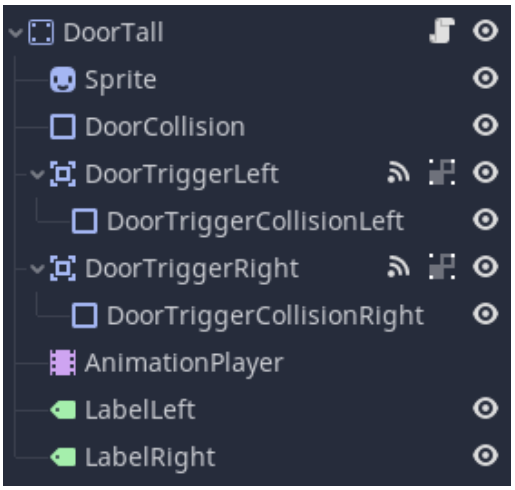

Figura 152: Nodes de DoorTall

Les funcions principals d'aquest objecte són **open()**, **close()**, **block()** i **unblock()**. A **open()**, es reprodueix l'animació d'obrir la porta i un cop acabada es desactiven les col·lisions. A **close()**, es reprodueix l'animació de tancar la porta i un cop acabada s'activen les col·lisions. A **block()**, es desactiven les col·lisions de *DoorTriggerLeft* i *Right*, per evitar que el jugador pugui interactuar amb la porta. A **unblock()**, s'activen les col·lisions de *DoorTriggerLeft* i *Right*, perquè el jugador pugui interactuar amb la porta.

6.5.5. Fireball cannon

Aquest objecte conté els següents nodes. Veure Figures 153 i 154.

- **FireballSpawner:** S'encarrega d'spawnejar les *fireballs* en la direcció correcta.
- **FirePosition:** Posició on les *fireballs* spawnejaran.
- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.

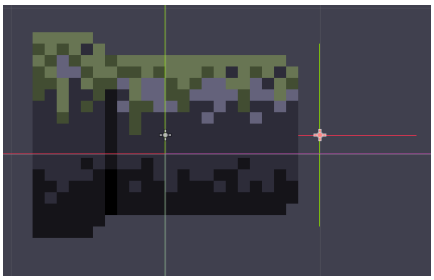

Figura 153: Components de Fireball Cannon

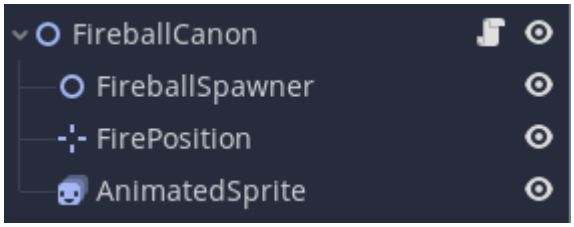

Figura 154: Nodes de FireballCannon

La funció principal d'aquest objecte és **fire()**. En aquesta funció, es passa per paràmetre la direcció en la qual s'ha de disparar. Primer, es crea una nova instància de projectil. Llavors, s'inicialitzen les variables necessàries i es crida el mètode *start* del projectil (el que passa les variables inicialitzades a la instància). Després, s'afegeix la nova instància com a fill de l'escena actual. Finalment, es reprodueix l'animació de disparar i un cop acabada es reprodueix la d'*idle.* Veure Figura 155.

```
16 func fire(direction):
17     var new_projectile = projectile.instance()
18     new_projectile.SPEED = 140
19     new_projectile.can_hurt_player = true
20     new_projectile.DURATION = 10
21     new_projectile.start(fire_position.global_transform, direction)
22     get_parent().get_parent().get_parent().call_deferred("add_child", new_projectile)
23     animated_sprite.play("Fire")
24     yield(animated_sprite, "animation_finished")
25     animated_sprite.play("Idle")
```

Figura 155: Funció fire

6.5.6. Palanca

Aquest objecte conté els següents nodes. Veure Figures 156 i 157.

- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.
- **DurationTimer:** *Timer* que gestiona el temps que la palanca es manté activada.
- **Label:** Mostra *Interact* quan el jugador es troba dins el rang d'interacció.

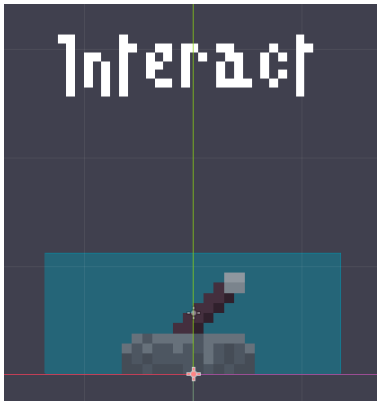

Figura 156: Components de Lever

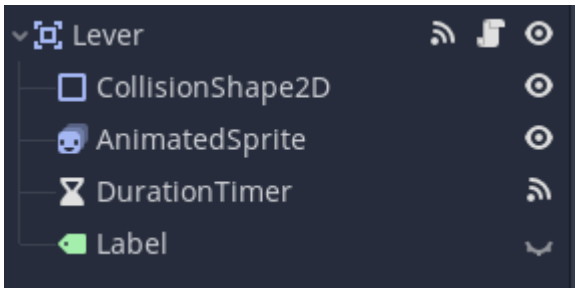

Figura 157: Nodes de Lever

La funció principal d'aquest objecte és **activate()**. En aquesta funció, primer es fa no visible el *label* que mostra *Interact*. Després, es reprodueix una animació segons si el *lever* és vertical (es troba en una paret) o horitzontal (es troba a terra). Llavors, s'inicia el *duration_timer* i es desactiven els *shields_obstacles* associats a la palanca, que ho indica la variable *link_id.* També hi ha una funció contrària anomenada **deactivate()**. Veure Figura 158.

```
28 func activate():
29     label.visible = false
30     if vertical:
31         animated_sprite.play("VerticalDown")
32     else:
33         animated_sprite.play("HorizontalLeft")
34     activated = true
35     duration_timer.start()
36     if link_id != 0:
37         for shield in shields_container.get_children():
38             if shield.id == link_id:
39                 shield.deactivate()
```

Figura 158: Funció activate

6.5.7. Plataformes

Hi ha dos tipus de plataformes: *Platform2x1* i *Platform4x1.* L'única diferència entre les dues és la mida. Aquests objectes contenen els següents nodes. Veure Figures 159 i 160.

- **Sprite:** *Sprite* de l'objecte.
- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).

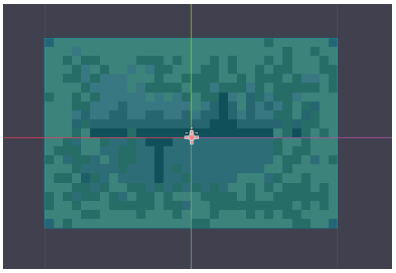

Figura 159: Components de Platform2x1

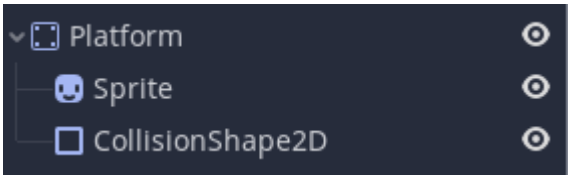

Figura 160: Nodes de Platform

6.5.8. Scroll

Aquest objecte conté els següents nodes. Veure Figures 161 i 162.

- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **Sprite:** *Sprite* de l'objecte.
- **Label:** Mostra *Interact* quan el jugador es trobar dins el rang d'interacció.

Figura 161: Components de Scroll

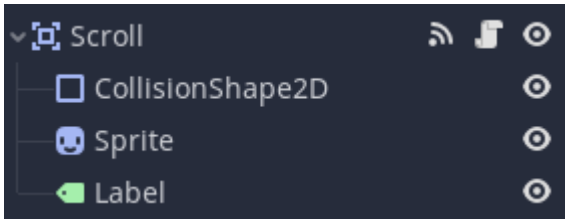

Figura 162: Nodes de Scroll

La funció principal d'aquest objecte és **collect()**. En aquesta funció, primer es desactiva el *label* de l'objecte. Després, segons la id de l'*scroll,* se suma un al comptador d'*scrolls* corresponent de *PlayerInfo*. Finalment, s'elimina l'objecte. Veure Figura 163.

```
20 func collect():
21     $Label.visible = false
22     match(id):
23         1: PlayerInfo.scrolls_shield += 1
24         2: PlayerInfo.scrolls_fireball += 1
25         3: PlayerInfo.scrolls_plunge += 1
26     queue_free()
```

Figura 163: Funció collect

6.5.9. Shield obstacle

Aquest objecte conté els següents nodes. Veure Figures 164 i 165.

- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.
- **DurationTimer:** Gestiona la duració de l'estona que es manté l'objecte desactivat.

La fletxa vermella que es veu a la Figura X indica la direcció de la col·lisió. El jugador pot moure's lliurement de dreta a esquerra però xocarà amb la *CollisionShape2D* en l'altre sentit.

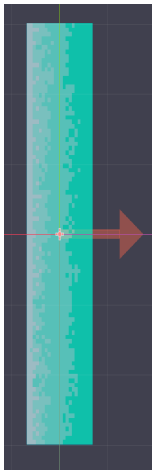

Figura 164: Components de ShieldObstacle

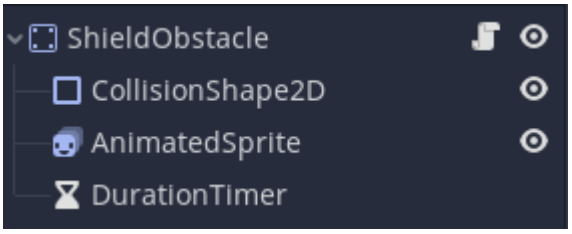

Figura 165: Nodes de ShieldObstacle

Les funcions principals d'aquest objecte són **activate()** i **deactivate()**. A **activate()**, s'activen les col·lisions de l'objecte i es fa visible l'*animated_sprite*. A **deactivate()**, es desactiven les col·lisions de l'objecte i es fa no visible l'*animated_sprite*. Aquestes dues funcions es criden des de les palanques respectives quan aquestes són activades i el *shield_obstacle* hi està relacionat.

6.5.10. Punxes

Aquest objecte conté els següents nodes. Veure Figures 166 i 167.

- **Sprite:** *Sprite* de l'objecte.
- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).

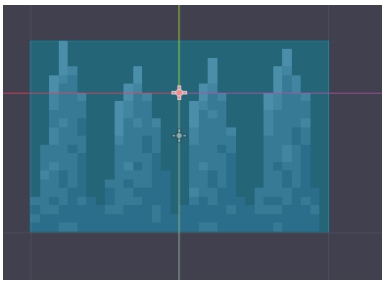

Figura 166: Components de Spikes

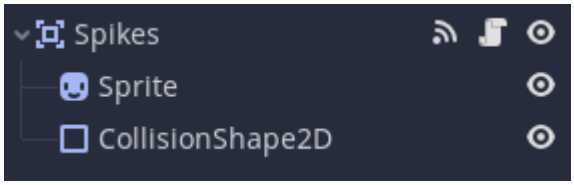

Figura 167: Nodes de Spikes

A les escenes, les punxes que formen part d'un mateix obstacle són instanciades dins un node anomenat SpikeGroup. Aquest node conté diferents *spikes* i una posició, que serà la posició on respawnejarà el jugador si col·lisiona amb alguna de les punxes de l'obstacle. Veure Figures 168 i 169.

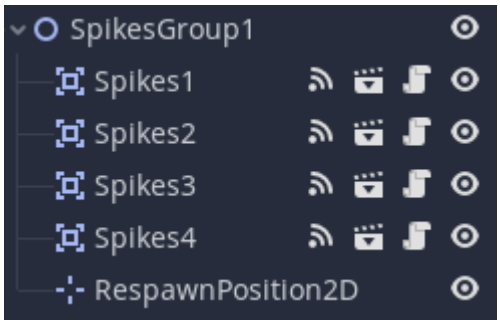

Figura 168: Nodes de SpikeGroup

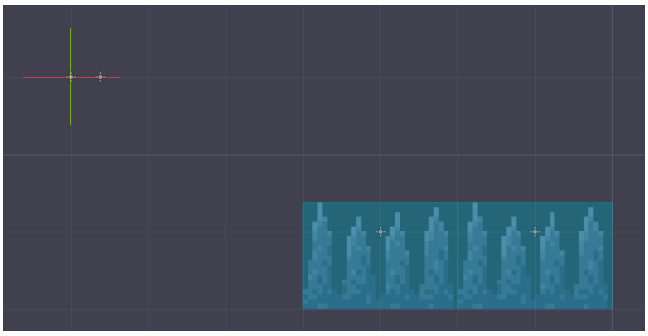

Figura 169: Components de SpikesGroup

La funció principal d'aquest objecte és **_on_Spikes_area_entered(area)**. En aquesta funció, es comprova si l'àrea que ha col·lisionat amb les punxes pot rebre dany (podrà si és el jugador). Si és així, es mou l'àrea a la posició de respawn del grup de punxes i es crida la funció de *hurt*. La variable *hurt_by_enemy* del jugador indica si l'ha ferit un enemic o un obstacle. Veure Figura 170.

```
6  func _on_Spikes_area_entered(area):
7      if area.get_parent().can_get_hurt:
8          area.get_parent().global_position = respawn_position
9          area.get_parent().hurt(damage)
10         area.get_parent().hurt_by_enemy = false
```

Figura 170: Funció _on_Spikes_area_entered

6.5.11. Upgrade

Aquest objecte conté els següents nodes. Veure Figures 171 i 172.

- **CollisionShape2D:** Detecta les col·lisions amb el jugador (rectangle blau clar).
- **AnimatedSprite:** Node que gestiona les animacions de l'objecte que cal reproduir.

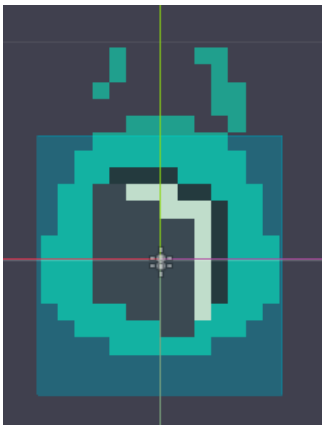

Figura 171: Components de Upgrade

Figura 172: Nodes de Upgrades

Les funcions principals d'aquest objecte són **_ready()** i **_on_Upgrade_body_entered()**. A **_ready()** es connecta el senyal que avisa al jugador que s'ha recollit un *upgrade.* A **_on_Upgrade_body_entered()** es crida quan el jugador recull l'objecte. Es comprova quin *id* té l'*upgrade* i s'actualitzen les variables relacionades amb aquell *upgrade*. Si aquest és una nova *ability*, es cridarà a **new_ability()**. En canvi, si és un nou *spell* es cridarà **new_spell()**. Veure Figura 173.

La funció **new_ability()** actualitza el player_menu amb la nova habilitat. La funció **new_spell()** actualitza el player_menu i les variables de PlayerInfo relacionades amb aquell *spell.*

```gdscript
func _on_Upgrade_body_entered(_body):
    match(id):
        1:
            PlayerInfo.WALL_JUMP = true
            new_ability(1)
            hud.show_upgrade_text(1)
        2:
            PlayerInfo.DASH = true
            new_ability(2)
            hud.show_upgrade_text(2)
        3:
            PlayerInfo.DOUBLE_JUMP = true
            new_ability(3)
            hud.show_upgrade_text(3)
        10: new_spell(true, 10)
        11: new_spell(true, 11)
        12: new_spell(true, 12)
        13: new_spell(true, 13)
        14: new_spell(true, 14)
        15: new_spell(false, 15)
        16: new_spell(false, 16)
        17: new_spell(false, 17)
        18: new_spell(false, 18)
        19: new_spell(false, 19)
    emit_signal("upgrade_collected")
    queue_free()
```

Figura 173: Funció _on_Upgrade_body_entered

6.5.12. Parets

Hi ha dos tipus de parets: *HiddenWall* i *DestructibleWall.* L'única diferència entre les dues és que la *DestructibleWall* al ser colpejada només es destrueix i la *HiddenWall* a part de destruir-se també revelar el *HiddenPath* que té associat. Aquests objectes contenen els següents nodes. Veure Figures 174 i 175.

- **Sprite:** *Sprite* de l'objecte.
- **CollisionShape2D:** Detecta les col·lisions amb el jugador (un dels rectangles blau clar).
- **Area2D:** Detecta les col·lisions amb les *hitboxes* del jugador (un dels rectangles blau clar).

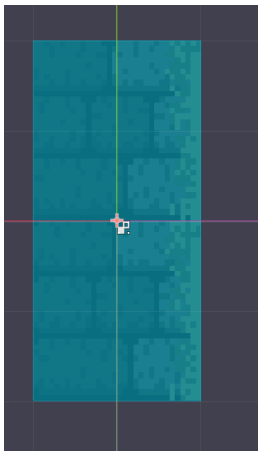

Figura 174: Components de HiddenWall

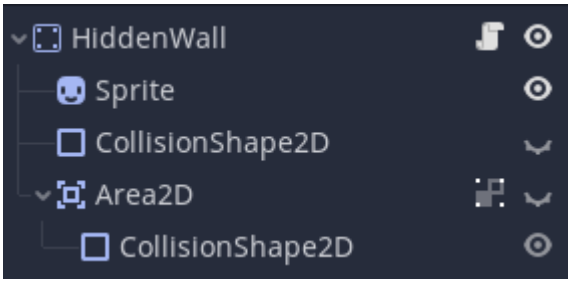

Figura 175: Nodes de HiddenWall

Les funcions principals d'aquest objecte són **reveal()** i **reveal_path()**. A **reveal()**, es busca el path d'entre tots els de l'escena amb el que coincideixen els *ids* del *path* i de la paret i es crida la funció **reveal(path)**. En aquesta funció es crida la funció **clear()** del *tilemap (path),* que l'elimina. Veure Figura 176.

```
22 func reveal():
23     for path in hidden_paths:
24         if actual_id == id:
25             reveal_path(path)
26         actual_id += 1
27
28
29 func reveal_path(path):
30     path.clear()
```

Figura 176: Funcions reveal i reveal_path

6.6. Proves

Durant el desenvolupament del projecte, s'ha anat testejant habitualment les noves mecàniques o zones implementades, per comprovar que el funcionament és el correcte. Distingim dues maneres diferents de realitzar aquestes proves.

6.6.1. Proves a l'editor

Aquestes proves es realitzen directament dins el motor Godot. Permet testejar una escena concreta o el funcionament complet des de l'inici del videojoc. Els principals avantatges d'aquesta manera de testejar són la rapidesa i la simplicitat de l'execució, ja que es realitzen dins l'editor i no cal exportar el projecte, i la possibilitat de visualitzar elements de *debug*, per comprovar col·lisions, estats de personatges, etc. Tot i això, tenen un desavantatge important: alguns errors permeten jugar correctament al videojoc a l'editor, però aquest no funcionaria correctament amb l'executable i possiblement faria aturar-ne l'execució. Per aquesta raó es realitzen les proves següents.

6.6.2. Proves amb l'executable

Aquestes proves es realitzen menys sovint, quan s'ha completat un part important d'una zona, d'una mecànica, d'un objecte, etc. Són més costoses de fer quant a temps invertit, ja que abans de poder realitzar les proves cal exportar el projecte. Una altra raó per la qual s'ha d'invertir-hi més temps és que cal testejar des de l'inici del videjoc per assegurar que tot el que ja funcionava segueix funcionant. Cal destacar que Godot permet guardar diferents perfils d'exportació, cosa que simplifica notablement el procés.

Un cop exportat el projecte, es fa una partida d'inici a final i es van anotant els diferents errors que es detecten per solucionar-los posteriorment. A la Figura 177f següent, es poden observar les opcions d'exportació que permet Godot.

Figura 177: Opcions d'exportació de Godot

6.6.3. Proves amb persones externes

Les proves comentades anteriorment sempre les ha realitzat el desenvolupador del projecte, però cal tenir una visió externa del videojoc. La principal raó és que, al realitzar les proves havent desenvolupat i implementat tot el videojoc, es coneix en gran detall el funcionament de les mecàniques, el comportament dels enemics, la disposició de les zones, etc. Quan algú extern al projecte, que no disposa de coneixement ni informació previs, realitza les proves, es pot comprovar si el disseny és correcte, si s'entèn que s'ha de fer en cada moment, si la dificultat està ben balancejada, etc.

Per aquestes raons, s'han realitzat proves amb dos jugadors, de diferent habilitat i gustos en videojocs, i s'han analitzat els temps i els intents necessaris per a completar cada un dels nivells (Tutorial i Rotting Castle, que en separarem les sales secundàries). També s'han desat els temps i intents al realitzar les proves internes per tenir una referència.

Idealment, les proves externes les realitzarien centenars, o fins i tot milers, de jugadors per poder cobrir totes les possibilitats de trobar errors i analitzar diferents estils de jugador. Però, per aquest projecte, no disposem dels recursos necessaris per dur a terme aquestes proves.

A les taules següents (veure Taules 6 i 7) es mostren els resultats de les proves realitzades.

	Temps Tutorial	**Intents Tutorial**	**Temps Rotting Castle**	**Intents Rotting Castle**
Desenvolupador	2 min	1	~20 min	4
Jugador 1	5 min	1	~35 min	11
Jugador 2	6 min	1	~45 min	15

Taula 6: Taula comparació proves desenvolupador i jugadors, zones principals

	Temps sales *upgrades*	**Intents sales *upgrades***	**Temps sales *trials***	**Intents sales *trials***
Desenvolupador	3 min	2	6 min	3
Jugador 1	6 min	3	9 min	3
Jugador 2	8 min	5	12 min	3

Taula 7: Taula comparació proves desenvolupador i jugadors, sales secundàries

Com podem observar a les taules anteriors (Taules 6 i 7), com era d'esperar, els jugadors tarden més temps i necessiten més intents per superar els diferents nivells. Però l'objectiu principal d'aquestes proves era comprovar si la duració i la dificultat dels nivells era correcta i després d'analitzar els resultats es pot determinar que sí.

7. Resultats

7.1. Assoliment dels objectius

Un cop finalitzat aquest projecte, es pot dir que el nivell d'assoliment dels objectius proposats a l'inici és satisfactori. A continuació es comproven cada un els objectius i si aquest s'ha assolit.

- S'ha après a utilitzar el motor Godot i el llenguatge GDScript de manera exhaustiva.
- S'ha après a utilitzar en profunditat el software Aseprite i les seves funcionalitats.
- S'ha dissenyat completament la part estètica del videojoc, tots els elements han estat creats i no s'ha agafat cap element extern.
- S'han dissenyat i implementat les mecàniques del jugador.
- S'ha implementat la intel·ligència artificial dels enemics.
- S'han desenvolupat les interaccions del jugador amb l'escenari i tots els seus components.
- S'han desenvolupat els menús i interfícies.

L'únic objectiu que no s'ha assolit per complet és implementar la intel·ligència artificial del *Boss* final de la primera zona, del que només se n'ha pogut implementar una part. Tota la resta d'objectius s'ha complert satisfactòriament, alguns fins i tot amb més extensió que la que s'havia previst al definir les tasques a l'inici del projecte.

7.2. Escenaris finals

A continuació, s'analitzaran els dissenys finals de cada una de les sales:

7.2.1. Tutorial

La disposició d'aquest nivell implica un avanç lineal del jugador. S'ha dissenya així perquè el jugador pugui anar aprenent les mecàniques bàsiques en l'ordre correcte i per a que sigui més simple de progressar mentre s'inicia i s'adapta al videojoc. Veure Figura 178.

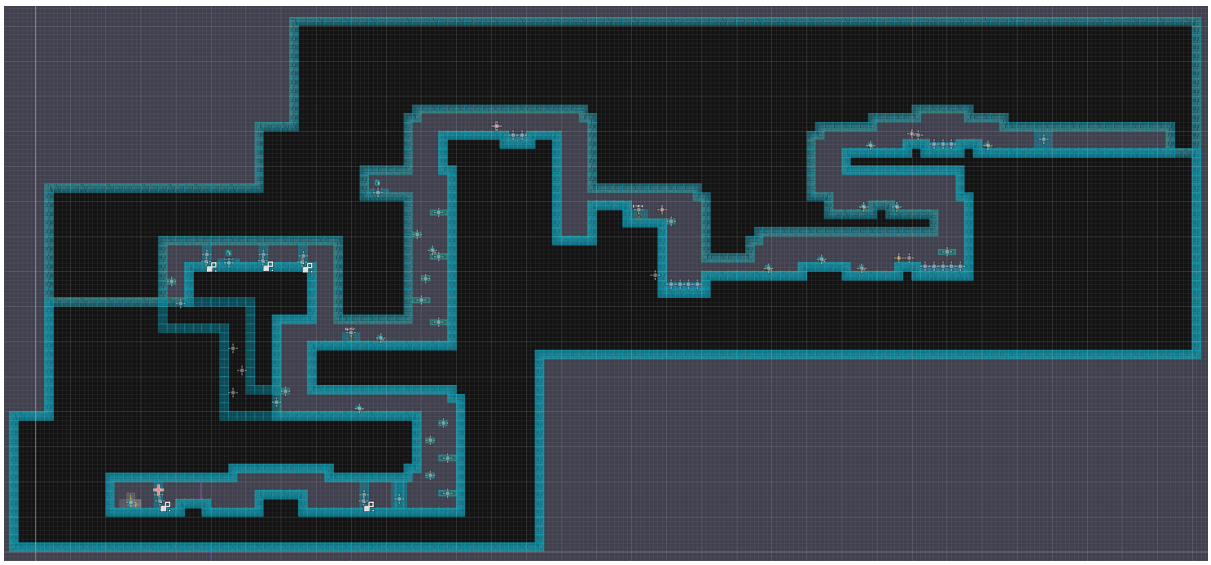

Figura 178: Disposició final Tutorial

7.2.2. Rotting Castle

La disposició d'aquest nivell està pensada per a que el jugador hagi d'explorar i, a mesura que vagi desbloquejant les habilitats, hagi de tornar a explorar zones del mapa abans bloquejades a les quals ara ja pot accedir. És la zona més gran del prototip i intercala zones de combat amb diversos enemics amb zones de puzzles de plataformes i trampes. També conté les entrades i sortides de les sales *d'upgrades* i *trials*. Veure Figura 179.

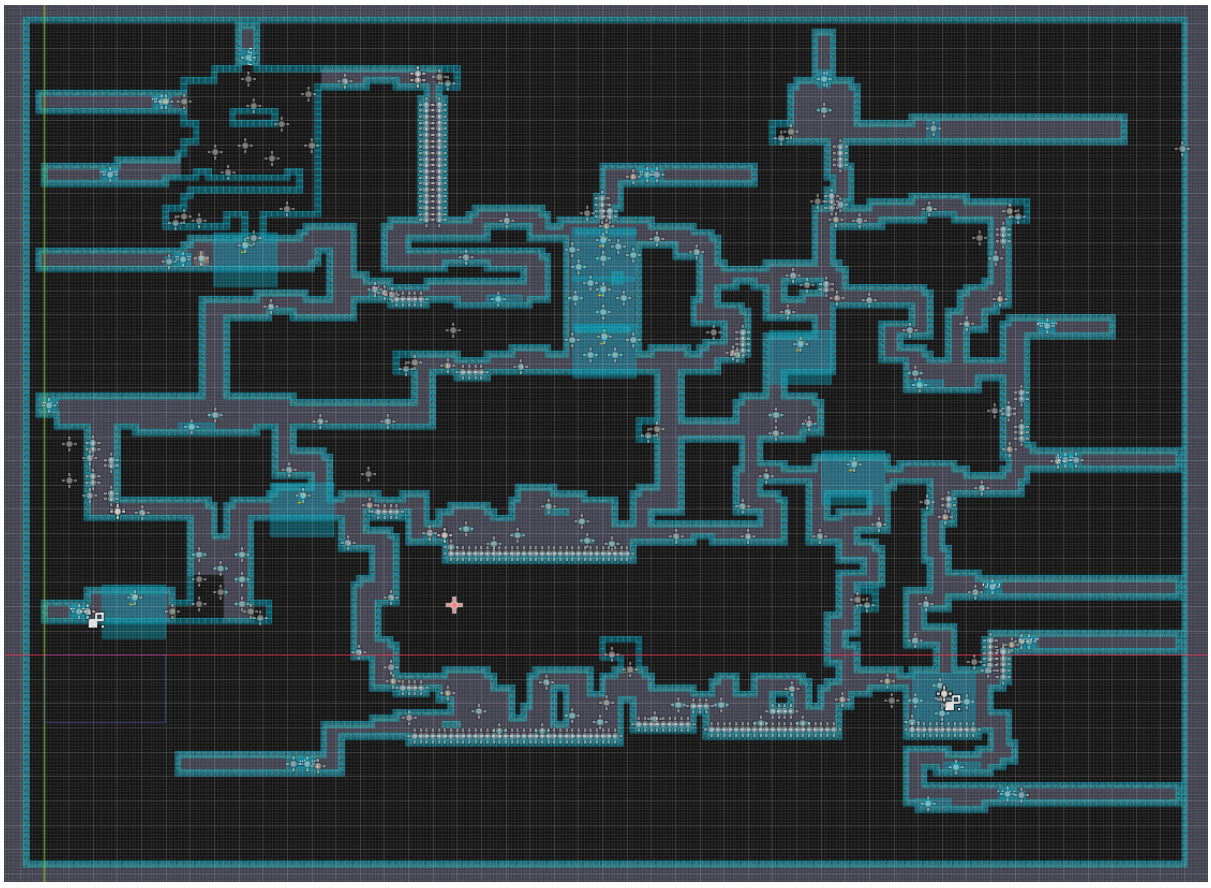

Figura 179: Disposició final de Rotting Castle

A continuació, a la Figura 180, es mostra una comparació entre les escales de la zona de tutorial i la del Rotting Castle, a la mateixa escala.

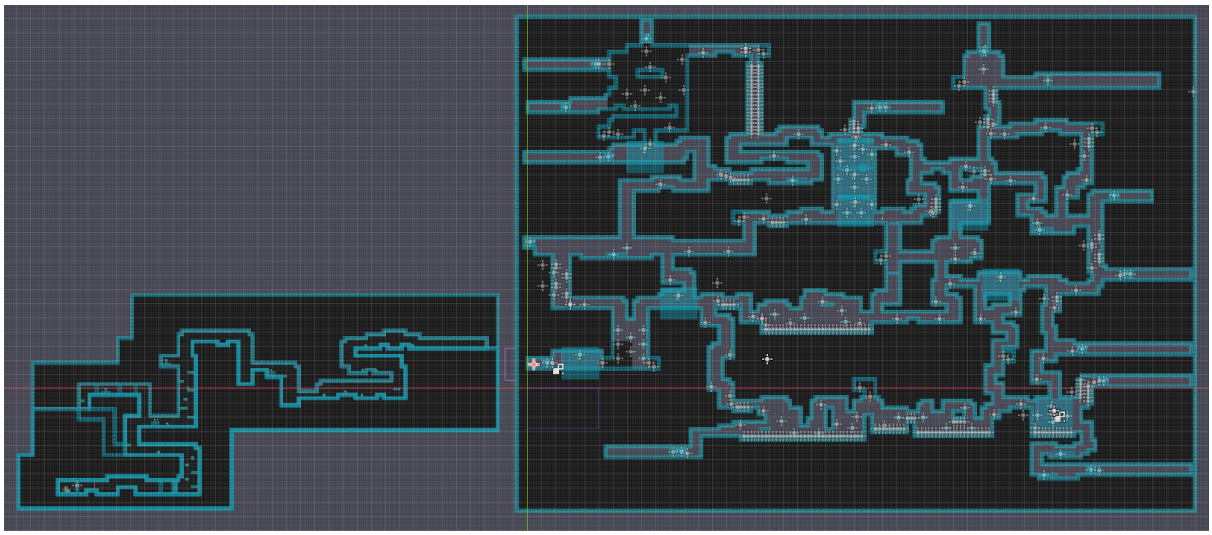

Figura 180: Comparació escales Tutorial i Rotting Castle

7.2.3. Sales upgrades dash i double jump

La disposició d'aquests nivells implica un avanç lineal del jugador. L'objectiu principal d'aquestes sales és que el jugador aprengui el funcionament de la nova mecànica que acaba de desbloquejar, el *dash* o el *double jump* segons la sala, a través d'una sèrie de puzzles que requereixen d'aquesta nova habilitat. Veure Figures 181 i 182.

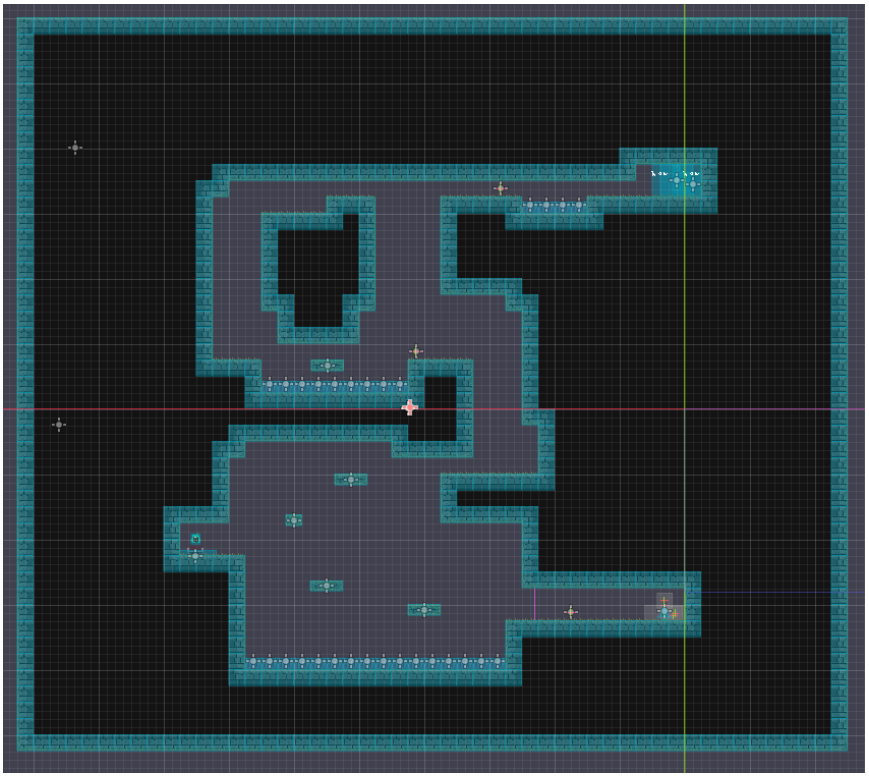

Figura 181: Disposició final sala upgrade dash

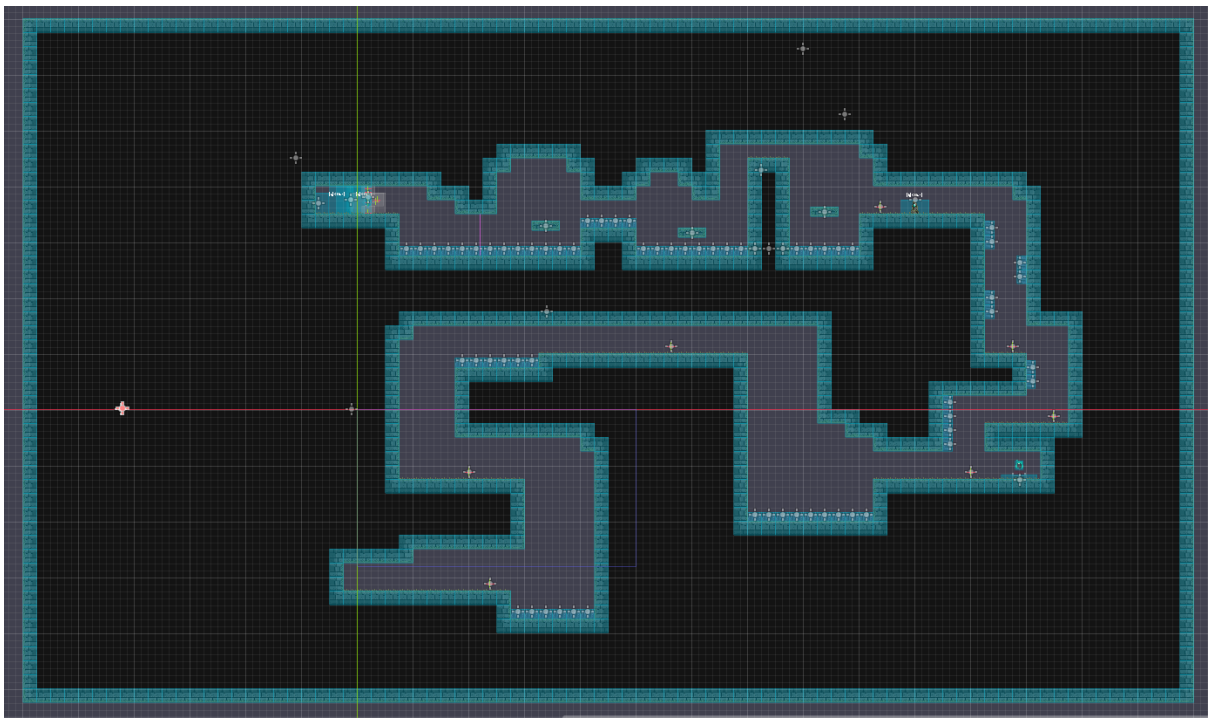

Figura 182: Disposició final sala upgrade double jump

7.2.4. Sales trials shield, fireball i plunge

La disposició d'aquests nivells implica un avanç lineal del jugador. En aquestes sales el jugador pot desbloquejar un nou *spell.* Les trampes d'aquestes sales estan relacionades amb l'*spell* que si desbloquejarà i insinuen al jugador el possible funcionament. Alguna sala requereix que el jugador utilitzi l'*spell* que acaba d'aconseguir per completar-la. Veure Figures 183, 184 i 185.

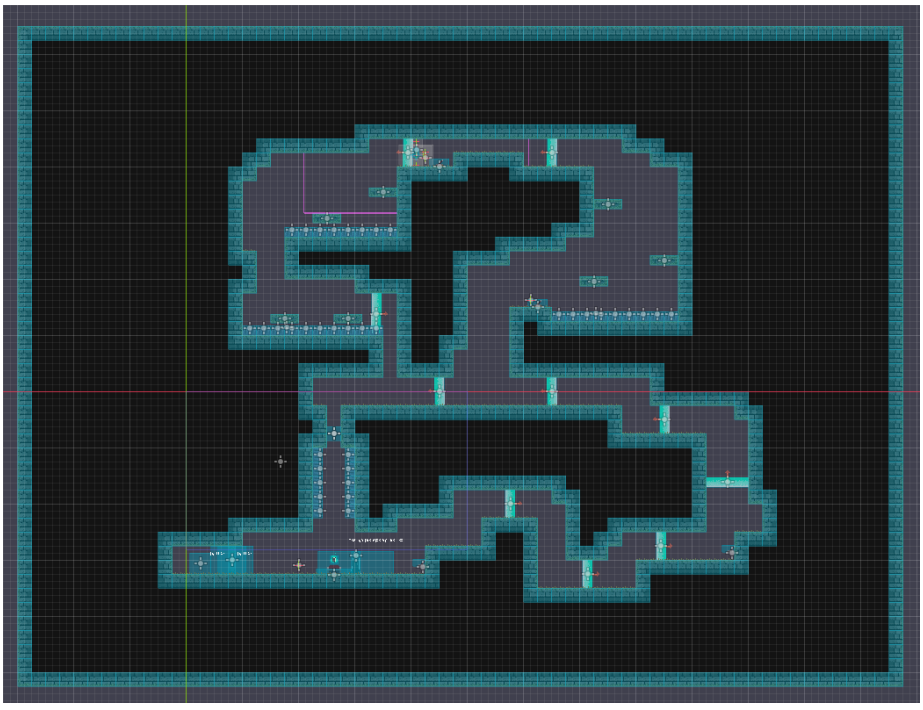

Figura 183: Disposició final sala trial shield

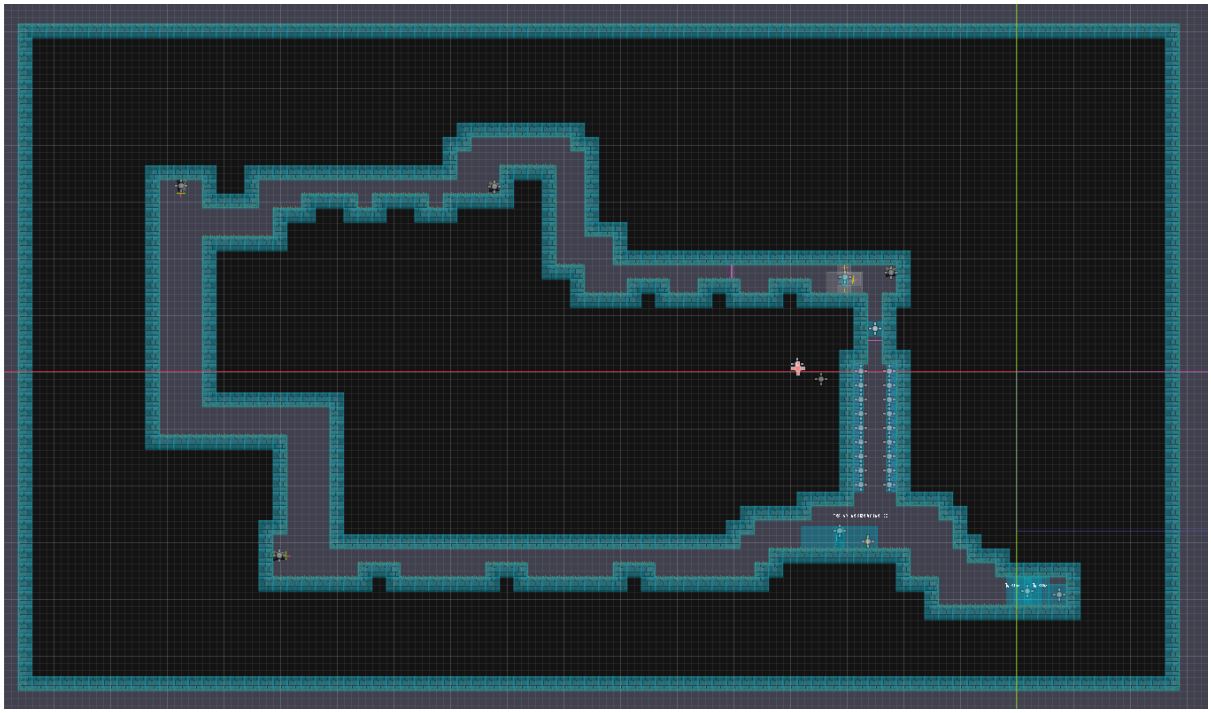

Figura 184: Disposició final sala trial fireball

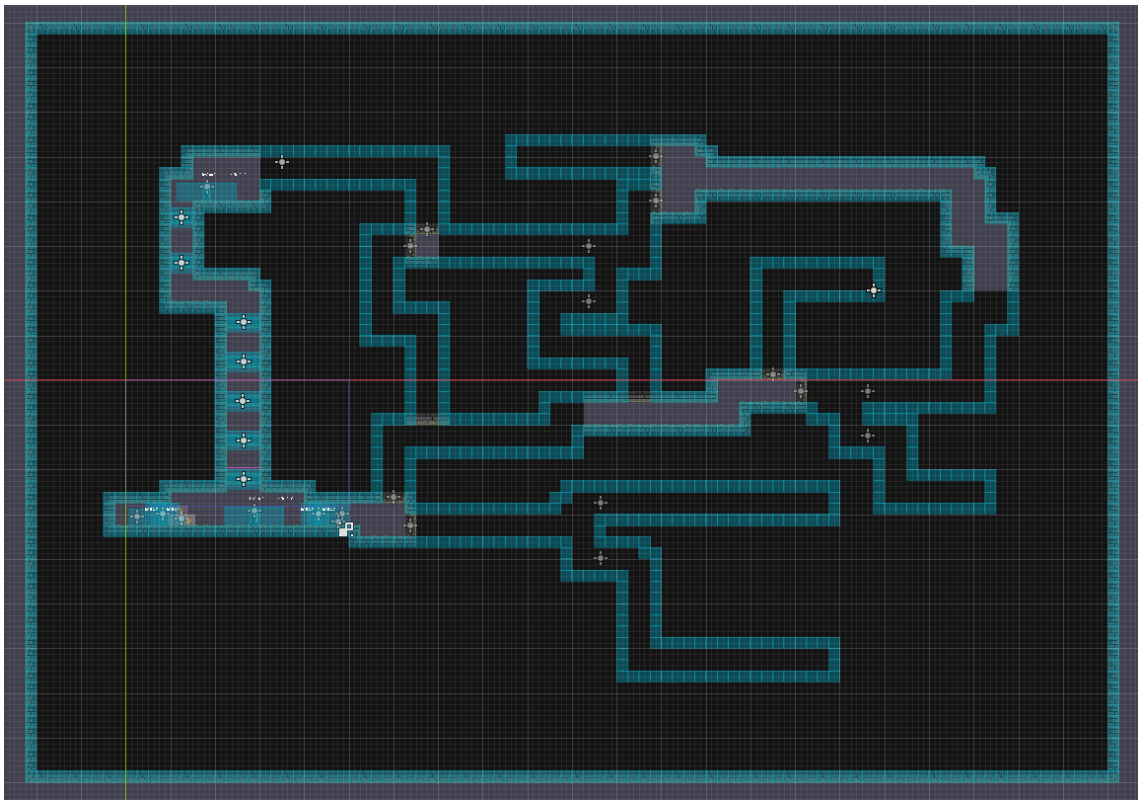

Figura 185: Disposició final sala trial plunge

7.2.5. Sala boss final

La disposició d'aquest nivell és molt simple, ja que l'objectiu és crear una àrea on es pugui dur a terme el combat entre el jugador i el *boss*, sense interrupcions d'altres enemics o trampes. Veure Figura 186.

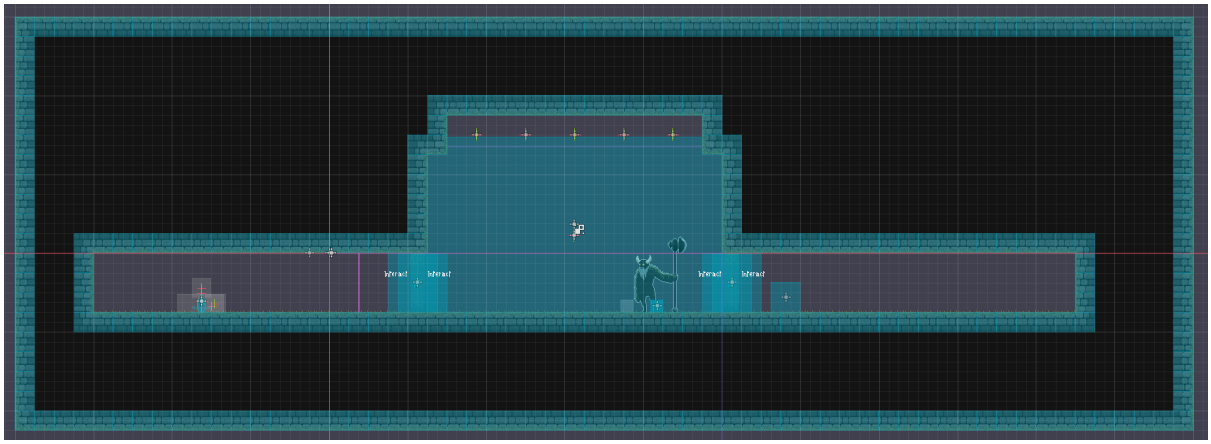

Figura 186: Disposició final sala Boss

7.3. Captures del videojoc final

7.3.1. Interaccions amb personatges

A les figures següents es pot veure al jugador lluitant o interactuant amb cada un dels personatges del videojoc.

Combat amb Roamer

A continuació es mostren com el jugador danya un Roamer i com es l'*sprite* quan es mor. Veure figures 187 i 188.

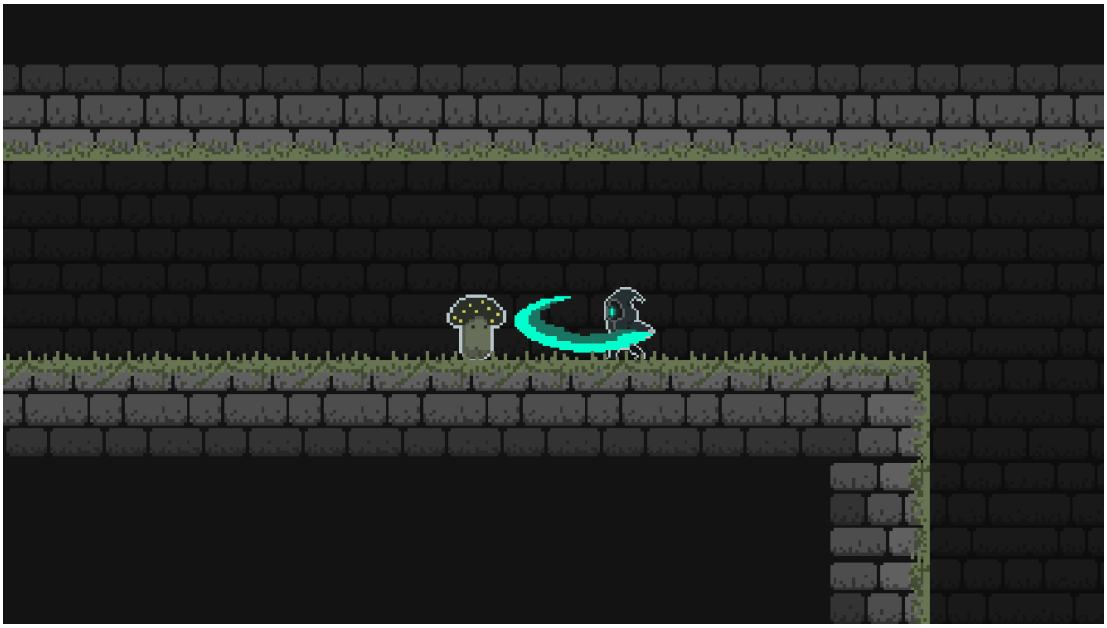

Figura 187: Jugador atacant un Roamer

Figura 188: Roamer mort

Combat amb Bomber

En les següents figures podem observar el jugador esquivant els atacs del bomber i després danyant-lo. Veure figures 189, 190 i 191.

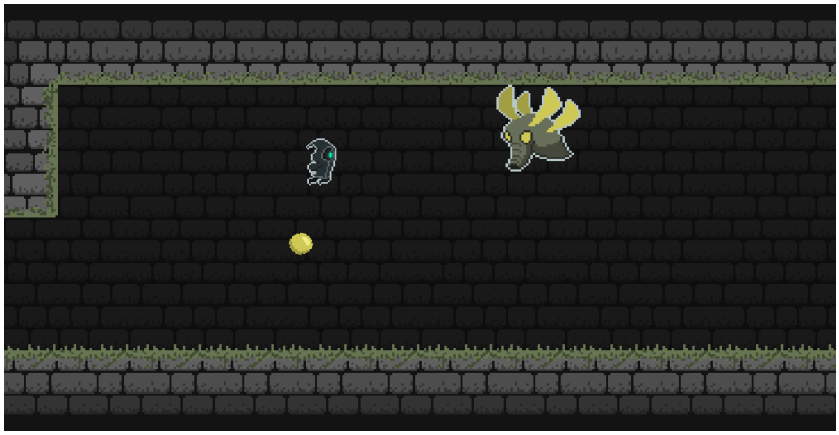

Figura 189: Jugador esquivant atac del Bomber (1)

Figura 190: Jugador esquivant atac del Bomber (2)

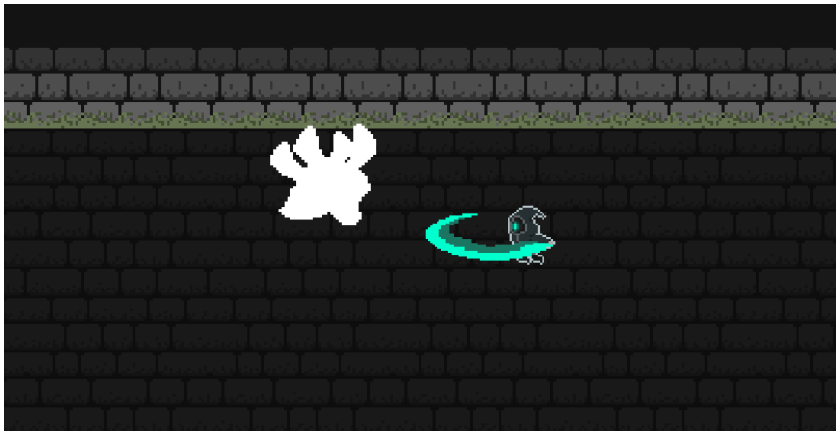

Figura 191: Jugador danyant a Bomber

Combat amb Charger

A continuació veiem diferents interaccions entre el Charger i el jugador. Veure figures 192, 193, 194 i 195.

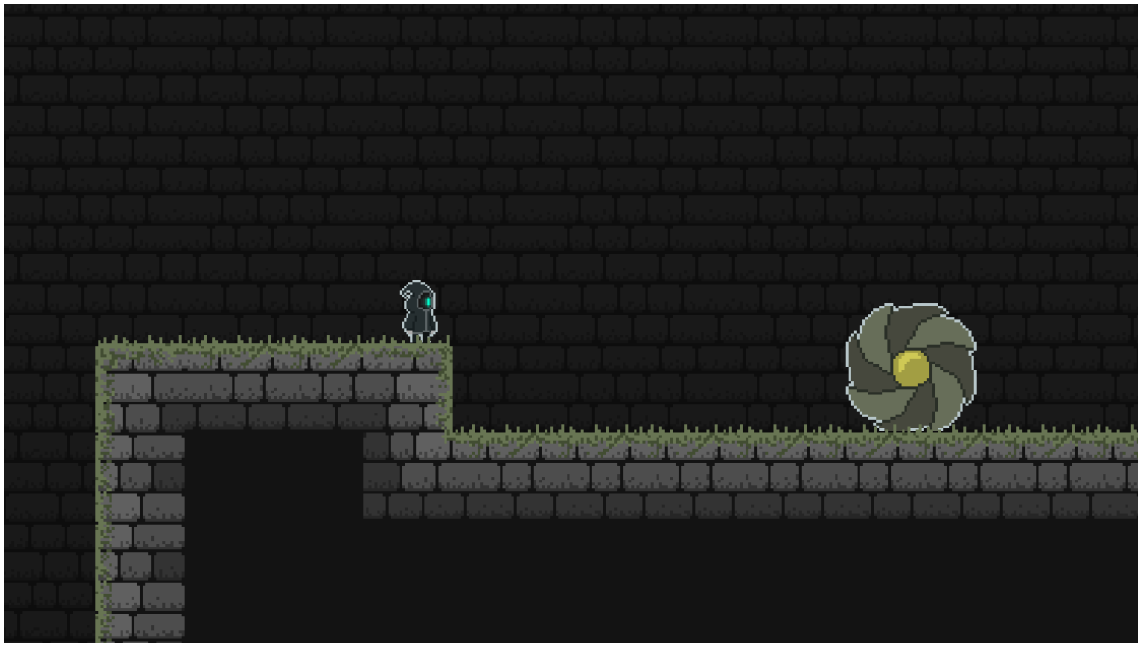

Figura 192: Jugador observant al Charger

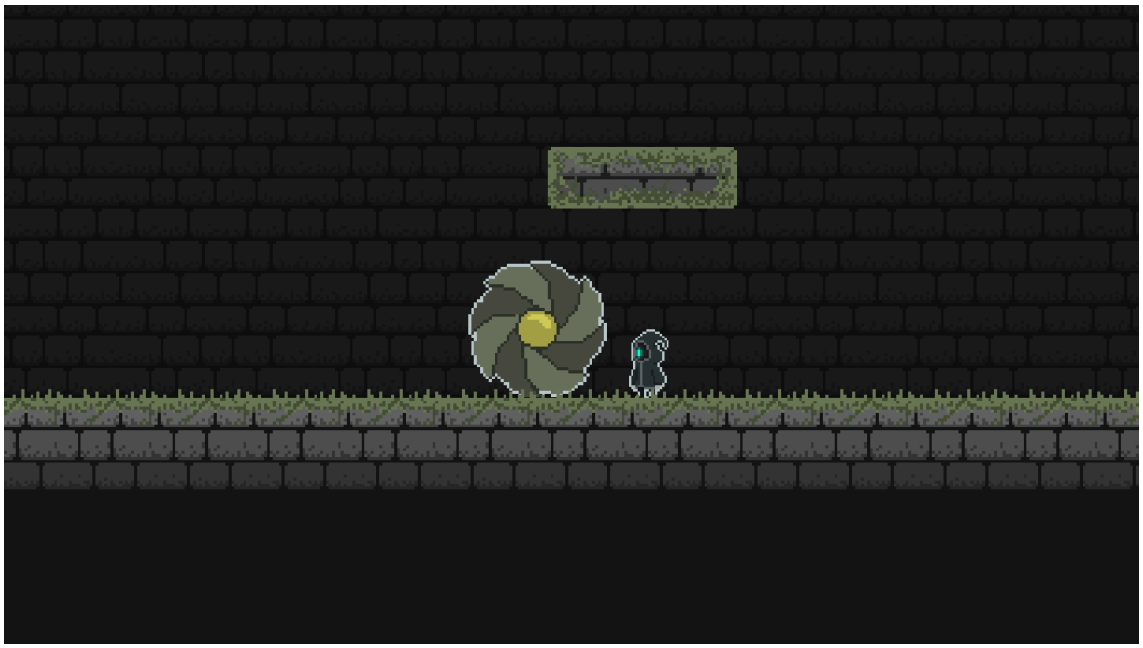

Figura 193: Charger atacant al jugador

Figura 194: Jugador danyant al Charger

Figura 195: Charger mort

Interacció amb NPC

A continuació es mostren les possibles interaccios entre el jugador i el NPC. Veure figures 196, 197 i 198.

Figura 196: Diàleg NPC quan jugador no té els scrolls necessaris

Figura 197: Diàleg NPC quan jugador té els scrolls però no ha completat el trial

Figura 198: Diàleg quan jugador ha completat el trial

Combat amb Boss

En les figures següents podem observar com són algunes de les fases de la lluita entre el jugador i el *boss*. Veure Figures 199 i 200.

Figura 199: Jugador abans del combat amb el Boss

Figura 200: Atac 1 del Boss

7.3.2. Interaccions amb objectes

A les figures següents es pot veure al jugador interactuant amb cada un dels objectes del videojoc.

Interacció amb BossEssence

A la figura següents s'observa com és una *BossEssence* i el missatge que es mostra per pantalla quan el jugador n'obté una. Veure Figura 201.

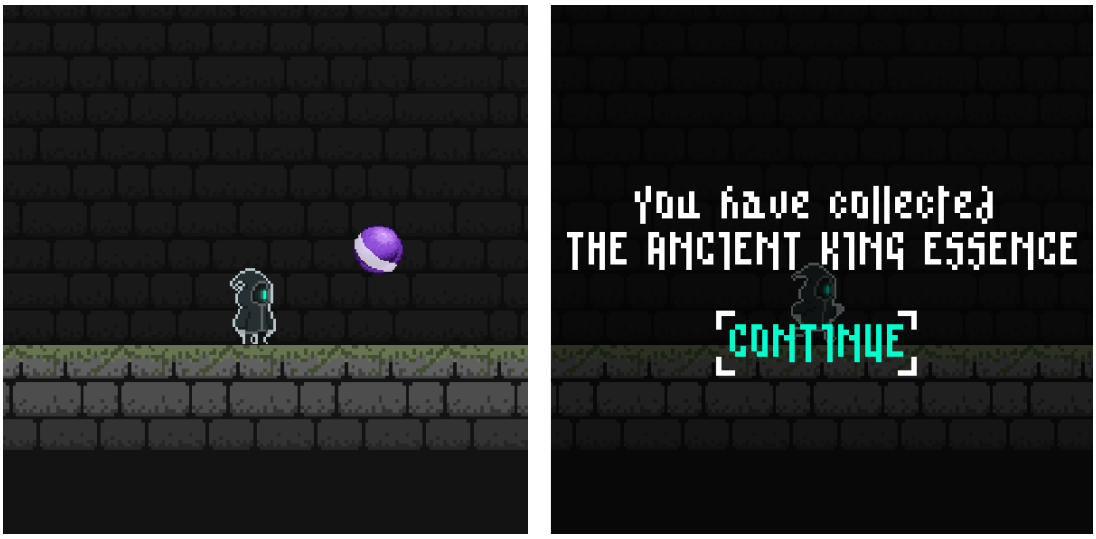

Figura 201: Interacció jugador amb BossEssence

Interacció amb Checkpoint

A continuació es pot veure com el jugador activa un checkpoint. Veure Figura 202.

Figura 202: Interacció jugador amb Checkpoint

Interacció amb cofre i Upgrade

A les figures següents podem observar al jugador obrir un cofre i recollir un upgrade, amb el missatge que es mostra quan ho fa. Veure Figures 203 i 204.

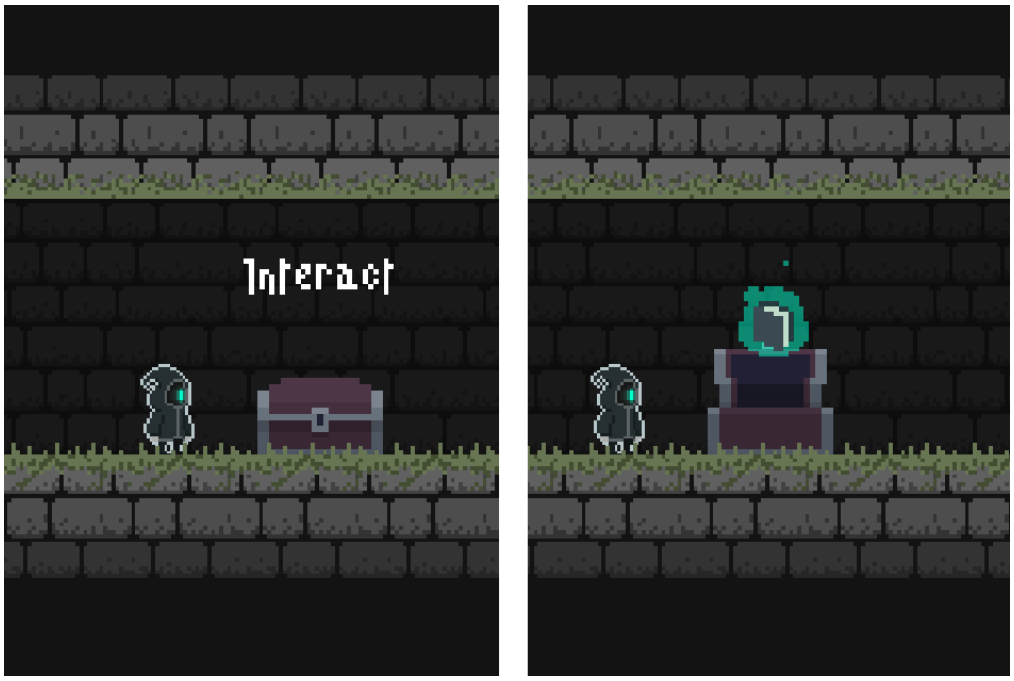

Figura 203: Interacció jugador amb cofre

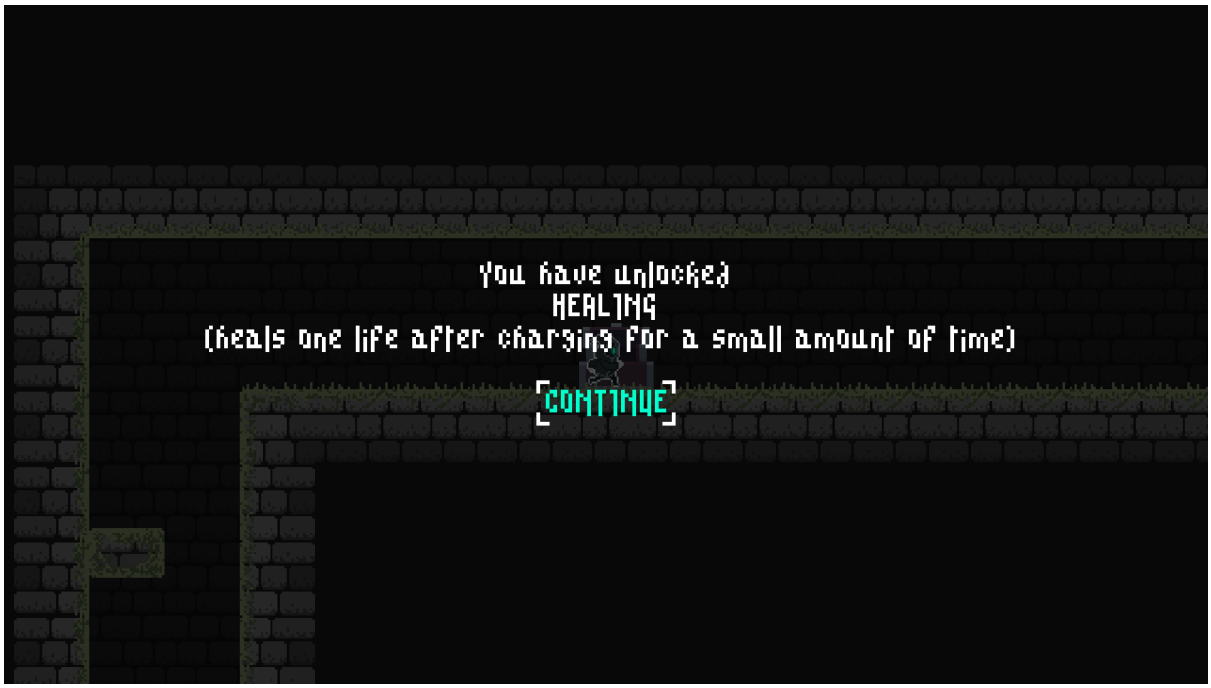

Figura 204: Interacció jugador amb Upgrade

Interacció amb porta

A la figura següent es mostra al jugador obrint una porta i, un cop l'ha travessada aquesta es bloqueja (canvia el color de les runes). Veure Figura 205.

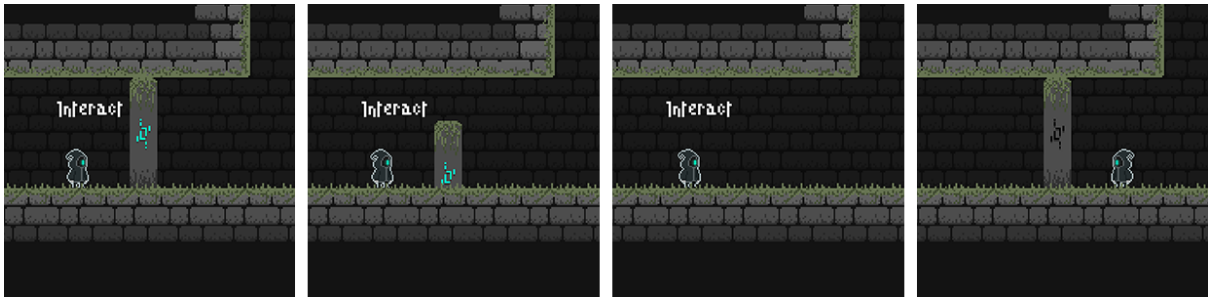

Figura 205: Interacció jugador amb porta

Interacció amb Fireball cannon

A continuació podem veure com el *fireball cannon* dispara un projectil. Veure Figura 206.

Figura 206: Interacció amb Fireball Cannon

Interacció amb palanca

A la figura següent veiem el jugador activant una palanca. Veure Figura 207.

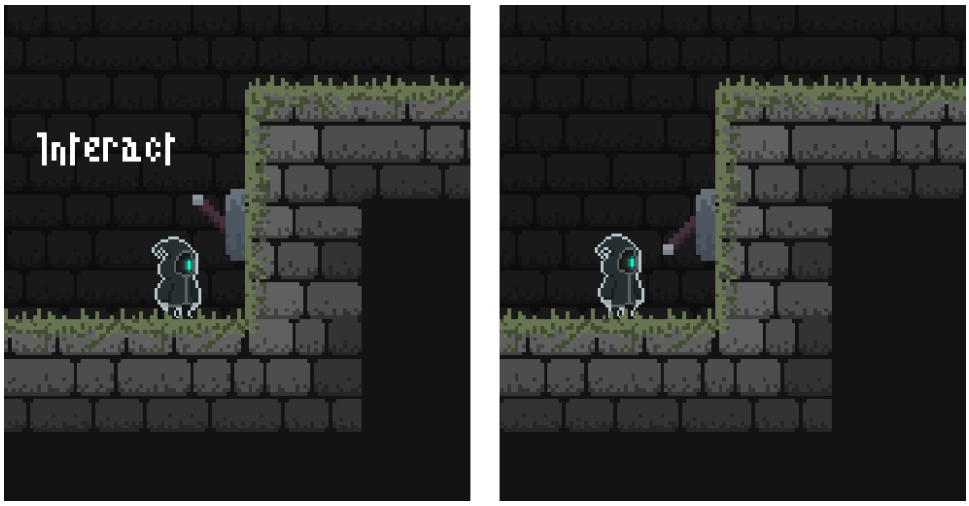

Figura 207: Interacció amb palanca

Interacció amb plataforma

A la figura següent veiem el jugador utilitzant una plataforma per evitar les punxes. Veure Figura 208.

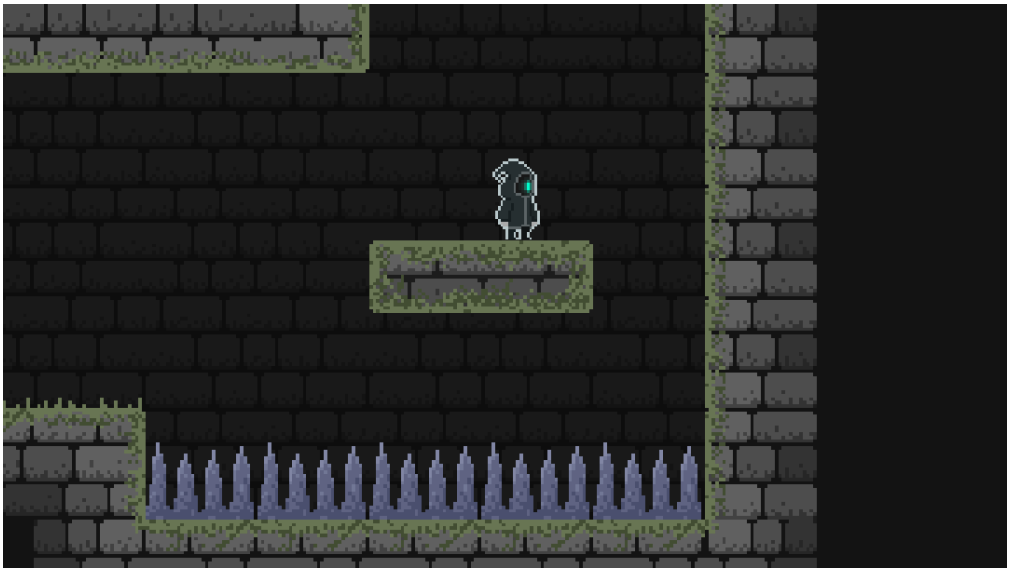

Figura 208: Interacció amb plataforma

Interacció amb Scroll

A continuació podem observar com el jugador detecta una paret amagada i al destruir-la es troba amb un *scroll*, que podrà recollir. Veure Figura 209.

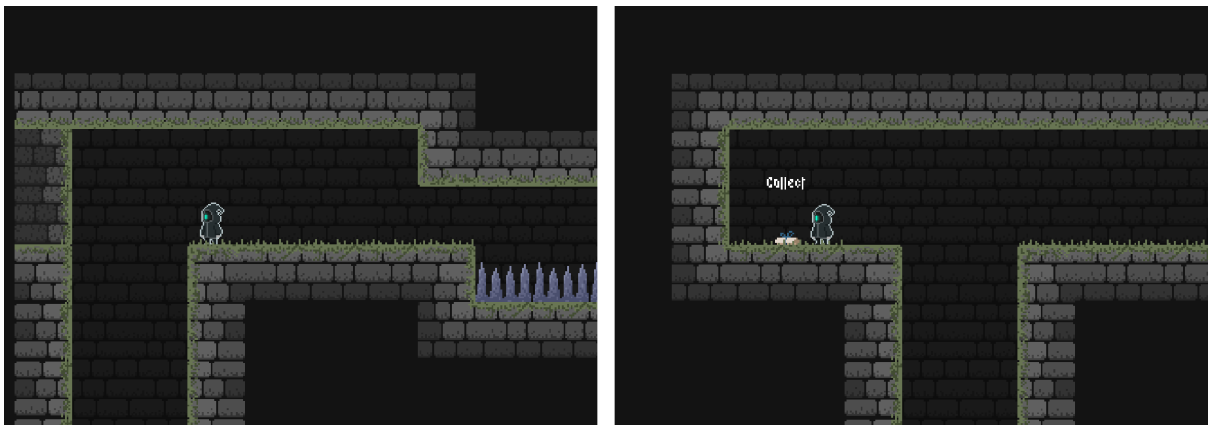

Figura 209: Interacció amb Scroll

Interacció amb Shield obstacle

A les figures següents veiem les interaccions del jugador amb un *shield obstacle*. A la primera figura, el jugador intenta travessar-lo però no pot perquè va en sentit de la col·lisió. A la segona, veiem que si pot travessar-lo perquè va en sentit contrari a la col·lisió. El sentit de la col·lisió es pot saber per la banda blanca que té el *shield obstacle.* Veure Figures 210 i 211.

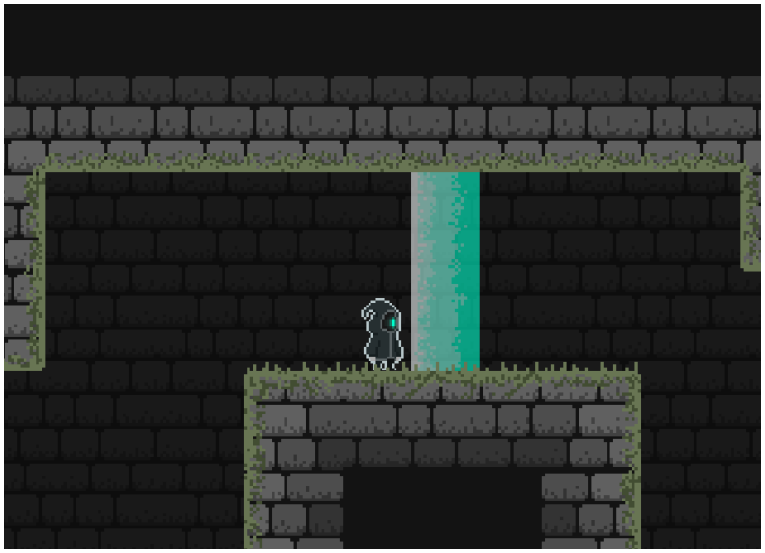

Figura 210: Interacció amb ShieldObstacle (1)

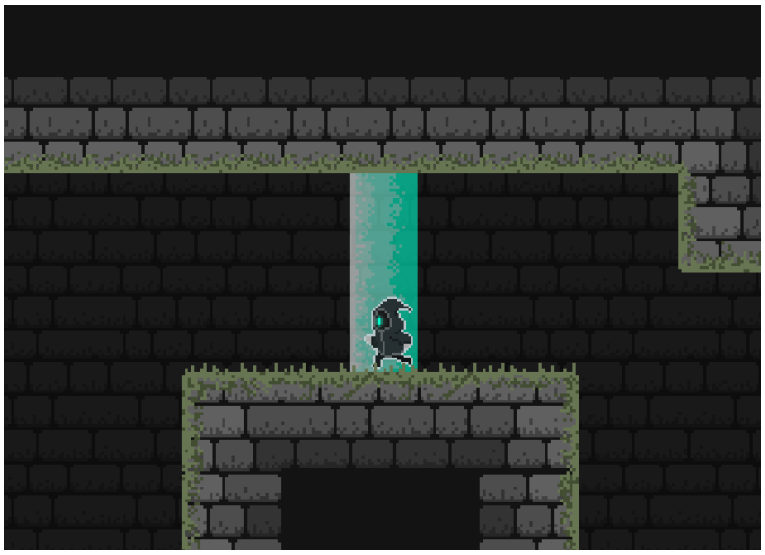

Figura 211: Interacció amb ShieldObstacle (2)

Interacció amb punxes

A continuació podem veure la seqüència del jugador interactuant amb les punxes. Aquest intentar saltar-se però cau sobre elles. Llavors, perd una vida i retorna a la posició d'*spawn* de les punxes. Veure Figura 212.

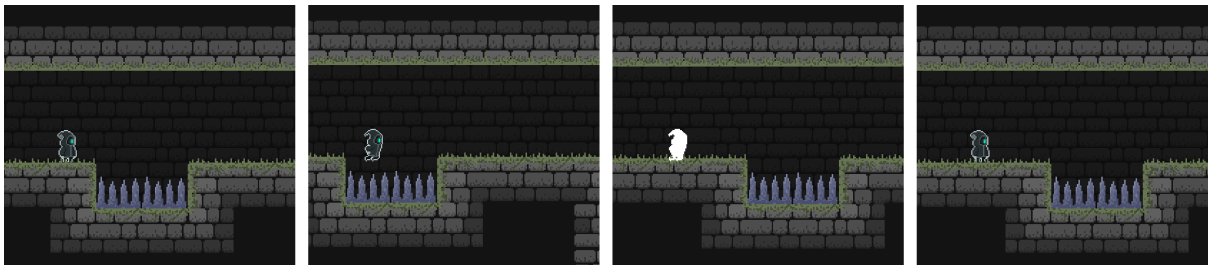

Figura 212: Interacció amb punxes

Interacció amb HiddenWall

A les figures següents es mostra el jugador destruïnt una paret amagada i revelant el camí secret. Veure Figura 213.

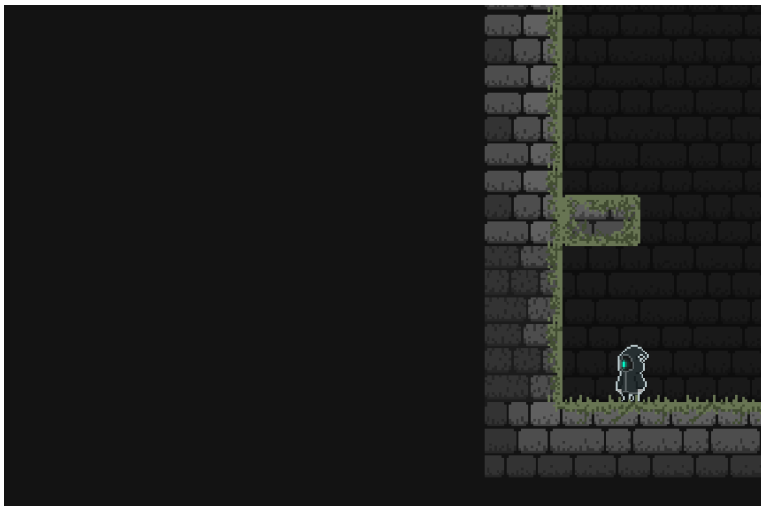

Figura 213: Interacció amb HiddenWall (1)

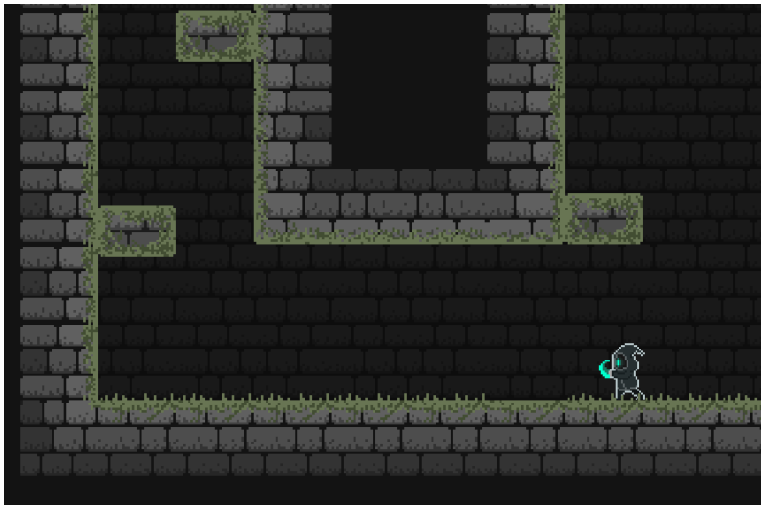

Figura 214: Interacció amb HiddenWall (2)

7.4. Classificació PEGI del videojoc

El sistema PEGI és un sistema de classificació de videojocs per edats que orienta als consumidors per ajudar-los a decidir si comprar un videojoc. El nostre videojoc entraria dins l'etiqueta d'edat de PEGI 12 (veure Figura 215) i tindria el descriptor de contingut de violència (veure Figura 216), ja que conté violència gràfica contra personatges fantàstics i violència no realista contra personatges semblants als humans.

Figura 215: Icona de PEGI 12

Figura 216: Icona de PEGI descriptor de violència

8. Conclusions

Aquest projecte ha estat el primer d'aquesta grandària i complexitat que s'ha realitzat de manera individual. Això ha fet que sigui necessari implementar la major part del videojoc, cosa que ha requerit posar en pràctica molts dels conceptes que s'han anat adquirint durant el grau. Alguns d'aquests conceptes havien estat menys desenvolupats, ja que sempre s'havia treballat en grup per projectes d'aquest abast. Cada membre s'ocupava principalment d'una àrea concreta, tot i que s'estava involucrat en totes però en menys mesura. Personalment, sempre he treballat més les parts de l'estètica i el disseny i ha estat interessant haver pogut dissenyar i implementar la part de la programació. Això també m'ha permès obtenir una visió més profunda i completa de l'àmbit del desenvolupament de videojocs.

En quant a la planificació, veient les dimensions del projecte es va decidir començar a l'inici del curs acadèmic per tenir temps suficient per poder desenvolupar-lo tranquil·lament i dedicar el temps adequat a cada una de les parts. Durant el projecte s'han anat modificant alguns elements del disseny inicial per adaptar-los més a les característiques del videojoc, ja que els previstos inicialment no s'acabaven d'ajustar a com estava evolucionant el videojoc. També s'han modificat les temporitzacions d'algunes tasques, principalment allargant la seva duració, degut a problemes que han anat sorgint o per millores que s'han volgut realitzar.

En quant a l'assoliment dels objectius del projecte, tot i ser un prototip del que seria una zona del videojoc final, s'ha aconseguit mostrar com seria el disseny, l'ambientació i la jugabilitat d'aquest un cop estés acabat. S'han pogut completar totes les tasques excepte la implementació del combat amb el *boss* final, que per la complexitat de programació, la necessitat d'una gran quantitat d'animacions i la limitació de temps només se n'ha pogut implementar una part. Això no ha estat un obstacle per assolir l'objectiu general que es va proposar a l'inici del projecte, que era el de desenvolupar un videojoc del gènere metroidvania amb mecàniques de plataformes i combat polides i un món interconnectat no lineal ambientat en una època medieval de fantasia.

9. Treball futur

En aquest apartat es comenten les tasques i elements que es voldrien afegir en un futur.

- **Implementació del *Boss1*:** Acabar la implementació del *Boss* de la primera zona perquè el combat sigui més interessant i desafiant.
- **Música i Efectes de so:** Crear la música que sonarà de fons, a cada una de les zones i segons la situació sonarà una música diferent; i els efectes de so, sons d'ambient, dels personatges, dels objectes, etc.
- **Disseny i implementació de la resta de zones:** Implica un seguit de subtasques d'entre les quals destaquen:
    - Disseny de noves mecàniques del jugador, enemics, trampes i *Bosses* finals
    - Afegir nous NPC. Això implica haver de crear un sistema de diàleg i modificar l'economia del videojoc.
- **Modificacions a l'economia:** S'afegiran nous traders, per transferir les "monedes" del jugador al NPC de les botigues; i nous converters, intercanviar les "monedes" per objectes, mapes, *spells, upgrades,* etc.
- **Sistema de diàleg:** A l'afegir nous NPC i la possibilitat de comprar objectes, caldrà tenir un sistema de diàleg que simplifiqui la tasca de crear i modificar converses.
- **Sistema de guardat:** Permetre al jugador guardar una partida diferent a cada un dels perfils i eliminar partides ja començades per començar de nou.
- **Millores d'animacions:** Millorar les animacions actuals i afegir-ne de noves per incrementar la qualitat de l'estètica.
- **Millores de codi:** Millorar i optimizar el codi per incrementar el rendiment del videojoc i evitar possibles errors.
- **Decoracions:** Afegir decoracions als nivells perquè se sentin més reals i amb més vida.
- **Testing més extens:** Fer un testeig del videojoc amb més usuaris i de manera més exhaustiva.
- **Comercialització:** Un cop acabat el videojoc, publicar-lo a Steam i publicitar-lo a les xarxes socials.

10. Bibliografia

Aquests són els recursos més rellevants que s'han anat consultant durant el projecte, la major part en diverses ocasions.

Godot Engine. (2022). *Godot Docs*. https://docs.godotengine.org/en/stable/

Aseprite. (2022). *Aseprite Docs.* https://www.aseprite.org/docs/

GDQuest. g*odot-demos/2018/finite-state-machine.* (4 setembre 2021). GitHub. https://github.com/GDQuest/godot-demos/tree/master/2018/04-24-finite-state-machine

jotson. *camera_shake_tutorial* (2 maig 2021). GitHub. https://github.com/jotson/camera_shake_tutorial

Yukitty. *godot-addon-integer_resolution_handler.* (28 octubre 2020). GitHub. https://github.com/Yukitty/godot-addon-integer_resolution_handler

Bright, G. *Build a Bad Guy Workshop - Designing enemies for retro games.* (22 abril 2014). https://www.gamedeveloper.com/design/build-a-bad-guy-workshop---designing-enemies-for-retro-games

studiominiboss. (2019). *Pixel art tutorials.* https://blog.studiominiboss.com/pixelart

KidsCanCode. (2021). *Godot recipes.* https://kidscancode.org/godot_recipes/

HeartBeast. (13 abril 2018). *Godot 3 2D Platform Game.* [Playlist]. YouTube. https://www.youtube.com/playlist?list=PL9FzW-m48fn2jlBu_0DRh7PvAt-GULEmd

Artindi. (31 octubre 2019). *Coyote Time Jumping - Now You Know Too - Godot Tutorial.* [Vídeo]. YouTube. https://www.youtube.com/watch?v=LeaaKRufdMc&list=PLJ69M9BmiffJvGYV0nNBY0GX_Z6rR-jDS&index=3&ab_channel=Artindi

HeartBeast (3 abril 2020) *Make an Action RPG in Godot 3.2 (P11 | Melee attacks with Hurtboxes and Hitboxes).* [Vídeo]. YouTube. https://www.youtube.com/watch?v=vDbEfmPcv-Q&list=PLJ69M9BmiffJvGYV0nNBY0GX_Z6rR-jDS&index=8&ab_channel=HeartBeast

Per a cerques de referències estètiques s'han utilitzat les dues webs següents:

Pinterest. (2022). https://www.pinterest.es/

Google Images. (2022). https://images.google.com/

11. Manual d’usuari i d’instal·lació

11.1. Manual d’instal·lació

El procediment següent és el que cal seguir per jugar per primera vegada. El videojoc només funciona en ordinadors amb sistema operatiu Windows.

1. Descomprimir l’arxiu ZIP
2. Entrar dins el directori extret
3. Executar *Awaken.exe*

11.2. Manual d’usuari

Un cop executat el videojoc, es carregarà el menú inicial.

Per jugar, cal seleccionar l’opció *Start Game* i cal seleccionar el primer perfil.

Si es volen editar les opcions de joc o consultar els controls, cal entrar al submenú *Options.*

Si es vol sortir del joc, cal obrir el menú del jugador, navegar fins el submenú *Settings* i escollir l’opció *Quit Game.*

11.3. Controls

Els controls del videojoc són els que es mostren a continuació a la Taula 8. També es poden consultar en qualsevol moment durant la partida accedint al menú del jugador i al submenús de *settings* i després *controls*. El videojoc es pot jugar amb comandament i amb teclat, tot i que es recomana amb comandament.

Acció	**Comandament (de Xbox)**	**Teclat**
Moure	Joystick esquerre	A / D
Saltar	Botó A	Space
Dash	Botó superior dret	F
Atacar	Botó X	K
Interactuar	Botó B	E
Pausar el joc	Botó menú	P
Llançar *spell down*	Gallet esquerre	J
Llançar *spell up*	Gallet dret	L
Canviar *spell down*	Creueta a baix	Fletxa a baix
Canviar *spell up*	Creueta a dalt	Fletxa a dalt
Pàgina següent (dins menú jugador)	Botó superior dret	E
Pàgina anterior (dins menú jugador)	Botó superior esquerre	Q

Taula 8: Controls