Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Projecte "Pawn Sacrifice": Joc roguelite on en cada nivell es perd un company

Document: Memòria

Alumne: Vicenç Verge Mondejar

Tutor: Gustavo Patow
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any) Juny / 2021

ÍNDEX

1. Introducció

Hi ha un gènere de videojocs el qual no és molt conegut i s'anomena *Rogue-Like*. Consisteix en superar diversos nivells, generalment ambientats en masmorres, fins a arribar al final mentre es derroten enemics, amb la particularitat de que els nivells són generats aleatòriament cada vegada que s'inicia una nova partida. En aquests videojocs el jugador millora el personatge durant el transcurs d'una partida, a través d'objectes o habilitats aconseguides a partir de diferents processos. Tot i això, tenint en compte que al començar de nou, aquest progrés també és esborrat.

Un videojoc que exemplifica aquest gènere de videojocs és el que es veu en la Figura 1.1, *The Binding of Isaac*, en el qual la enorme quantitat d'objectes i enemics que té fan que cada partida sigui diferent a la anterior.

Figura 1.1: Captura de The Binding of Isaac: Repentance

La meva intenció en aquest projecte és crear un joc del gènere dels *Rogue-Like*, en el qual el jugador vagi acompanyat i es faci més fort a partir de sacrificar un d'aquests aliats i obtenir una habilitat de cada un d'ells en cada nivell.

1.1 Motivacions

La idea per a fer aquest videojoc va començar com a un concepte per a fer un joc durant pocs dies i moure'm al següent. Però, realment tenia potencial, vaig decidir convertir-ho en el meu TFG. Això va venir a causa de la motivació que tenia de fer un algorisme de generació

procedural, els nivells, i de crear quelcom que no he vist en el mercat, com pot ser un *Rogue-Like* en el qual el jugador no va sol durant l'aventura.
A més, el fet que és el primer prototip de videojoc que creo en solitari, pel que el fet que el videojoc sigui bo o no recau absolutament en mi.

1.2 Propòsit

Tot i que el gènere del que forma part el videojoc no és molt conegut, hi ha molts jocs al mercat. Tot i això, no conec a cap en el qual el personatge controlat pel jugador vagi acompanyat per PNJ (Personatges No Jugables) i que, a més, tinguin una funció vital en la jugabilitat, a causa que en cada nivell es canvia un PNJ per una de les seves habilitats, la qual es sumarà al personatge principal. Aquesta és una mecànica que tampoc és molt comuna, ja que això farà que, depenent de les decisions preses pel jugador, la dificultat de la partida serà més fàcil o difícil.

1.3 Objectius

L'objectiu d'aquest TFG és desenvolupar el prototip d'un videojoc en el que seguiré treballant a posteriori, del gènere dels *Rogue-Like*, en el qual el jugador va acompanyat d'altres PNJs, amb la particularitat que, cada vegada que es passa de nivell, s'ha de prescindir d'un company, obtenint a canvi alguna de les seves habilitats. Això s'ha de fer mentre es lluita en equip contra els diferents enemics que hi hauran repartits en el mapa, pel que l'elecció del company a prescindir serà un factor estratègic important de cares al joc. A més, la generació procedural de la masmorra també és un gran aspecte a tenir en compte i el qual ha de ser lo suficientment bo com per fer que cada partida sigui diferent a l'anterior.

Aquest treball es centra en l'apartat de la programació i en el disseny més que en l'estètic. Això és a causa que les arts no són el meu fort i no puc proporcionar un gran estil artístic a un videojoc.
El joc es desenvoluparà en C# a partir del motor de videojocs Unity3D, i l'apartat gràfic amb Aseprite. Les tasques a desenvolupar en aquest projecte inclouen, tot i que no es limiten a:

- Disseny del videojoc.
- Estudiar i implementar un algorisme de generació procedural per a la masmorra.
- Implementar les mecàniques principals del jugador (caminar, atacar, ...).
- Estudiar i implementar IAs pels aliats i pels enemics.
- Desenvolupament de la mecànica de prescindir d'un aliat i així obtenir una de les seves habilitats, per fer-les servir a partir d'aquell moment.
- Arrodoniment de la documentació del joc.

1.4 Quadre d'Autoavaluació

Al ser jo l'únic participant en aquest projecte i al no tenir grans capacitats artístiques, he utilitzat un seguit d'assets de tercers per a l'apartat gràfic del videojoc. A aquests els he afegit alguna imatge i animació de collita pròpia. Això es veurà reflectit en la Taula 1. Una altre cosa a dir és que el videojoc gairebé no té cap component narratiu en aquest prototip, pel que també es veurà reflectit en la Taula 1, tot i que la masmorra, els personatges i els enemics seleccionats ja tenen una certa ambientació que fan veure al jugador que es troba un món fantàstic situat en l'edat mitjana.

En canvi, la major part de l'esforç anirà destinat a les mecàniques, ja que és el que marcarà l'experiència del jugador amb l'entorn (mapa, enemics, aliats) i els controls. És amb el que em centraré per a aquest TFG. La segona part més important és la tecnologia, ja que tot el joc estarà desenvolupat en Unity3D, i el coneixement sobre aquest motor em permetrà implementar les mecàniques dissenyades a dintre del joc.

Estètica	5%
Narrativa	5 %
Mecàniques	45 %
Tecnologia	45 %

Taula 1.1: Quadre d'autoavaluació

2. Estudi de viabilitat

En aquest apartat es comenten els recursos i les eines utilitzades per al desenvolupament del projecte. A més, es calcula el possible cost per a la realització del producte final el qual ha de tenir en compte recursos humans, software i altres costs.

2.1 Recursos Tècnics

El prototip del videojoc s'ha desenvolupat en la seva totalitat en un sol ordinador d'ús personal, pel que es pot assumir que el joc complet també es pot dur a terme en el mateix ordinador. Per al desenvolupament del projecte s'han utilitzat els següents recursos:

- **Ordinador AMD Ryzen 7 amb 24 GB de RAM i una gràfica NVIDIA RTX 2070:** Ordinador personal utilitzat per a desenvolupar el projecte.
- **Unity3D v2020.2.5f1:** Motor de videojocs utilitzat.
- **Aseprite:** Software de dibuix en píxels, utilitzat per a modificar l'art de tercers o crear-ne.
- **Visual Studio Code 2020:** Editor de codi utilitzat per a treballar en Unity3D.
- **Word Drive:** Editor de text usat per a escriure la documentació del treball.

A excepció de l'ordinador i Aseprite, els recursos utilitzats són gratuïts, i els que tenen cost econòmic van ser adquirits amb anterioritat a l'inici del projecte. Per tant, no hi ha hagut cap inversió econòmica en l'àmbit tecnològic.

2.2 Recursos Humans

En tot desenvolupament de videojocs hi ha varies persones les quals es reparteixen les funcions a tenir en compte per la creació del videojoc. En aquest cas tota la feina la he fet jo, o la he adquirit de tercers a través d'internet. Ara bé, a l'hora de fer el joc complet, necessitaré els següents perfils de treballadors:

- **Dissenyador/Programador del joc:** És l'encarregat de definir els diferents aspectes del joc com les mecàniques, els enemics, les habilitats, … I de implementar-ho en el joc, a part de programar tot l'aspecte tècnic del projecte.
- **Artista:** És la persona que dissenyaria i dibuixaria tot lo que hi ha en el joc, a més, de fer les animacions i els efectes de partícules.
- **Dissenyador de sons:** Encarregat de dissenyar, crear i implementar la música i els efectes de so dintre del videojoc.

El primer lloc de treball seria el meu, ja que se'm donen millors els aspectes tècnics i la programació en un videojoc. Els altres dos hauran de ser persones contractades.

2.3 Viabilitat Econòmica

En l'àmbit econòmic el prototip ha requerit zero gastos a causa que els recursos necessaris mencionats abans, ja siguin tecnològics o humans, ja es tenien a mà o són gratuïts. Ara bé, a la Taula 2.1 es pot veure el cost total del prototip si en un principi no hagués tingut accés a cap dels recursos.

Recurs	Cost
Ordinador	1500 €
Aseprite	20 €
Word Drive, Visual Studio, Unity3D	0 €
Salari propi	0 €
Suma Total	1520 €

Taula 2.1: Quadre de Costs del Prototip

En la producció del joc complet es donaria lloc a més gastos, a causa que s'han de pagar als demés treballadors o pagar per a assets de tercers. Aquests costs no es poden calcular amb exactitud a causa que, de moment, no s'ha plantejat la extensió que tindria el joc complet i, per tant, no es pot saber les hores que si haurien de dedicar ni els assets que s'haurien de comprar.
Tot i això es pot parlar de salaris que serien donats als treballadors:

- **Dissenyador/Programador del joc:** 15 €/hora
- **Artista:** 13 €/hora
- **Dissenyador de sons:** 13 €/hora

En el moment de planificar el joc sencer es podria estimar el cost econòmic a partir d'aquestes aproximacions.

2.4 Estudi de Mercat

Al pensar la idea i el disseny d'aquest projecte ja tenia clar per on anar, això va ser degut a que ja coneixia d'avantmà el mercat dels videojocs en el gènere *Rogue-Like*. Gràcies a aquests coneixements he pogut pensar una manera de destacar en aquest reduït mercat.

2.4.1 Estat de l'Art

Els jocs que mostraré a continuació m'han servit d'inspiració per alguns aspectes del disseny, tots ells formen part del gènere dels *Rogue-Like*. De cada un en faré una petita descripció.

The Binding of Isaac

The binding of Isaac i els seus conseqüents DLCs és un videojoc amb una estètica, que tot i tractar temes molts obscurs, intenta ser amistosa. Utilitza una vista axonomètrica com es pot veure a la Figura 2.4.1.1 amb una generació de sales aleatòria, en la qual poden haver-hi sales grans i petites. El personatge millora a través de recolectar objectes que poden variar les estadístiques base del personatge o canviar el comportament d'atac entre altres coses.

Figura 2.4.1.1: Imatge de The Binding of Isaac Repentance

UnderMine

UnderMine és un joc que, com es veu a la Figura 2.4.1.2, utilitza un Pixel Art detallat amb una estètica amistosa. Aquest té un component narratiu que, tot i ser irrelevant per a la jugabilitat, hi és. També usa la vista axonomètrica amb una generació de sales similar a The Binding of Isaac. En aquest cas, el jugador també millora el personatge a partir d'objectes, tot i que aquestes millores en són moltes menys en relació amb el joc anterior.

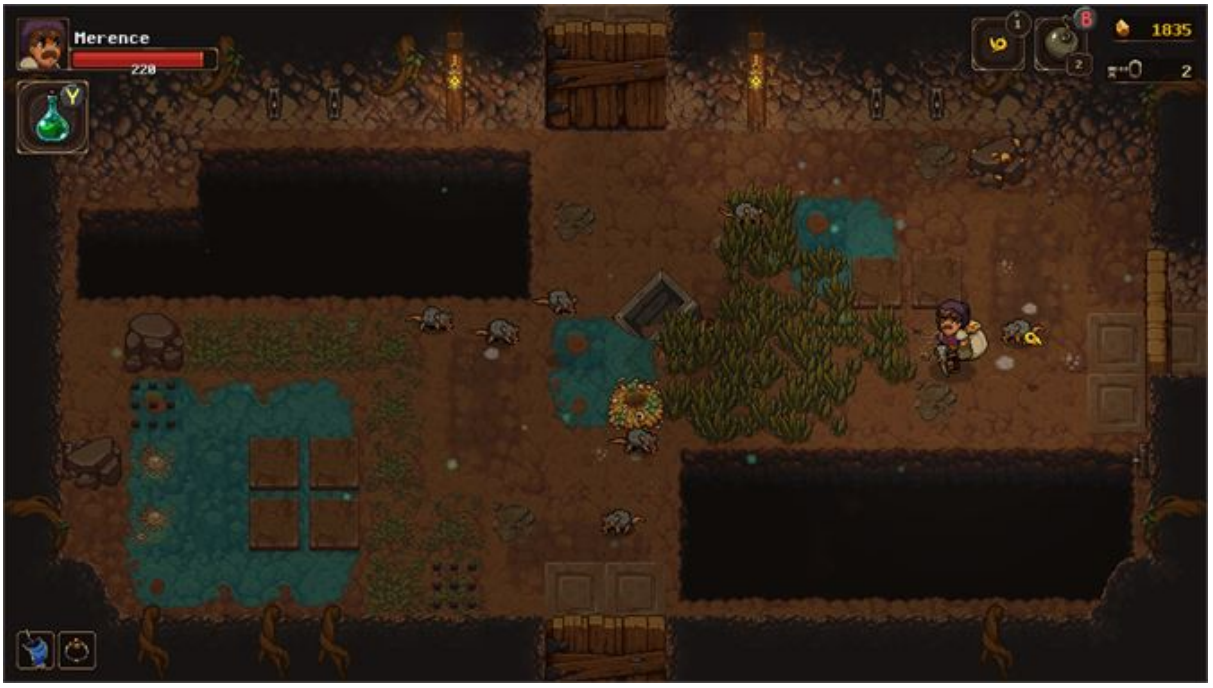

Figura 2.4.1.2: Imatge de UnderMine

Rogue Legacy 2

Rogue Legacy 2 és un videojoc amb una estètica senzilla però eficaç i amistosa, utilitzant, com es pot veure a la Figura 2.4.1.3, una vista frontal, pel que en aquest joc s'ha de tenir en compte la verticalitat de l'espai de joc. En aquest videojoc el jugador depèn molt menys dels objectes que va obtenint durant la partida, ja se n'obtenen pocs, i depèn més de la seva habilitat. El que si canvia dràsticament és que a l'inici de cada partida el jugador haurà d'escollir un personatge amb unes habilitats que el diferencia de la resta. Amb això ja es diferencia una partida d'una altra. A més, la generació del castell, en aquest cas, és semblant a les ja mencionades, encara que en aquest cas les sales varien bastant a causa que s'ha de tenir en compte l'eix vertical.

Figura 2.4.1.3: Imatge de Rogue Legacy 2

2.4.2 Conclusions Estudi de Mercat

El que s'ha pogut extreure dels jocs presentats és el següent. La majoria d'ells utilitza una estètica amigable. A més, la generació dels nivells no sol variar molt entre videojocs a causa de la partició de sales que tenen. On es diferencien més és en com millora el jugador a través de la partida, es veu com en el primer joc la varietat d'objectes pot arribar a canviar molt la jugabilitat, mentre que en UnderMine es segueix modificant la jugabilitat però sense ser tan extrem, i llavors tenim l'últim videojoc que demana més esforç al jugador per passar-se els nivells. A part de que en tots ells el jugador va sol, a excepció d'acompanyants que no tenen tant de pes en el joc, a causa que no poden rebre mal i tenen una funció de suport molt limitada, mentre que en aquest projecte els aliats són una part fonamental en la jugabilitat.

Donat tot això i com es pot veure a la Taula 2.4.2.1, aquest projecte utilitzarà una estètica amistosa en Pixel Art, com a UnderMine, però més senzilla. Comptarà amb una generació de nivells procedurals semblant a The Binding of Isaac ja que hi hauran sales grans i petites. Finalment, hi haurà un sistema de millora semblant a Rogue Legacy 2, ja que no hi haurà objectes en el prototip i tota la millora que obtingui el jugador es donarà gràcies a la seva habilitat i a les habilitats que obtingui dels aliats.

	Estètica	Tipus de Vista	Millora en la partida
The Binding of Isaac	Pixel Art	Axonomètrica	Objectes
UnderMine	Pixel Art	Axonomètrica	Objectes
Rogue Legacy 2	2D	Frontal	Habilitat del jugador
Pawn Sacrifice	Pixel Art	Axonomètrica	Habilitat del jugador i habilitats d'aliats

Taula 2.4.2.1: Matriu de Competivititat

2.5 Públic Objectiu

A l'hora de dissenyar un videojoc s'ha de tenir en compte per a quin tipus de públic s'està fent. En aquest cas s'està produint per a persones de 15 anys cap amunt, ja que l'estètica no deixa a ningú enrere, i la dificultat pot ser abrumadora en un inici, ja que potser a les primeres partides el jugador no arribarà a l'enemic final, però, amb una mica d'esforç, és molt assequible el passar-se el joc. Per suposat, el tipus de jugador que es busca ha de voler una certa dificultat al moment de jugar i tenir ganes de rejugar el videojoc ja que és una cosa que es busca en el gènere dels *Rogue-Likes*.
Ara bé, per ser més específic utilitzaré una classificació de jugadors segons Richard Bartle que els classifica segons la personalitat i comportament envers al joc. Aquests es classifiquen de la següent manera:

- **Achiever:** tenen com a objectiu resoldre tots els reptes que proposa el joc. Obtenen plaer en aconseguir la màxima quantitat de punts, assoliments i recompenses. Estan més interessats a interactuar amb el món.
- **Explorer:** gaudeixen amb l'exploració, el descobriment i l'aprenentatge de qualsevol cosa nova o desconeguda del joc. Estan més interessats en interactuar amb el món.
- **Socializer:** són els jugadors que senten atracció pels aspectes socials per damunt de l'estratègia o obtenció de recompenses. Estan més interessats a interactuar amb els jugadors.
- **Killer:** són els jugadors que troben el plaer en la competitivitat i tenen l'objectiu d'arribar a ser els millors jugant. Estan més interessats a interactuar amb els jugadors.

En aquest videojoc el tipus de jugadors que es busca és **l'explorer**. Això és a causa que l'explorer voldrà veure totes les sales de cada nivell, ja que pot ser que no les hagi vist abans i, a més, voldrà provar les diferents combinacions d'habilitats dels personatges, pel que jugarà el joc repetides vegades.

3. Planificació

En aquest apartat explico com em vaig organitzar inicialment de manera que pogués tenir un calendari en el qual hi especifiqués quan fer cada feina. Aquest calendari el vaig fer a partir dels diagrames de Grantt que, com es pot veure a la Figura 3.1, va del 15 de febrer de 2021, data en la que vaig començar a treballar, fins el 10 de juny de 2021, data d'entrega. Aquesta organització és aproximada ja que en el transcurs del treball algunes tasques poden haver donat problemes i haver necessitat més hores de feina, o al contrari, haver estat més fàcils del que pensava.
Finalment, les tasques nombrades en el diagrama són un recull general del que s'havia de fer per a desenvolupar el videojoc, ja que, per exemple, en el disseny i implementació dels personatges també s'hi adhereix el sistema de combat.

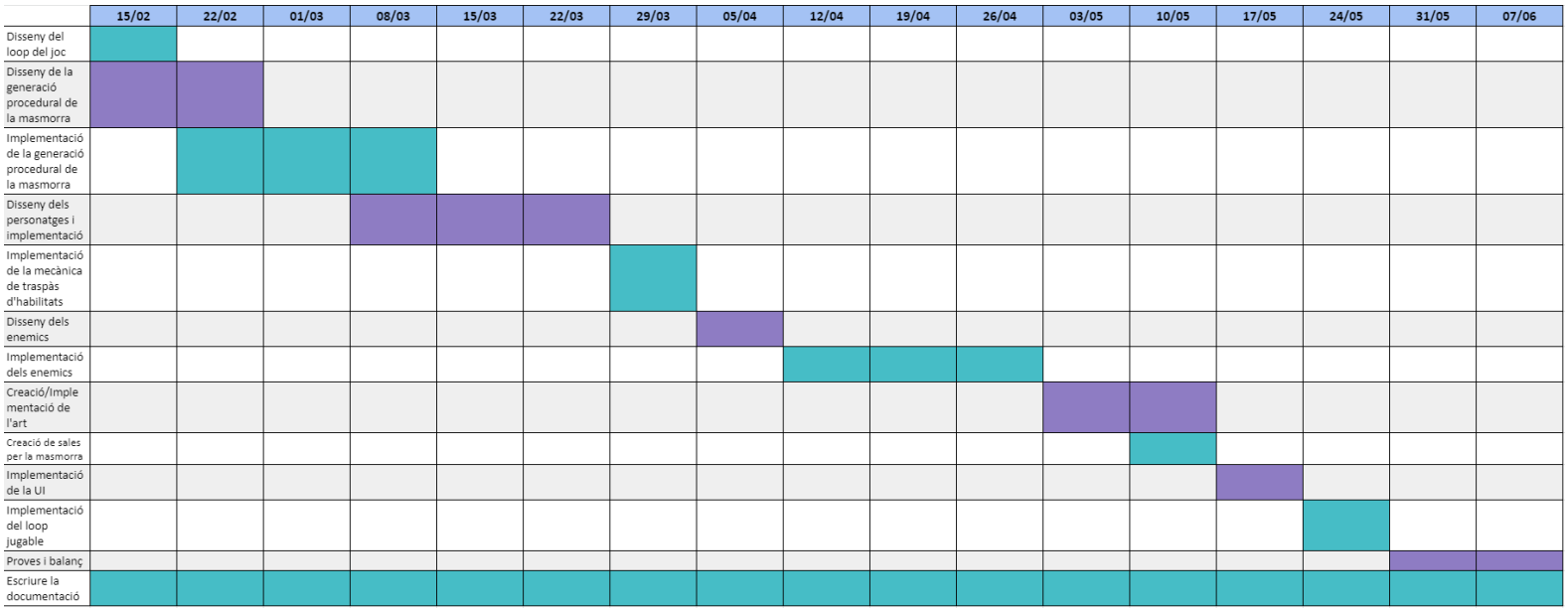

Figura 3.1: Diagrama de Grantt del projecte

La metodologia de treball que he seguit durant tot el projecte ha estat de dissenyar, implementar i provar. Així, per a qualsevol tasca proposada, a l'hora d'implementar-la, ja es tenia en compte els possibles problemes que podia comportar al relacionar-se amb altres sistemes del videojoc. Tot i això, la part de proves ha estat molt important ja que, per norma general, apareixen errors inesperats que s'han d'arreglar.

4. Marc de Treball i Conceptes Previs

En aquest apartat posaré al lector en context sobre el videojoc per a que, en els següents apartats, es pugui entendre tot de forma més senzilla.

Primer de tot, explicaré el funcionament del videojoc. En cada nova partida el mapa es genera de forma procedural pel que sempre serà diferent. Aquest mapa està compost de sales connectades entre si per portes, les quals es tanquen quan el jugador entra per primer cop en elles, i s'obren quan tots els enemics han estat eliminats. El jugador, acompanyat des d'un inici per 3 aliats, ha d'anar avançant en cada pis fins a trobar la baixada al següent. Al fer-ho, ha d'escollir quin aliat eliminar i quina de les seves dues habilitats es quedarà el personatge controlat pel jugador. Aquest procés es repeteix fins a arribar al quart pis, on el jugador es troba sol i ha de lluitar contra l'enemic final. Si aquest és eliminat pel jugador, la partida s'acaba amb una victòria. En canvi, si el personatge principal mor en qualsevol moment, es perd la partida.
L'enemic final és conegut en el món dels videojocs com a "boss" i aquest ha de suposar un repte major al vist durant la partida. En aquest cas, disposa d'atacs únics i d'una gran quantitat de vida.

En segon lloc, els videojocs dels quals he agafat referències i he après són els ja mencionats a l'Apartat 2 d'aquest document, en l'estudi de mercat, on també parlo del que he utilitzat de cada un d'ells, què n'he après i en què destaca aquest projecte.

Per altra banda, el motor de videojocs seleccionat ha estat el Unity3D a causa que permet treballar en videojocs 2D, té eines i llibreries molt útils i, a més, és el motor amb el que em sento més còmode ja que és al que li he dedicat més hores. Cal afegir que Unity3D té una gran comunitat, i això permet que sigui molt accessible quan es té algun dubte sobre certs aspectes del motor, o sobre com implementar una mecànica o un sistema.

5. Disseny del Videojoc

Aquest és un dels apartats més importants del document, ja que explicaré el perquè de les decisions preses en aquest projecte i com les he portat endavant.

5.1 Mecàniques

5.1.1 Espai de Joc

Com ja he explicat amb anterioritat, en els videojocs del gènere *Rogue-Like* el món on es juga està format de forma aleatòria/procedural en cada nova partida. Aquest joc no és una excepció. En cada nivell de la masmorra se li demana a l'algoritme de generació de nivells uns mínims, aquest són els següents:

- L'inici del nivell està on hi havia el final del nivell passat, en el primer nivell l'inici es troba al centre.
- Hi ha d'haver un mínim de vuit habitacions i un màxim de 13. Això està fet així per a donar certa regularitat en les dimensions de cada nivell.
- Ha d'aparèixer un mínim d'una habitació anomenada "cul de sac", la qual només té una porta i s'utilitza per, entre les que apareixen, com a habitació per passar al següent nivell.

Un exemple d'aquesta generació de nivells es pot veure en la Figura 5.1.1.1.

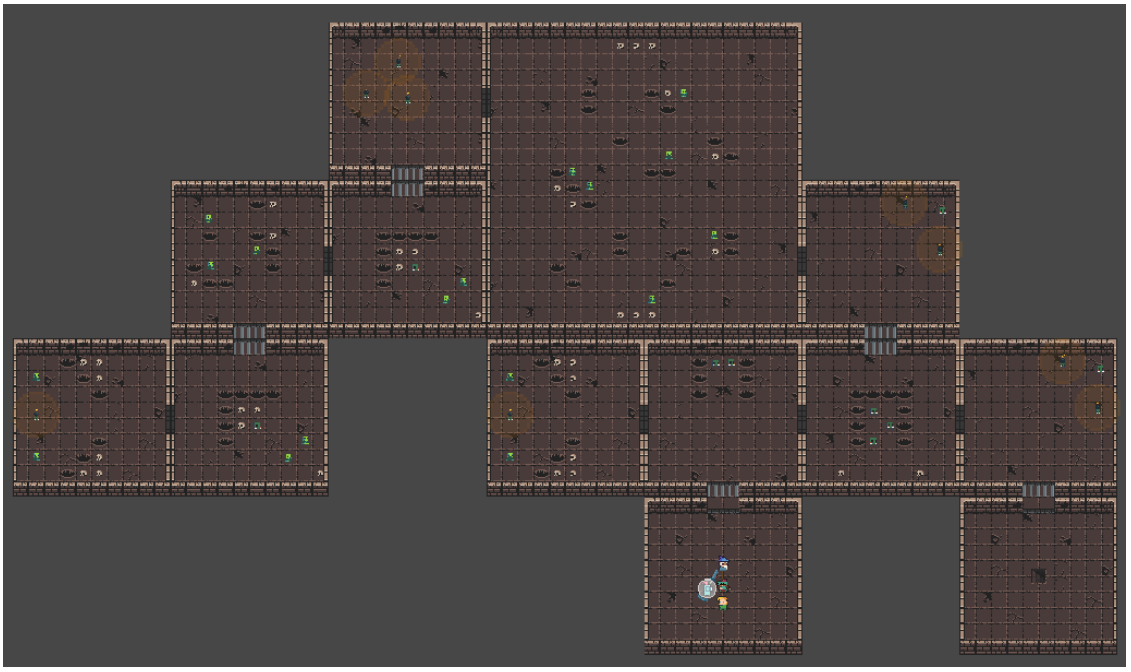

Figura 5.1.1.1: Nivell generat de Pawn Sacrifice

L'excepció a aquesta generació de nivells n'és l'últim, el pis número 4, on es lluita contra l'enemic final. Aquest nivell consta de l'habitació inicial i de l'habitació on es lluita contra l'enemic, que està just a sobre. Aquesta decisió la vaig prendre perquè en el joc no hi ha cap manera de curar-se, a no ser que es passi de nivell o el personatge es mori i sigui reviscut per una habilitat. Per això crec que a l'últim combat s'hi hauria d'anar amb tota la vida per no frustrar el jugador i que pensi que si hagués arribat a la lluita a plenes capacitats, potser hagués guanyat.

Altrament, en cada sala hi ha elements que cada una d'elles té, a excepció de la sala inicial i la sala final. El que contenen és el següent:

- **Parets:** serveixen de límits de la sala per a delimitar una de l'altre.
- **Obstacles:** s'utilitzen per donar varietat a cada sala i per donar dinamisme al combat a causa que el jugador ha d'estar pendent per on va al esquivar als enemics. No totes en tenen, ja que depèn del disseny de la sala.
- **Enemics:** són monstres que funcionen amb intel·ligència artificial i ataquen al personatge principal i als aliats. Varien en cada sala, i no n'hi ha a les sales inicial i final.
- **Portes:** són el pas de sala a sala. Es tanquen quan el jugador entra en una sala on el jugador encara no hi havia estat, i s'obren quan tots els enemics de la sala han mort.

A continuació, en la Figura 5.1.1.2, podem veure cada part d'una habitació amb tots els elements mencionats.

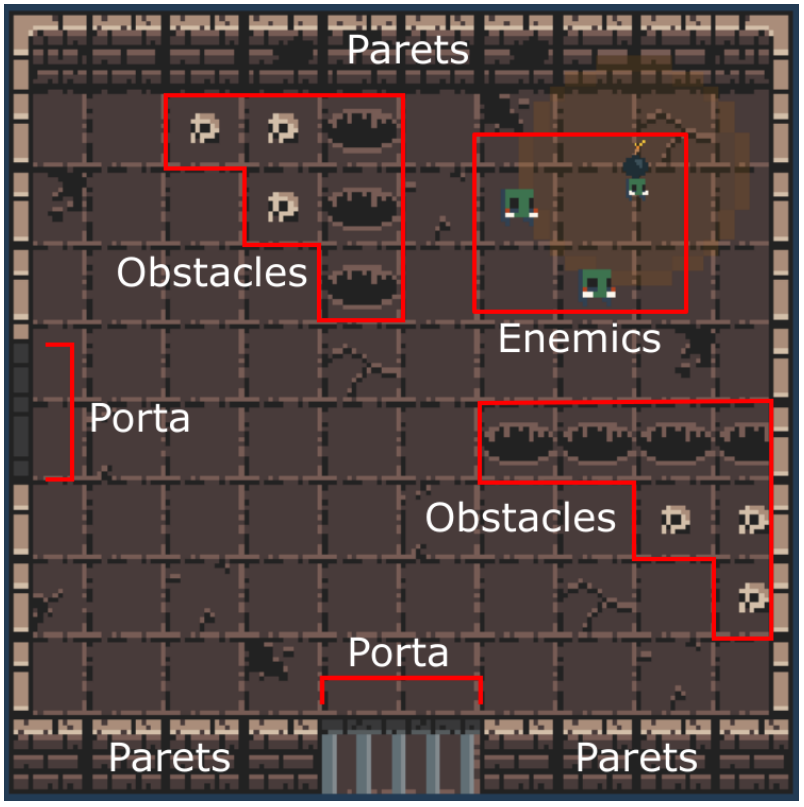

Figura 5.1.1.2: Sala de Pawn Sacrifice

5.1.2 Mecànica de joc, reptes que ha d'afrontar el jugador

Per a destacar aquest projecte dels demés videojocs amb els que comparteix gènere, havia de pensar en una mecànica principal que destaqui i sigui diferent, mai vista en el gènere. Aquesta, com ja s'ha vist, és sacrificar un dels aliats a l'arribar al final de cada nivell, a canvi d'una de les seves habilitats, la qual el personatge principal aconseguirà. Per tan, l'efecte que s'aconsegueix amb això és que el jugador hagi de prendre decisions importants al acabar cada nivell, perquè cada una de les habilitats és útil i al elegir-ne una, s'està prescindint de la segona habilitat del personatge. A més, a mesura que s'avança en la partida el jugador ha de dependre cada vegada menys en els seus aliats ja que aquests van desapareixent, i ha de dependre més en ell mateix, ja que quants menys aliats hi hagi, més poderós serà per ell sol.

Actualment, en el prototip hi ha 4 personatges, pel que hi ha 8 habilitats diferents. Amb aquests números ja hi ha moltes possibilitats de combinacions entre habilitats que poden fer diferent cada partida. Per tant, en la versió final del videojoc, on hi haurà molts més personatges, les possibilitats seran molt majors.

5.1.3 Objectes, Recursos i Interaccions que pot fer el Jugador

Per aquest prototip no he dissenyat objectes amb els que el jugador pugui interactuar, són una idea a futur. Això és a causa que són un afegit al videojoc i, per tant, no pertanyen al nucli d'aquest, ja que aportarien millors estadístiques pels jugadors com podria ser fer més mal o tenir més vida. En aquest aspecte, aquest projecte s'assemblaria al joc Rogue Legacy 2, del que he parlat en l'Apartat 2.4.1, en el qual els objectes no són el principal atractiu. En canvi, els recursos dels que disposa el jugador són els aliats i la vida dels personatges. La vida compleix la funció de que el jugador vagi en compte a l'hora de lluitar contra els enemics, ja que si la vida del personatge principal arriba a 0, la partida s'acaba. Els aliats poden morir, ja que al passar algunes sales seran reviscuts automàticament. No obstant, la vida del personatge principal només es recupera al passar de pis o al ser reviscut per una de les habilitats del joc.
Quan parlo d'aliats com a recurs em refereixo a que s'utilitzen com a suports en el combat contra enemics i per obtenir les seves habilitats. Com he dit amb anterioritat, el jugador aconsegueix altres habilitats al passar de nivell, que fa interactuant amb l'escala que simula el pas entre pisos. Al fer-ho, es desplega una interfície on surten els aliats, el jugador en selecciona un i després selecciona una de les dues habilitats que posseeixen per a aconseguir-la.

5.1.4 Economia Interna del Joc

En aquest projecte no hi ha implementada cap tipus d'economia. El que ha de motivar el jugador a jugar el joc és poder acabar amb l'enemic final a través d'haver millorat en el joc al fer varies partides, i també, el voler provar diferents combinacions d'habilitats.

5.1.5 Interfícies

Les interfícies en aquest treball consten de menús (l'inicial, el de pausa, el de selecció d'habilitats i el de victòria o derrota) i d'informació sobre la partida (la vida dels personatges i el minimapa). Els menús són senzills però funcionals. El d'inici permet jugar, entrant a la partida directament, o sortir del joc. El menú de pausa permet tornar al joc, anar al menú inicial o sortir del joc. El menú de victòria o derrota té les mateixes funcionalitats que el de pausa, però sense poder tornar a jugar. En canvi, el de selecció d'habilitats primer demana a l'usuari que seleccioni un personatge, per després poder seleccionar una de les seves habilitats.
D'altra banda, la interfície que mostra la vida dels jugadors ho fa amb un número de cors, que es correspon a la vida de cada aliat i del personatge principal, ensenyant la icona de cada personatge corresponent. Finalment, el minimapa ensenya les habitacions per les que ha passat el jugador, per a que es pugui situar en la habitació en la que està. A més, també mostra les habitacions adjacents a una habitació visitada a les quals encara no ho han estat, per a que tingui el coneixement de per on pot anar.

5.2 Estudi i Disseny de Personatges

5.2.1 Estètica i Pes Narratiu

Primer de tot, l'art no és la meva especialitat i tampoc se'm dóna bé, pel que vaig optar a obtenir la majoria dels elements artístics de tercers. En la cerca d'aquests assets vaig poder trobar un paquet, el qual es pot veure a la Figura 5.2.1.1, molt complert en el que venen tant els personatges, com parts de l'escenari i diferents objectes. Aquest també s'adhereix a la temàtica que estava buscant, la qual és un món fantàstic ambientat en l'edat mitjana. A més, pertany a una estètica amistosa que, com he mencionat a l'Apartat 2.4.2, és molt utilitzada en el mercat. A aquest paquet se li suma l'art que he produït:

- L'arc de l'arquer juntament amb la fletxa, en la Figura 5.2.1.2.
- La vareta del mag juntament amb la bola de foc que dispara, en la Figura 5.2.1.3.
- L'escut i l'escut del darrere del cavaller, en la Figura 5.2.1.4.
- Les animacions de mort dels personatges que en tenen, en la Figura 5.2.1.5.

- L'enemic anomenat *bomber*, el qual és un enemic del paquet amb una bomba a sobre, en la Figura 5.2.1.6.
- Les portes amb les seves animacions, en la Figura 5.2.1.7 i Figura 5.2.1.8.

Figura 5.2.1.1: Paquet d'art de l'autor 0x72

Figura 5.2.1.2: Arc i fletxa de l'arquer

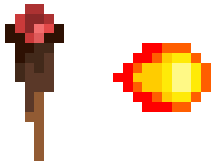

Figura 5.2.1.3: Vareta i bola de foc del mag

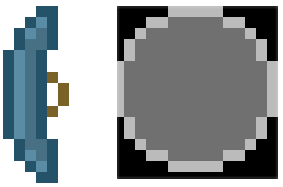

Figura 5.2.1.4: Escut del darrere i escut del cavaller

Figura 5.2.1.5: Animacions de mort

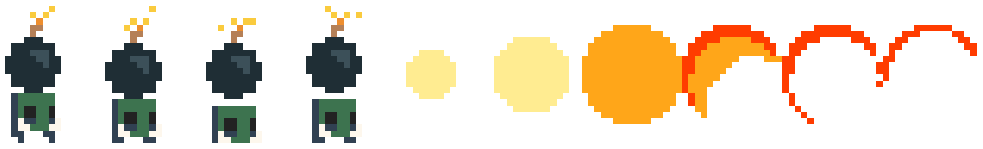

Figura 5.2.1.6: Bomber

Figura 5.2.1.7: Porta vista des del costat

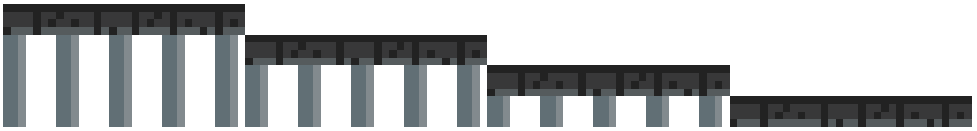

Figura 5.2.1.8: Porta vista des de devant

Havent mostrat l'estètica dels personatges i els altres elements, toca parlar del pes narratiu dels personatges. Aquest no és molt elevat, perquè com ja en parlaré a l'Apartat 5.3, el joc careix d'una història. Per tant, el pes narratiu dels personatges recau en l'ambientació, ja que veient-los es pot saber que ens situem en l'edat mitjana en un món fantàstic on existeix la màgia i éssers estranys. A part, l'aspecte ens pot donar informació del que poden arribar a fer. Per exemple, l'enemic que s'anomena *slime* i sembla una mucositat verda, al morir es divideix en petits *slimes* com en molts altres videojocs tal i com es pot veure en la Figura 5.2.1.9.

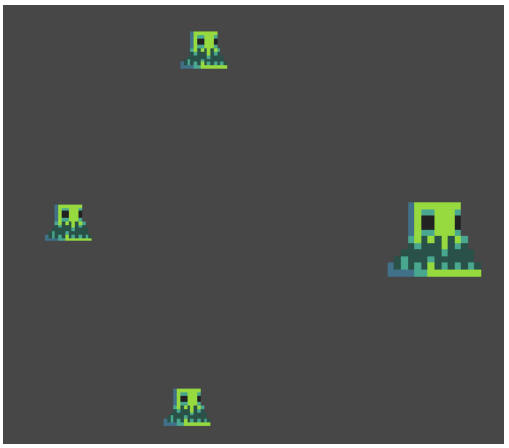

Figura 5.2.1.9: Slime normal a la dreta amb els slimes resultants d'una divisió

5.2.2 Personatges Jugables

Hi ha dos tipus de personatges jugables, els que ataquen a distància (arquer i mag) i els que ataquen cara a cara (cavaller i ensinistrador). Els primers llencen projectils de de la seva posició cap a on apunten, mentre que els segons desplacen la seva arma cap a on apunten per atacar. Quan el jugador ataca amb el personatge, aquest es veu relantitzat a la meitat de la seva velocitat de moviment, això ho he fet així per dificultar el truc d'atacar i fugir indefinidament d'un enemic.

Quan aquests personatges són aliats passen a ser controlats per una intel·ligència artificial, la qual es pot comportar de dos maneres diferents. Si el personatge ataca a distància, el comportament és el de buscar un enemic al qui atacar. Si el té a vista, comença a atacar-lo. En canvi, si no te una línia directa d'atac, es mou cap a ell fins que ho pugui fer. Per altra banda, si el personatge ataca cara a cara, aquest busca un enemic al qui atacar i quan el troba, va directa cap a ell. Al tenir l'enemic en distància d'atac, l'ataca. En els dos casos hi ha comportaments compartits, un d'ells essent que, quan no hi cap enemic al qui atacar, segueixen al jugador. L'altre és que, quan un enemic es troba massa proper, intenta moure's en la direcció contrària per evadir-lo.

A més, quan un personatge aliat es mor, queda mort al terra. Això es manté així fins que el jugador passa de nivell o fins que completi 5 habitacions. Al moment que el jugador se les passi, els personatges morts reviuran i apareixen a la següent habitació.

D'altra banda, la majoria de personatges i enemics tenen animacions per caminar i morir. La animació de mort s'executa quan la vida d'un personatge arriba a zero. En canvi, la animació de caminar s'executa de forma continua, a excepció de l'enemic final ja que té una animació

al fer un atac de distància com es pot veure en la Figura 5.2.2, a més, les imatges dels personatges sempre estan de cares al seu objectiu.

Figura 5.2.2: Animació d'atac a distància de l'enemic final

Finalment, he fet que cap dels personatges es diferenciï en les estadístiques base. Això ho he fet a causa de que vull que la diferència entre personatges siguin les seves habilitats, i que més enllà de la combinació d'aquestes que el jugador pugui tenir, no senti que hauria tingut una avantatge superior si li hagués tocat un altre personatge.

Explicat això, ara ho faré amb cada un d'ells en profunditat.

5.2.2.1 Arquer

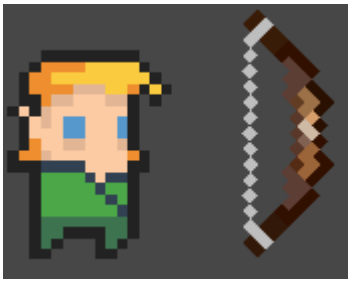

Figura 5.2.2.1: Arquer

Com es pot veure a la Figura 5.2.2.1, l'arquer posseeix un arc amb el qual llança fletxes a distància. Es caracteritza per anar vestit de verd i tenir el cabell ros.

D'altra banda, les habilitats que posseeix per defecte són les següents:

- **Penetració:** els projectils penetren als enemics i poden impactar a altres enemics. Per defecte els projectils impacten en el primer enemic i es destrueixen. En els personatges que no disparen projectils, aquesta habilitat els permet atacar a varis enemics en un sol atac de l'arma, ja que en un cas normal, en impactar al primer enemic, l'atac es cancela.

- **Visió:** aquesta altra habilitat permet al jugador poder veure en el minimapa la sala final del nivell, podent així tenir una idea de cap a on anar per passar al següent nivell.

5.2.2.2 Mag

Figura 5.2.2.2: Mag

Com es veu a la Figura 5.2.2.2, el mag es caracteritza per portar una vareta màgica amb la qual dispara boles de foc als enemics. Aquest es caracteritza per anar vestit de blau, amb un barret i una gran barba blanca. Les habilitats que posseeix per defecte són aquestes:

- **Cremar:** aquesta habilitat permet al personatge cremar els enemics amb els projectils/atacs. Al cremar un enemic, aquest rep dany cada pocs segons de forma continuada.
- **Reviure:** dóna la possibilitat de tornar a la vida a un jugador quan la seva vida arribi a zero. Aquesta habilitat només es pot utilitzar una vegada per pis, i només li retorna la meitat de la vida al personatge reviscut.

5.2.2.3 Cavaller

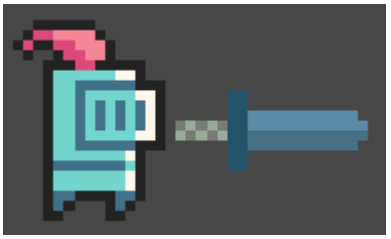

Figura 5.2.2.3: Cavaller

Com es veu a la Figura 5.2.2.3, el cavaller es caracteritza per portar una espasa amb la que ataca cos a cos. Aquest es caracteritza per anar vestit amb una armadura de cos sencer amb una ploma vermella al capdamunt.

A més, les seves habilitats per defecte són:

- **Escut:** aquest escut rodeja al personatge permetent rebre un atac enemic sense que li baixi la vida, però fent que l'habilitat desapareixi. Torna a aparèixer un cop el jugador ha complert tres habitacions.
- **Escut de darrere:** aquesta altra habilitat és un escut que està col·locat a la posició contrària on el personatge apunta l'arma, i permet parar projectils enemics. En aquest prototip només serveix per a l'enemic final ja que és l'únic de dispara projectils, però en la versió final del joc hi hauran més enemics que ataquin a distància.

5.2.2.4 Ensinistrador

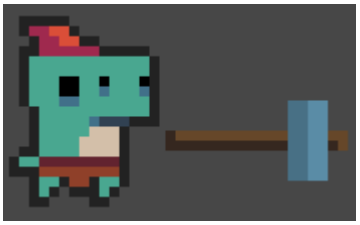

Figura 5.2.2.4: Ensinistrador

Com es veu a la Figura 5.2.2.4, l'ensinistrador es caracteritza per portar un martell de guerra amb el que ataca als enemics cara a cara. Aquest es caracteritza per ser l'únic personatge jugable no humà, passant a ser una especie de reptil humanoide.

D'altra banda, aquestes són les habilitats que té per defecte:

- **Paralitzar:** aquesta habilitat permet al personatge deixar immòbils als enemics amb l'atac/projectil durant varis segons. Aquesta habilitat només es pot utilitzar en un enemic cada 7 segons, temps en el qual no té efecte.
- **Minion:** minion permet fer aparèixer un petit ajudant al costat del personatge en cada sala amb enemics. Aquest consta d'una simple intel·ligència artificial que el fa anar cap a l'enemic més proper i, si la sala s'ha netejat d'enemics i segueix viu, es mor sol. El minion no pot atacar, només serveix com una simple distracció per a que els personatges puguin atacar als enemics.

5.2.3 Enemics

Cada sala, a excepció de les sales inicials i finals de cada nivell, tenen enemics. Aquests tenen comportaments diferents en quan a com atacar als personatges jugables que estiguin a la sala. No obstant, quan el jugador encara no ha arribat a la sala, esl enemics estan "apagats" fins que aquest no hi entri. A continuació explicaré cada un d'ells en profunditat.

5.2.3.1 Charger

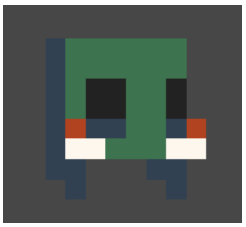

Figura 5.2.3.1: Charger

Aquest enemic, que es pot veure en la Figura 5.2.3.1, s'anomena charger. Aquest nom ve donat a causa de com ataca al jugador i als aliats. Aquest atac consisteix en una càrrega cap a l'objectiu. El charger fa dany per contacte, el que vol dir que si un personatge el toca, rep dany. Això fa que la seva càrrega sigui molt efectiva si no s'esquiva.

El charger és l'enemic que va més ràpid, quan carrega contra un objectiu, però va en una velocitat normal quan camina. D'altra banda, té una vida dintre del que es pot considerar normal, ja que ha de aguantar varis atacs per a que pugui fer quelcom en un combat.

A la Figura 5.2.3.2 es pot veure el comportament que segueix. Quan l'enemic troba un objectiu al qui atacar, mira si és possible atacar en línia recta fins a la seva posició. Si és possible, ataca, si no ho és, es mou a una posició propera aleatòria. En els dos casos, a continuació torna a buscar un nou objectiu.

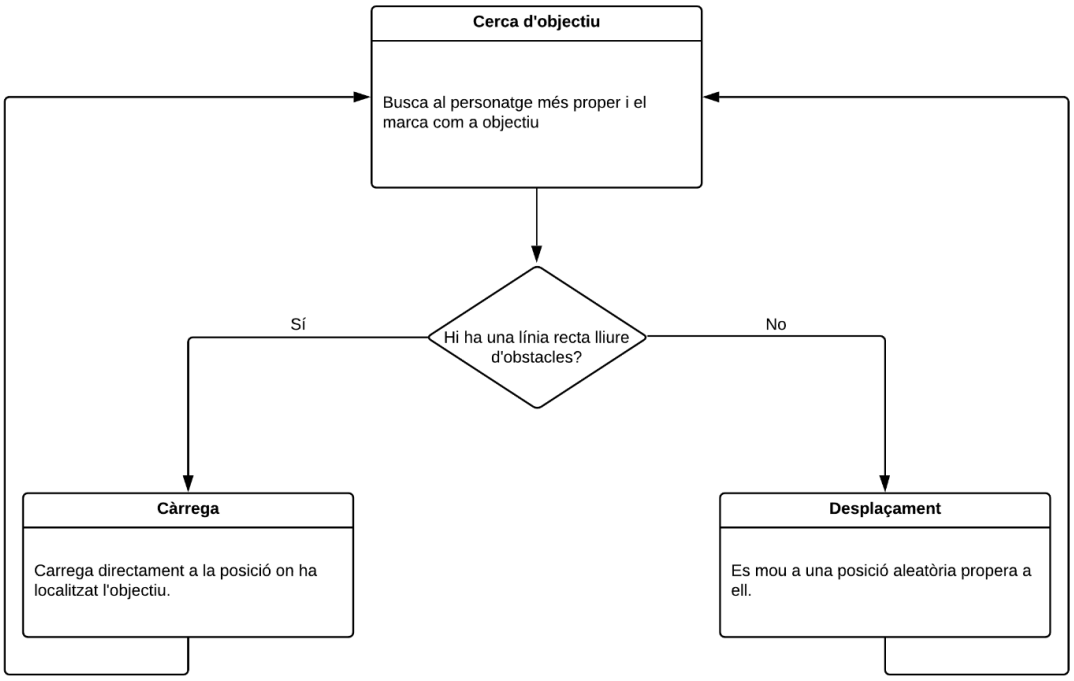

Figura 5.2.3.2: Diagrama del comportament del charger

5.2.3.2 Bomber

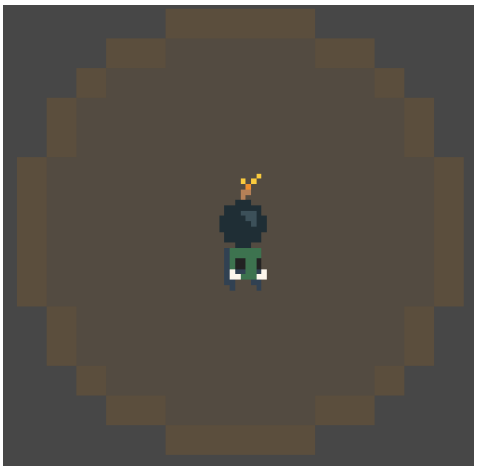

Figura 5.2.3.3: Bomber

Aquest enemic és anomenat bomber, i com es pot veure a la Figura 5.2.3.3, d'aspecte s'assembla al charger, amb la característica que aquest porta una bomba a sobre el cap. A més, està envoltat per una circumferència. El bomber no fa dany per contacte, ja que la seva via per a fer mal al jugador és explotar. L'atac funciona de tal manera que, quan troba un objectiu, va cap a ell, i quan aquest o qualsevol altre personatge entra en el radi de la circumferència que l'envolta, la bomba s'activa, indicant-ho com es pot veure a la Figura

5.2.3.4, on bàsicament s'augmenta i es disminueix la opacitat de la circumferència. Al cap d'uns segons, si no es mor, explota, el que causa fins a dos unitats de dany a qualsevol personatge dintre del radi d'explosió. D'altra banda, al explotar l'enemic s'autodestrueix.

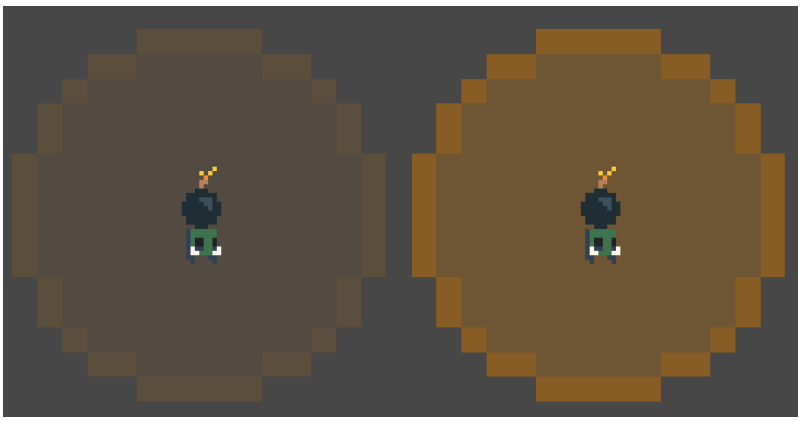

Figura 5.2.3.4: Bomber activant-se

A més, el bomber es caracteritza per ser ràpid però amb poca vida, a causa que el seu atac és l'únic del videojoc que pot arribar a fer dos unitats de dany. Això fa que pugui ser relativament fàcil matar-lo, però que si el jugador no ho fa, pateix les conseqüències.

En la Figura 5.2.3.5 es pot veure el comportament del bomber. Aquest busca un objectiu, quan el troba el segueix, i cada dos segons, o si l'objectiu ha mort, en busca un de nou, que pot ser el mateix. En qualsevol moment, pot activar-se, cosa que ja he explicat en aquest apartat.

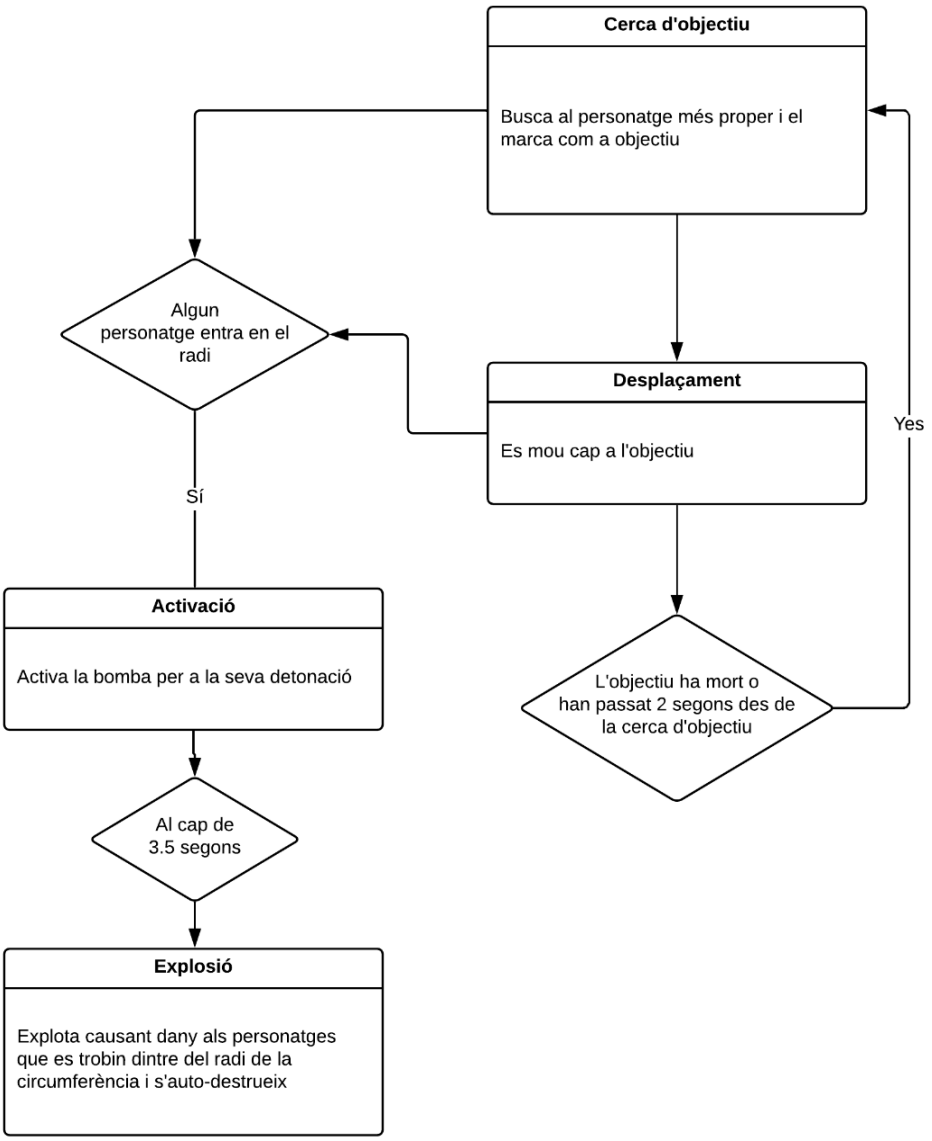

Figura 5.2.3.5: Diagrama del comportament del bomber

5.2.3.3 Slime

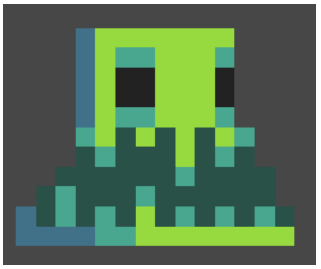

Figura 5.2.3.6: Slime

Aquest enemic, com es pot veure a la Figura 5.2.3.6, s'anomena slime i sembla ser una viscositat verda. El slime és un enemic molt comú en els mons fantàstics de videojocs on normalment al morir es divideix en petits slimes. En aquest joc no hem volgut fer l'excepció, però, quan es divideix en tres petites versions d'ell mateix, les quals tenen un punt de vida menys, dues d'elles s'intenten fusionar per formar un slime gran de nou, mentre que l'altre va a l'atac, com es pot veure a la Figura 5.2.3.7. Aquest slime gros és més dèbil que l'original, a causa de que té la mateixa vida que els petits slimes que l'han format. Per altra banda, ataca igual que el charger, de forma que fa dany als personatges per contacte

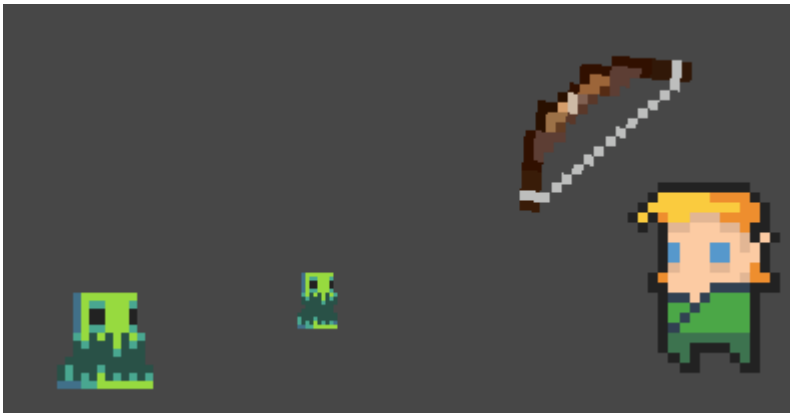

Figura 5.2.3.7: Slime gran format per la fusió de dos petits i slime petit

El slime posseeix una velocitat semblant a la del charger quan camina, pel que és l'enemic més lent del videojoc. A més, la seva vida és la mateixa que la del bomber, ja que al matar-lo es divideix en més slimes, pel que no necessita tanta vida com el charger.

A continuació, en la Figura 5.2.3.8, es pot veure el comportament d'un slime normal, mentre que en la Figura 5.2.3.9 es veu el comportament d'un slime el qual s'ha de fusionar amb un altre. En el primer es pot veure el que ja hem explicat, afegint que quan un slime s'ha de dividir, si les subdivisions haguessin de tenir zero o menys de vida, directament el slime no es divideix i es mor. D'altra banda, en el segon es veu com la fusió es dona per la proximitat entre els dos slimes.

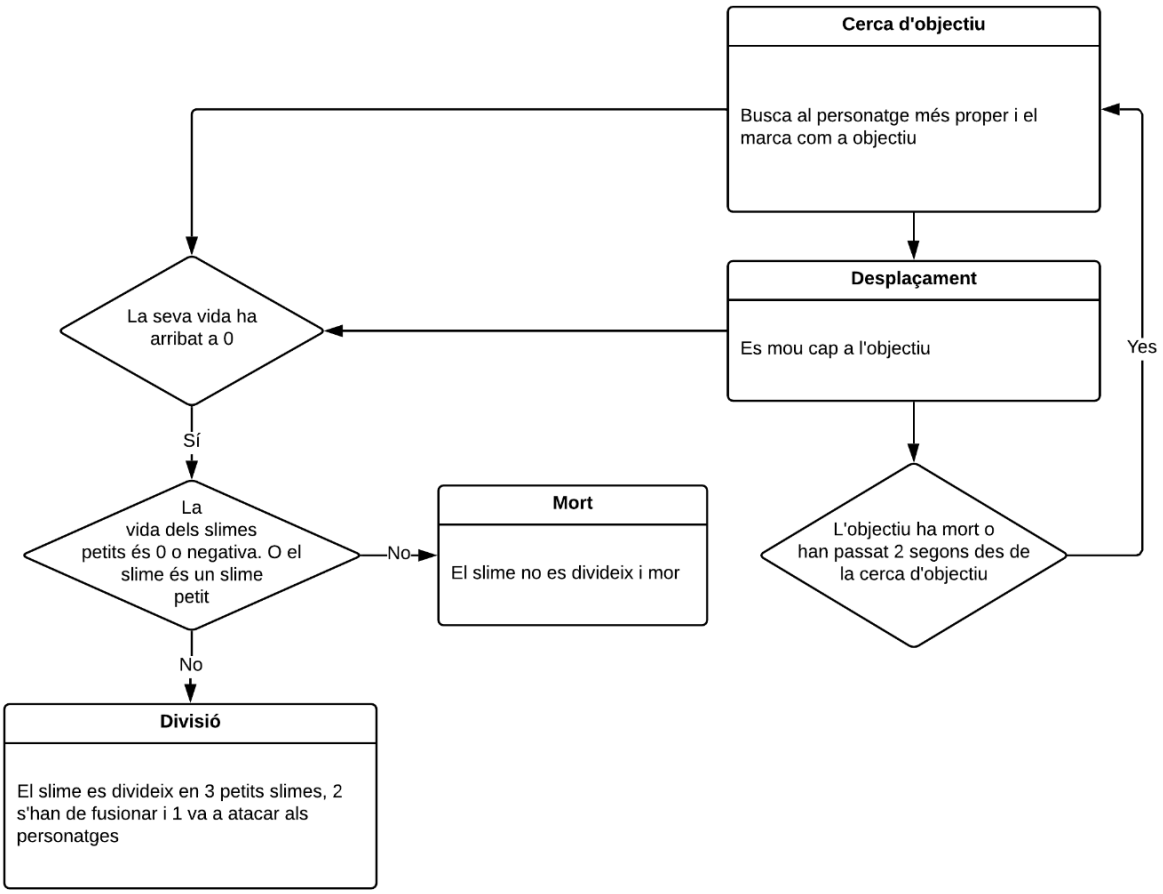

Figura 5.2.3.8: Diagrama del comportament d'un slime normal

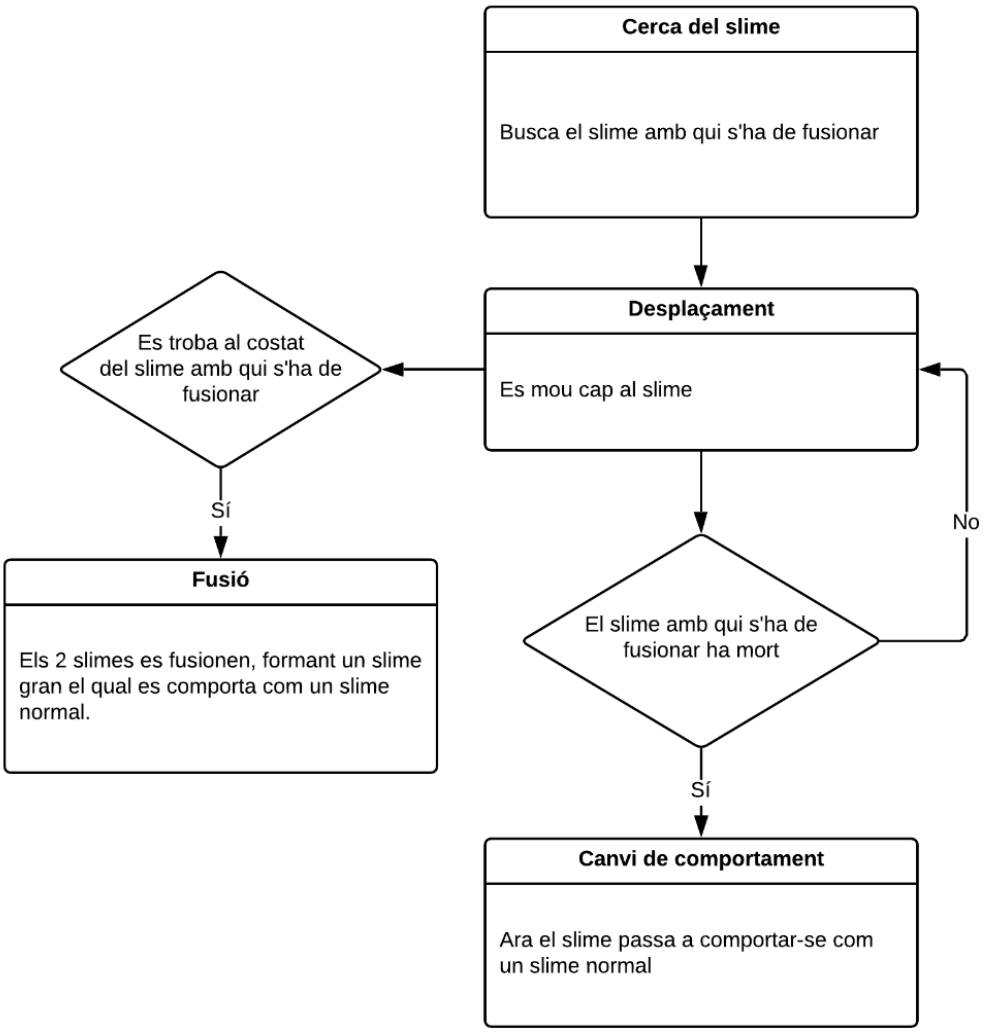

Figura 5.2.3.9: Diagrama del comportament d'un slime que s'ha de fusionar

5.2.3.4 Boss

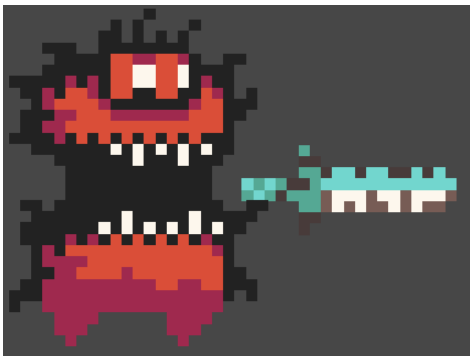

Figura 5.2.3.10: Boss

Aquest és l'enemic final el qual s'anomena boss i, com es pot veure a la Figura 5.2.3.10, és l'enemic amb un aspecte més grotesc. A més, el caracteritza el fet de ser l'únic enemic posseïdor d'una espasa.

El boss, a part de fer dany per contacte, ataca amb l'espasa d'igual manera que ho fa un personatge que ataca cos a cos, atacant amb ella cada pocs segons mentre va perseguint al jugador. A més, és l'únic enemic que compta amb un atac a distància, el qual són boles de foc llançades en diverses direccions a la vegada. Aquest atac el fa cada cert número de segons i les direccions en què són llançades les boles varien cada vegada igual que el temps en què es produeix l'atac.

Aquest enemic compta amb una velocitat per sobre de la mitjana dels enemics tot i que inferior a la càrrega del charger. No obstant, la seva quantitat de vida és absurdament elevada si la comparem amb els seus altres companys.

A continuació, en la Figura 5.2.3.11, es pot veure el comportament que segueix el boss. Cal destacar que quan es produeix l'atac a distància el boss es queda immòbil en la seva posició fins que ha acabat.

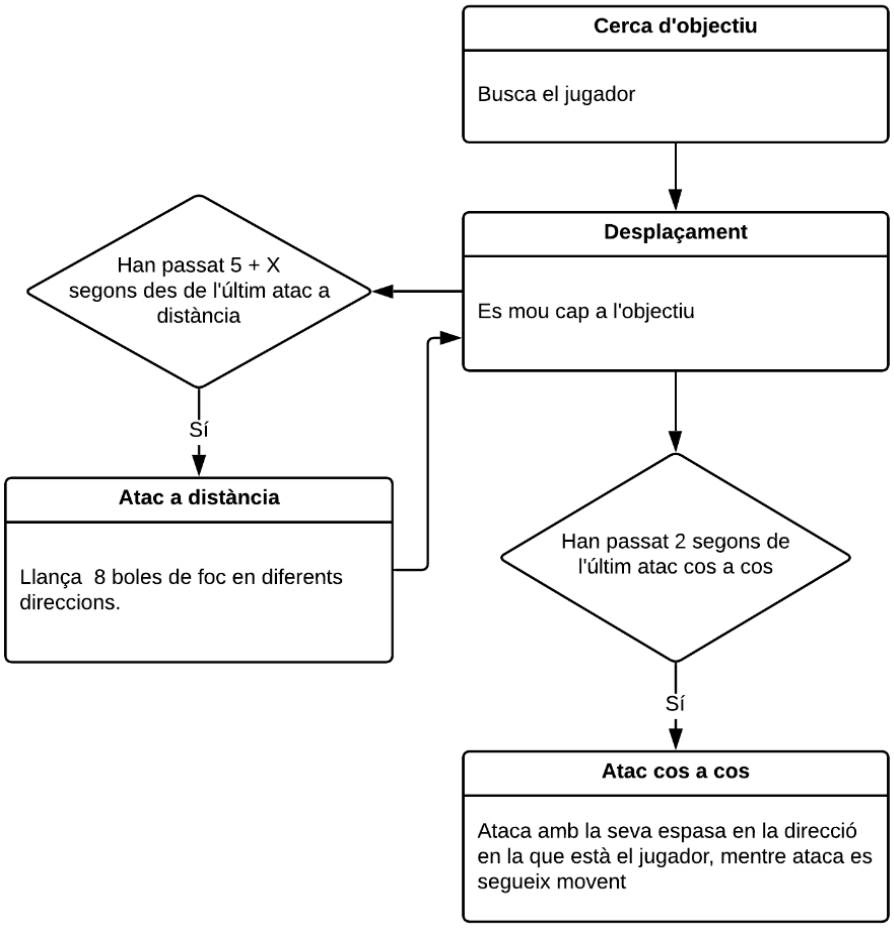

Figura 5.2.3.11: Diagrama del comportament del boss

5.3 Narrativa

Com ja he explicat amb anterioritat en aquest document, aquest projecte no té una història que seguir. Això és a causa que el joc es basa en disfrutar les mecàniques i el combat. Per tant, la única narrativa que es pot extreure d'aquest projecte és la seva ambientació.

El món del videojoc està ambientat en un món fantàstic on la màgia i els éssers estranys existeixen. Es pot veure tant els enemics, ja que aquests no existeixen en el nostre món, com amb els personatges jugables, entre els quals es troba un mag que llança boles de foc i l'ensinistrador el qual és un reptil humanoide. A més, també es pot veure que està ambientat en l'edat mitjana, això es pot veure en elements com el cavaller i l'arquer, personatges típics de la época.

Si profunditzem en l'ambientació del mapa, es pot veure que el joc es situa en una masmorra subterrània on cada nivell és un pis, i cada vegada que en passem un estem baixant de pis. Això es pot veure en els obstacles, els quals consten de forats al terra i calaveres, en les parets i el terra els quals mostren un cert desgast, i en les portes les quals són pals de ferro.

Més enllà del que ja he parlat no es pot dir res més sobre la narrativa del videojoc a causa de la carència d'una història que lligui la jugabilitat amb el món del joc.

5.4 Interfícies

Primer de tot, cal dir que aquest prototip careix de qualsevol element sonor. Això és a causa a que no he cregut essencial la utilització d'aquests, a part, preferia utilitzar el temps dedicat en el projecte en altres parts del treball.

No obstant, els textos que surten en les interfícies visuals estan escrites amb una tipografia anomenada *Joystix monospace*, la qual he obtingut d'un tercer i manté l'estètica pixel art del videojoc.

Per una altra banda, classificaré les interfícies visuals en dos tipus: els menús i els elements que donen informació en la partida.

5.4.1 Menús

Els menús en aquest videojoc són senzills, però compleixen la seva funció. Durant una sessió de joc el jugador se'n pot trobar varis, els quals són el d'inici, el de pausa, el de victòria o derrota i el de selecció d'habilitats.

Els botons dels menús tenen tres estats. L'estat sense seleccionar, el qual es pot veure en la Figura 5.4.1.1. L'estat on el ratolí hi és a sobre, on es pinten les lletres d'un color més clar que el següent estat, hi ha dos variants, la del botó per sortir del joc el qual és de color taronja, i la dels demés botons amb un color blau clar, com es opt veure a la Figura 5.4.1.2. I per últim, l'estat en que el botó ha estat clicat, aquest cas és similar a l'anterior però amb colors més foscos, tal i com es veu a la Figura 5.4.1.3.

Figura 5.4.1.1: Botó sense seleccionar

Figura 5.4.1.2: Botons per jugar i sortir de la pantalla amb el ratolí a sobre

Figura 5.4.1.3: Botons per jugar i sortir de la pantalla seleccionats

El menú d'inici, que es pot veure en la Figura 5.4.1.1, permet al jugador triar entre iniciar una partida en la que se li dona un personatge principal aleatori i uns aliats que també ho són. En aquest prototip, al només haver-hi 4 personatges, aquests mai varien, però en un futur si que ho farien. També pot sortir del joc. A més, s'hi veu el títol del joc.

Figura 5.4.1.1: Menú d'inici

El menú de pausa, el qual es pot visualitzar en la Figura 5.4.1.2, apareix si enmig de la partida el jugador prem la tecla ESC. Aquest pausa el joc i li dona al jugador les opcions de sortir del menú i treure la pausa del joc, anar al menú d'inici o sortir del videojoc.

Figura 5.4.1.2: Menú de pausa

El menú de victòria o derrota, visible en la Figura 5.4.1.3, es mostra quan el jugador principal mor, surt el menú de derrota, o quan el jugador derrota a l'enemic final, surt el de victòria. La diferència entre aquests dos és un text que marca que ha passat, si s'ha guanyat o perdut. Les opcions que aquest menú dona al jugador són les mateixes que el menú de pausa, a excepció de l'opció de tornar a la partida.

Figura 5.4.1.3: Menú dederrota

Per últim, el menú de selecció d'habilitats, que es pot veure en la Figura 5.4.1.4, es mostra quan el personatge principal interactua amb les escales que permeten passar de nivell. Aquest menú ensenya el nom de tots els aliats que hi ha en el moment d'haver-se obert. Llavors, el jugador selecciona el que vulgui, i seguidament es mostren les habilitats de l'aliat seleccionat. A continuació, el jugador ha de triar quina habilitat vol que el personatge aconsegueixi, i quan ho fa, l'aliat prèviament seleccionat desapareix i es passa al següent pis.

Figura 5.4.1.4: Menú de selecció d'habilitats

5.4.2 Elements Visuals que donen Informació

En aquest subapartat es parla dels elements visuals utilitzats en el projecte per a transmetre certa informació al jugador. Aquesta informació bàsicament és la vida dels personatges, la disposició del mapa, i quan un personatge o enemic rep dany.

La vida que tenen els personatges inicialment sempre és igual a cinc, i a mesura que lluiten disminueix. Aquesta es mostra a la part superior esquerra de la pantalla, com es veu a la Figura 5.4.2.1, i ho fa ensenyant una icona del personatge juntament amb la quantitat de cors corresponents a la vida. El primer de tots sempre és el jugador principal. Quan un dels aliats desapareix quan es passa de pis, la seva icona juntament amb la seva vida desapareixen.

Figura 5.4.2.1: Interfície on es mostra la vida

D’altra banda, la disposició del mapa del nivell en el que el jugador es troba es mostra a través d’un minimapa el qual el jugador ha d’anar descobrint. Aquest es troba a la cantonada superior de la dreta i es veu com a la Figura 5.4.2.2. Quan el jugador entra a una sala en la que no hi havia estat, aquesta es mostra en el minimapa. A més, les habitacions adjacents connectades amb aquesta es mostren de forma opacada, de tal manera que el jugador sàpiga que allà hi ha sales si decideix anar per un altre camí. En afegit, la sala on es troba el final del pis està marcada amb un punt vermell.

Figura 5.4.2.2: Minimapa on es veu la sala del final gràcies a la habilitat “Visió”

Per acabar, quan un personatge o enemic rep dany, es pot veure a través de la seva imatge, ja que aquesta es veu afectada per un color en concret. A continuació explicaré cada tipus de dany i com es mostra.

El primer és el dany normal que pot fer un enemic o un personatge sense cap habilitat ofensiva. Aquest es mostra a partir d’un parpelleig de color vermell.

Després tenim el dany causat per la habilitat de cremar, la qual és permanent en l’enemic i causa que aquest es torni d’un color taronjós el qual va fluint en opacitat.
Per últim, hi ha la habilitat de paralitzar, la qual no fa dany sinó que immobilitza a l’enemic durant dos segons. En aquest període de temps l’enemic es pinta de color gris fins que es pot tornar a moure.

Tots els efectes es poden veure en la Figura 5.4.2.3, on el primer és l’efecte de dany, el segon és l’efecte de cremar i l’últim és l’efecte de paralitzar.

A més, l’efecte de reviure a un aliat, que es pot veure en la Figura 5.4.2.4, es mostra com l’efecte de rebre dany però amb el color verd.

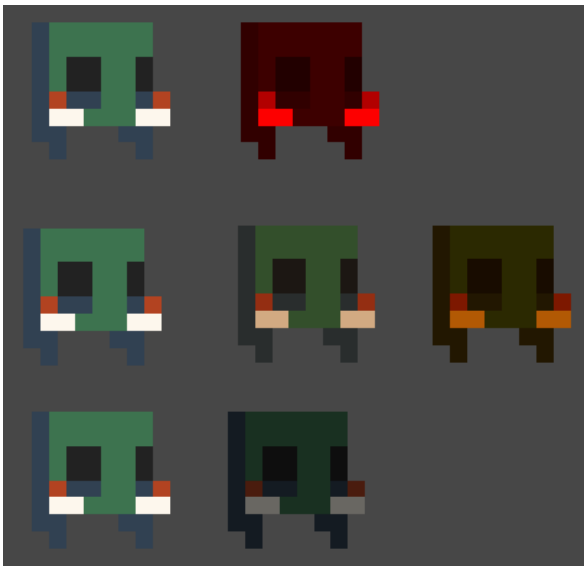

Figura 5.4.2.3: Efectes de dany, de cremar i de paràlisis en el charger

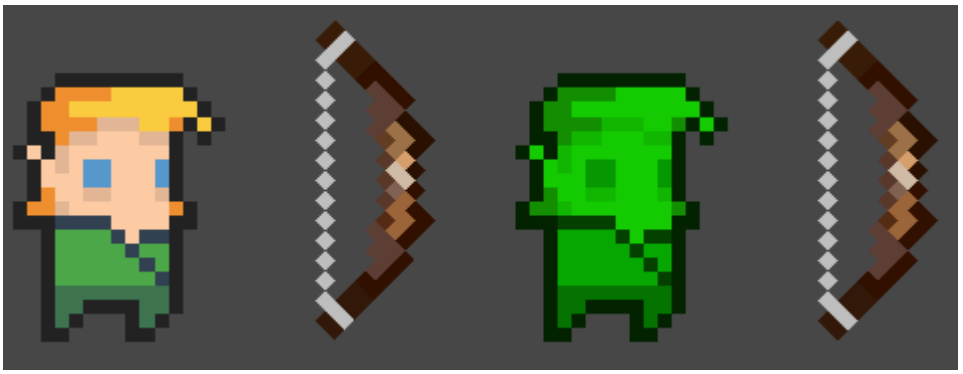

Figura 5.4.2.4: Efectes de reviure en l'arquer

5.5 Game Layout Chart

En aquest apartat mostraré amb diagrames el desenvolupament d'una partida a Pawn Sacrifice.

En la Figura 5.5.1 es pot veure el diagrama que es dóna en el menú inicial.

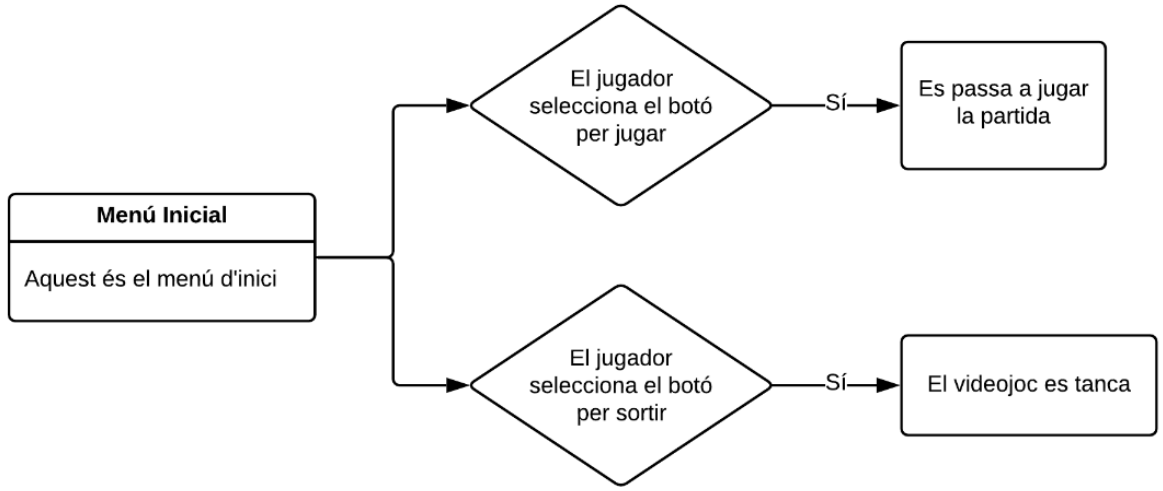

Figura 5.5.1: Diagrama del menú inicial

A la Figura 5.5.2, es veu el diagrama que es produeix una vegada el jugador ha iniciat la partida i ja està dintre la masmorra.

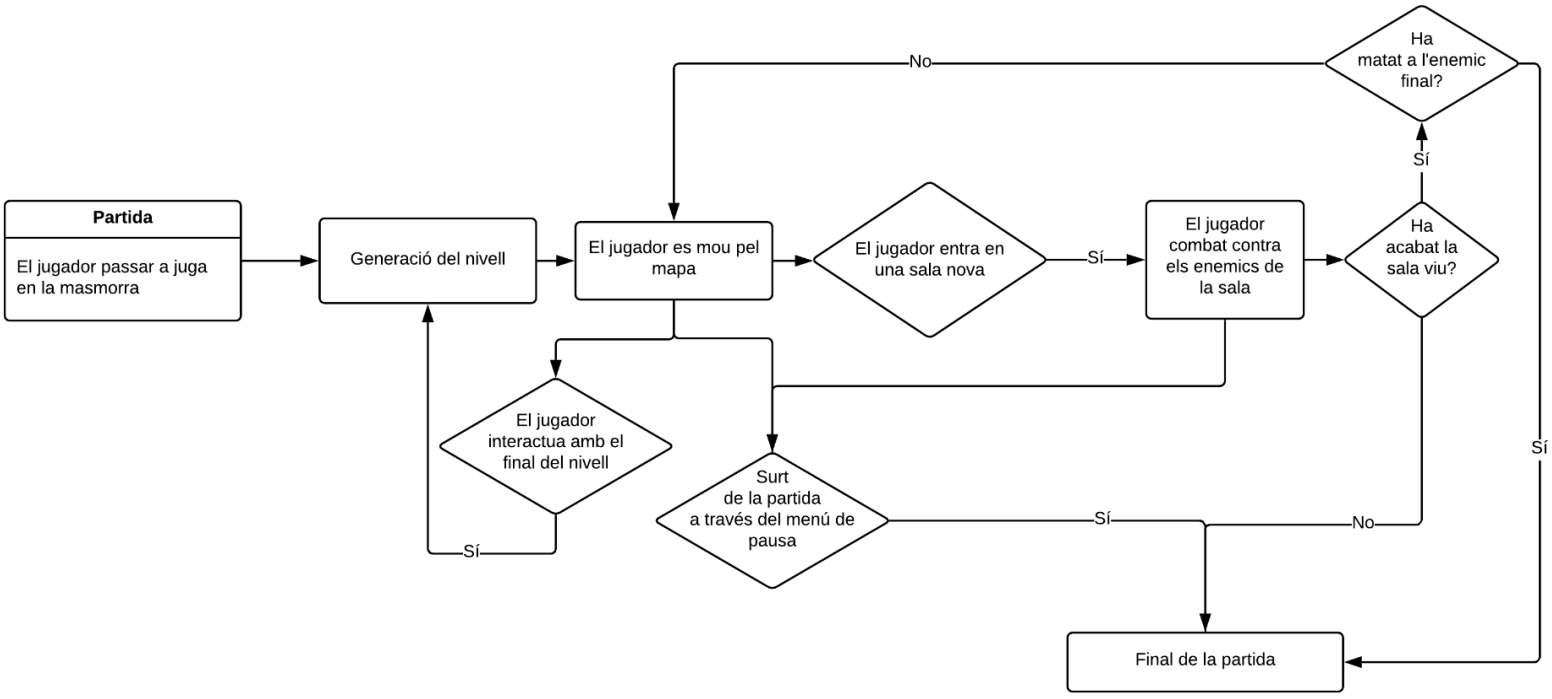

Figura 5.5.2: Diagrama de la partida en la masmorra

Per acabar, tenim la Figura 5.5.3 on es mostra el diagrama que es dóna en cada sala on hi hagin enemics.

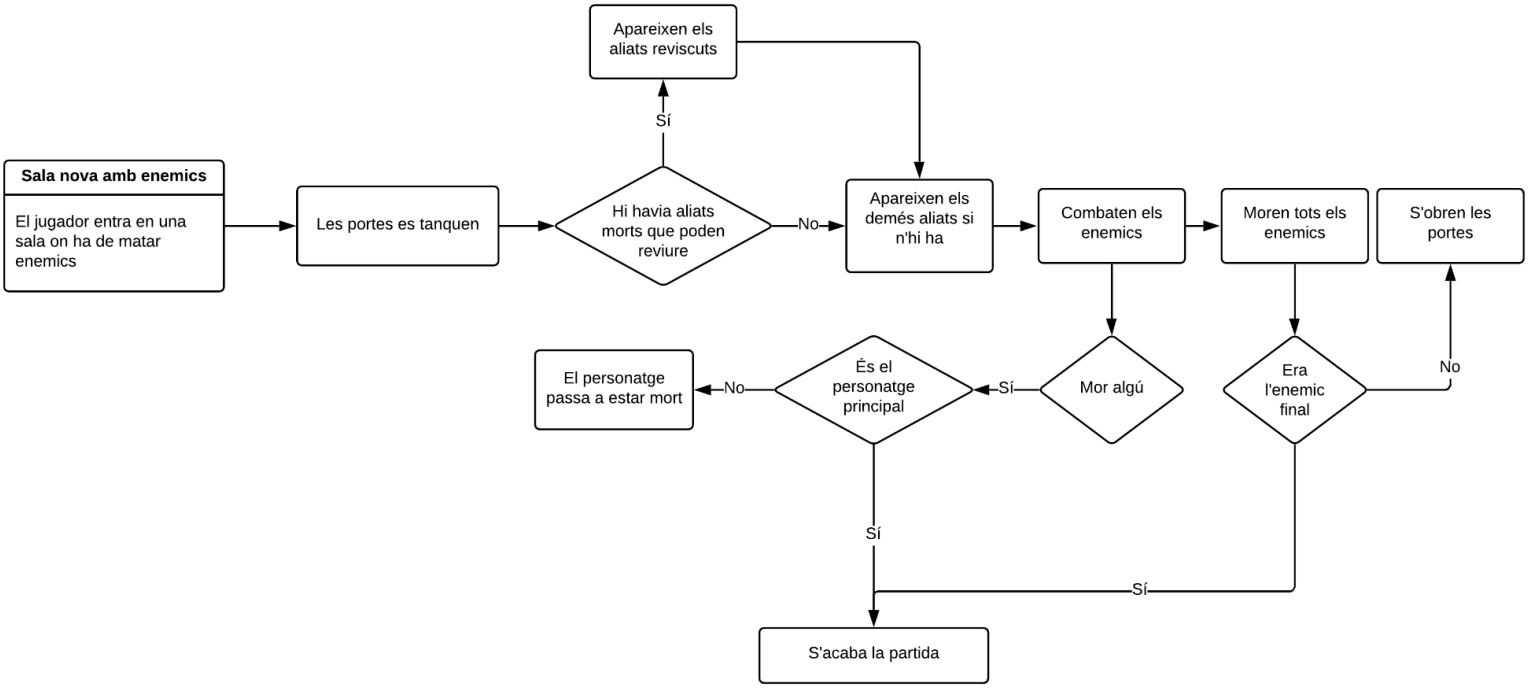

Figura 5.5.3: Diagrama d'una sala amb enemics

5.6 Disseny de Nivells i Creació d'Ambients

Aquest apartat el qual parla del disseny de nivells ja ha estat gairebé explicat en la seva totalitat en l'Apartat 5.1.1. Tot i això, aquí explicaré les etapes per les que ha passat el disseny de nivells i de les sales d'aquest projecte.

Al principi només tenia dissenyat un algoritme que tenia en compte habitacions petites, com es pot veure a la Figura 5.6.1. No obstant, la generació de nivells no ha canviat ja que vaig haver de fer petits canvis per afegir les habitacions grans i que formés nivells, com es pot veure a la Figura 5.6.2. L'únic repte va ser encaixar les habitacions grans en el nivell.

Igual que amb la generació de nivells, les sales no han tingut grans canvis, a excepció de la última versió. Aquestes al principi eren cubs les quals es diferenciaven per la quantitat i posició de les portes, com es pot veure en la Figura 5.6.1. Després els hi vaig posar un terra per a que la intel·ligència artificial hi pogués moure's. Finalment, vaig col·locar en cada una enemics, obstacles i vaig canviar les imatges del terra, parets i portes. Les parets i el terra els vaig posar gràcies als tilesets de Unity3D, els quals podem veure en les Figures 5.6.3, 5.6.4 i 5.6.5.

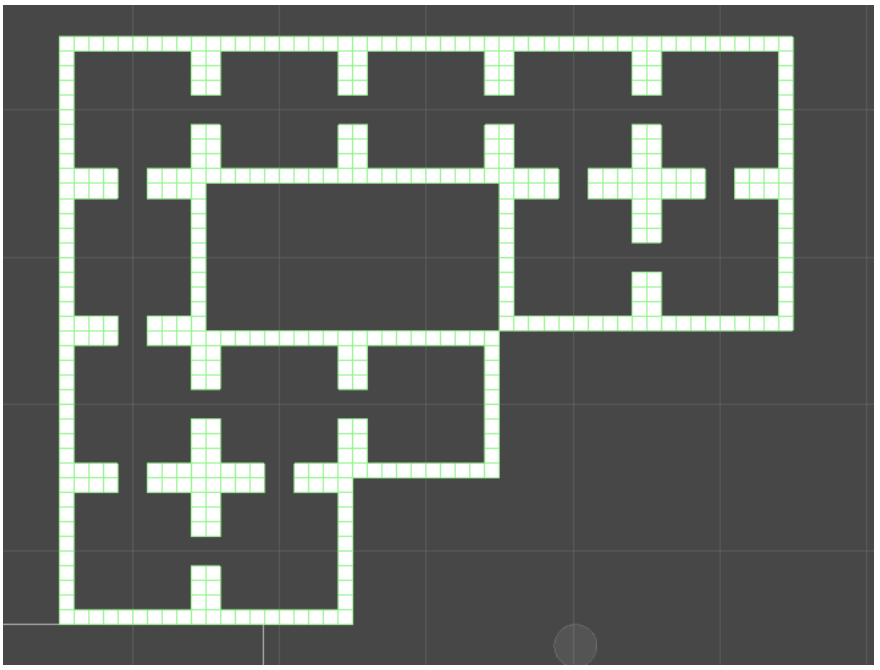

Figura 5.6.1: Primera versió de la generació de nivells

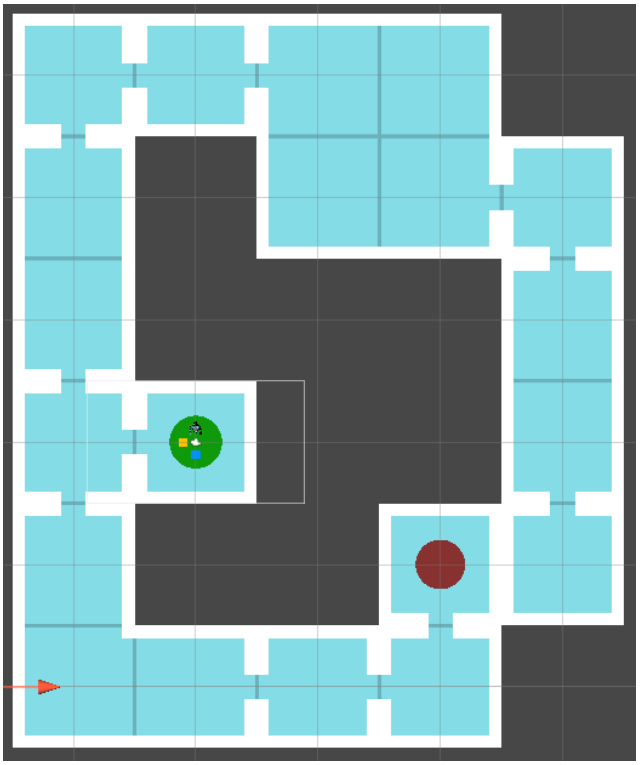

Figura 5.6.2: Segona versió de la generació de nivells

Figura 5.6.3: Tileset de les parets

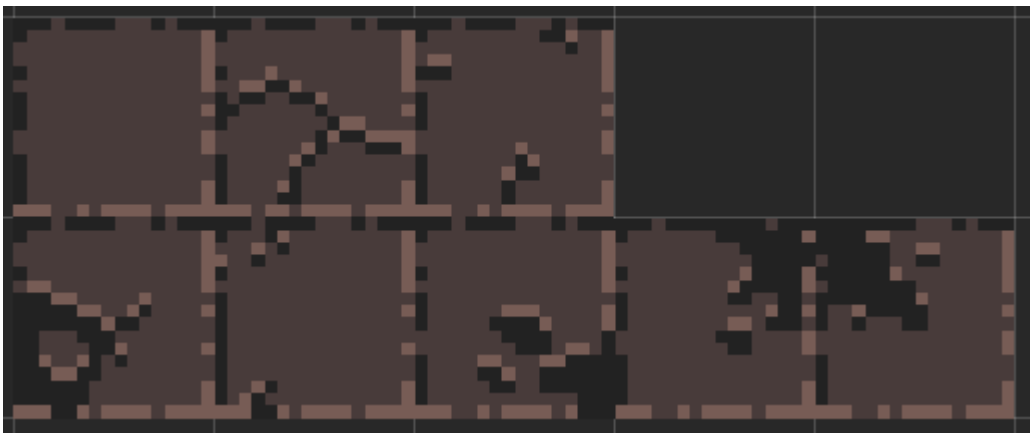

Figura 5.6.4: Tileset del terra

Figura 5.6.5: Tileset dels obstacles

5.7 Disseny d'Objectes

Com ja he comentat a l'Apartat 5.1.3, aquest prototip no disposa d'objectes. Això es a causa que aquests seran implementats en la versió completa del videojoc, i en aquest prototip em volia centrar en el nucli del videojoc que es basa en les habilitats dels personatges. Aquests objectes només canviarien estadístiques bàsiques dels personatges pel que no influencia en gran mesura la jugabilitat.

5.8 Llista d'Elements a Desenvolupar

Com ja he anat explicant en aquest document, els elements que no pertanyen a l'apartat de programació, són majoritàriament de tercers. Es pot veure el llistat d'elements que he creat personalment en l'Apartat 5.2.1. Entre els elements que he extret de tercers es troben elements d'interfícies, com poden ser els cors que mostren la vida dels personatges, els personatges i enemics, els elements de l'entorn i algunes de les armes que s'utilitzen en el joc. Per tant, es pot veure que la falta de *sketchs* i primeres versions d'aquests elements està recolzada a causa que la majoria d'elements a desenvolupar els he obtingut de fonts externes.

5.9 Disseny de la Càmera

La càmera principal en aquest projecte proporciona una vista axonomètrica del videojoc. Aquesta decisió la vaig prendre influenciat per videojocs del mateix gènere com The Binding of Isaac o UnderMine, ja que permet veure l'entorn i seguir els moviments del jugador des de dalt. A més, fa que en el joc no calgui afegir un efecte de gravetat.

Aquesta càmera té dos modalitats, una en la que es troba fixa, i l'altre en la que segueix al jugador. La primera, com es pot veure en la Figura 5.9.1, s'utilitza per les sales petites perquè es pot abarcar tota la sala dintre la càmera. En canvi, la segona modalitat s'usa en les sales grans, tal i com es veu en la Figura 5.9.2. Això es fa a causa que no tota la sala hi cap en la càmera. En aquesta la càmera segueix al jugador, però se li aplica un retràs que produeix que el personatge principal no es trobi en el centre de la càmera a no ser que es quedi immòbil. A causa d'això, se li dona més dinamisme al moviment del jugador en les sales grans.

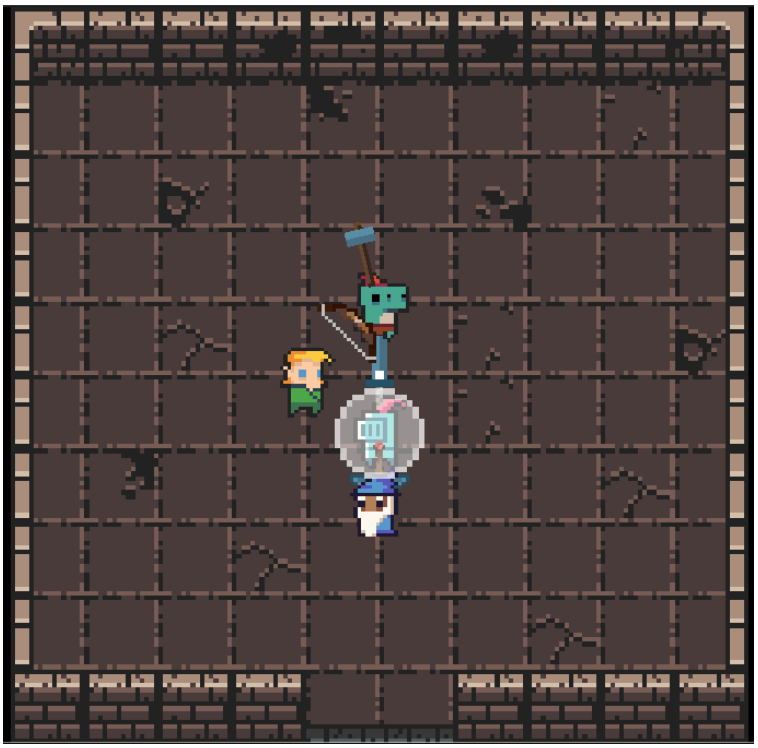

Figura 5.9.1: Càmera fixa

Figura 5.9.2: Càmera mòbil

6. Implementació i Proves

En aquest apartat explicaré els algorismes i processos més importants d'aquest projecte, amb parts del codi si és necessari, i explicant les proves que he fet per veure si funcionaven correctament.

6.1 Organització del Projecte

Primer de tot, mostraré la organització en les diferents escenes de Unity3D, quines són i què fan. La primera d'elles és la escena anomenada "Menu". Aquesta escena conté el menú inicial del videojoc i és amb el primer que es troba el jugador. Per tant, el que conté només és una interfície d'usuari la qual ja he explicat. Del funcionament del menú no mostraré res, ja que simplement trasllada al jugador a l'escena de joc o el treu del joc.

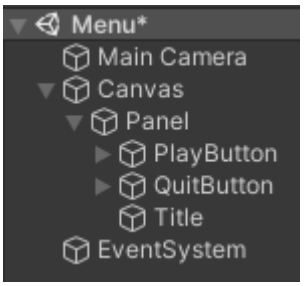

Figura 6.1.1: Organització de la escena "Menu"

Com es pot veure a la Figura 6.1.1, l'escena està composta de la càmera (Main Camera), d'un canvas on es mostren les interfícies visuals, i els botons que es troben dintre d'aquest canvas que són el de jugar (PlayButton) i el de sortir del joc (QuitButton), més un text (Title).

La segona escena s'anomena "GameScene", és la principal escena del projecte, a causa que hi és on tota la part jugable del videojoc es duu a terme. Per això, conté una major quantitat d'elements que l'anterior escena. A més, molts d'ells es creen de forma dinàmica, perquè el videojoc es basa en un sistema de generació de mapa procedural. D'aquesta escena sí que explicaré el funcionament dels diferents components en apartats posteriors.

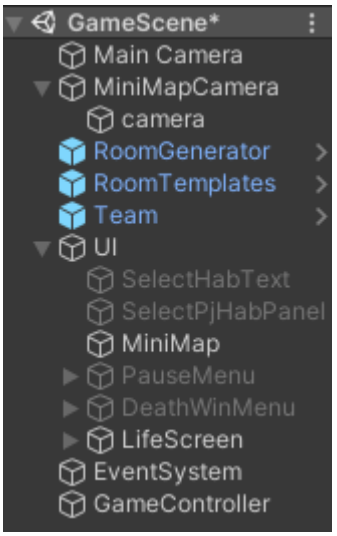

Figura 6.1.2: Organització de la escena “GameScene”

Com es veu a la Figura 6.1.2, l’escena té els següents elements:

- **Main Camera:** càmera principal del joc.
- **MiniMapCamera:** conté la càmera del minimapa, i és l’element que el va dibuixant.
- **Camera:** càmera secundària que enfoca al minimapa.
- **RoomGenerator:** element que crea i conté el mapa del nivell.
- **RoomTemplates:** objecte que serveix per emmagatzemar totes les habitacions que es poden utilitzar per a crear el mapa.
- **Team:** element que conté i gestiona a l’equip format pel personatge principal i els aliats.
- **UI:** canvas que conté totes les interfícies visuals de l’escena.
- **SelectHabText:** text que es mostra en el menú de selecció d’habilitats.
- **SelectPjHabPanel:** panell que conté els botons per a seleccionar el personatge i l’habilitat al passar de pis.
- **MiniMap:** interfície que mostra la càmera del minimapa en la càmera principal.
- **PauseMenu:** objecte que conté el menú de pausa.
- **DeathWinMenu:** element visual que emmagatzema el menú de victòria o derrota.
- **LifeScreen:** gestiona i guarda les interfícies visuals que mostren la vida dels personatges jugables.
- **EventSystem:** objecte necessari pel funcionament dels botons en els menús.
- **GameController:** gestiona la connexió entre diferents elements com *Team, SelectHabText* i *SelectPjHabPanel,* i el menú de pausa.

Per últim, hi ha una escena anomenada “Proves”, que no està dintre del videojoc, i només la he utilitzada per a fer proves sobre el funcionament de la majoria de les mecàniques. Per tant, ha anat variant al llarg del desenvolupament i no en mostraré res.

6.2 Implementació de la Generació Procedural de Nivells

La generació procedural de la masmorra va ser el primer que vaig fer d'aquest projecte. Tot i que hi ha hagut dos versions de la mateixa, una on només hi havia habitacions normals, i la última en la que també hi ha habitacions grans. Explicaré només la segona ja que és la que ha estat finalment implementada en el joc.

Per a implementar aquest algorisme he separat el mapa amb una matriu on cada cel·la ocupada és una habitació. Llavors, per a la creació de les habitacions he utilitzat un algorisme recursiu, el funcionament del qual és, primer de tot, crear l'habitació inicial en la casella on estava la habitació final de l'anterior nivell o, en el cas del primer nivell, en el centre de la matriu. A continuació, decideix de forma aleatòria quina porta ha d'estar oberta. Quan ja ho ha decidit, se'n va a la casella contigua a l'habitació inicial en la direcció de la porta oberta. Per exemple, si l'habitació inicial està en la casella (2,2) i s'ha obert la porta de la dreta, l'algorisme anirà a la casella (3,2). A partir d'aquí, va generant habitacions fins que totes les branques arriben a l'estat final de la recursivitat o fins que s'ha arribat al màxim d'habitacions, tal i com es pot veure en la Figura 6.2.1.

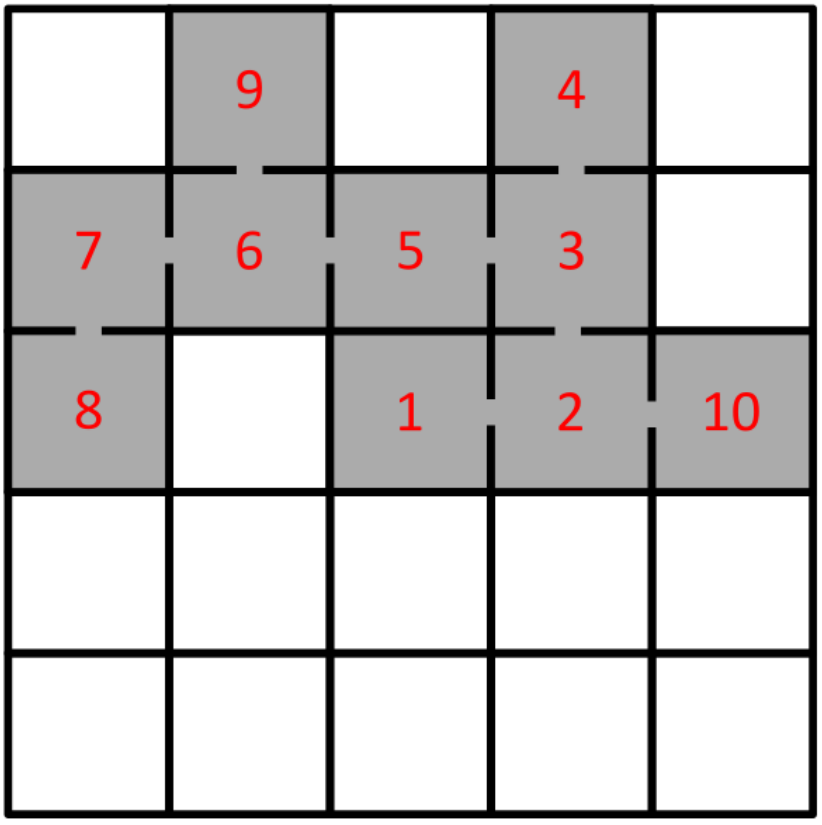

Figura 6.2.1: Matriu que exemplifica l'ordre en el que es creen les sales

Al crear-se una habitació nova, la qual no pot estar fora de la matriu, ni generar-se en la mateixa casella on ja hi ha una habitació, de forma aleatòria decideix si serà una habitació “cul de sac” (no genera més habitacions), una habitació gran o una habitació normal. En els tres casos, l'habitació que s’ha generat obre la porta que la connecta a l’habitació que la ha generat. D’aquesta manera sempre hi ha un camí d’inici a fi de la masmorra. Ara bé, en una habitació normal, s’intenta crear una habitació adjacent en cada direcció, la qual ho farà si compleix el ja esmentat, i si la genera, obre una porta en la seva direcció. En canvi, en les habitacions grans, la cosa canvia bastant. En aquestes es mira quina porta ha d’estar oberta per així saber quines de les habitacions grans es poden generar, a causa que cada una d’elles té 2 portes i les versions d’habitacions grans no són tan nombroses com en les normals. Al saber-ho, comprova quina de les possibilitats hi cap en la matriu, ja que, per exemple, una habitació gran, que és la suma de dos habitacions normals de forma horitzontal, necessita l’espai actual de la matriu i dos espais més cap a un costat, a causa que també ha de generar per força una habitació per l'altra porta. Una vegada fet això, continua amb la generació d’habitacions. En la Figura 6.2.2, es poden veure els diferents tipus d’habitacions grans.

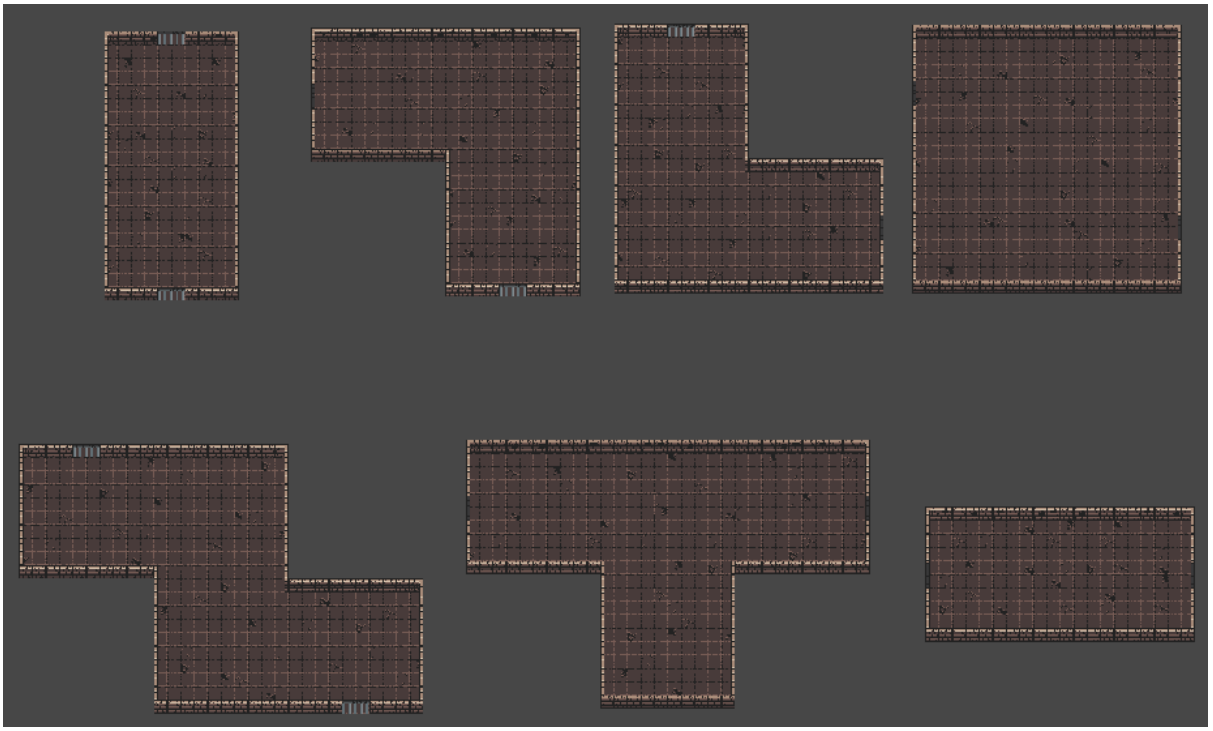

Figura 6.2.2: Tipus d’habitacions grans

Un cop generada la forma que tindrà la masmorra, es torna a repassar tota la matriu per ara obrir portes de forma aleatòria on abans no n’hi havia. Això, es fa per a que la masmorra estigui més connectada entre si i no es vegin tants camins allargats sense cap possibilitat de girar cap a una altra direcció. No obstant, hi ha limitacions amb quines habitacions es pot connectar una habitació normal. Aquesta, no pot fer-ho amb habitacions “cul de sac”,

habitacions grans i l'habitació inicial. Ara ja es pot escollir una habitació "cul de sac" per a ser la habitació final del nivell.
Ara sí, ja tenim la forma de la masmorra feta. D'altra banda, fins ara s'estava treballant amb arxius que guardaven la informació de les habitacions que es volen crear. El que es fa ara és buscar en el "RoomTemplates" el tipus d'habitació que concorda amb la informació de l'arxiu, i dintre de cada tipus d'habitació hi ha diverses habitacions de les quals se'n tria una per a ser generada. D'altra banda, la informació que es guarda en els arxius dels que he parlat és la següent: portes obertes i tancades, i el tipus d'habitació.

A més, si les habitacions generades no arriben al mínim establert o no es compleix alguna condició com la del número mínim d'habitacions "cul de sac", la generació del nivell es repeteix fins a que es compleixin tots els paràmetres. A continuació, en les Figures 6.2.1, 6.2.2 i 6.2.3, mostraré la part de codi més important d'aquest algorisme, la qual és on es genera la forma general del mapa.

```
private void generation(int x, int y, RoomDoors preRoom, int[] prevPos)
{
    //Pre: valid position of the grid
    //Post: room in the grid, and neighbours generation

    int[] pos = { x, y };

    preRoom.getPositionSpecial(pos, false, false, false, size);
    preRoom.openDoor(prevPos);
    miniMap.getNormalRoom(preRoom, pos);
    generatedRooms += 1;

    if(generate(x - 1, y, pos))
    {
        int[] aux = { x - 1, y };
        preRoom.openDoor(aux);
    }
    if (generate(x + 1, y, pos))
    {
        int[] aux = { x + 1, y };
        preRoom.openDoor(aux);
    }
    if (generate(x, y - 1, pos))
    {
        int[] aux = { x, y - 1 };
        preRoom.openDoor(aux);
    }
    if (generate(x, y + 1, pos))
    {
        int[] aux = { x, y + 1 };
        preRoom.openDoor(aux);
    }
}
```

Figura 6.2.3: Funció "generation()" de la generació de nivells

En la Figura 6.2.3, es pot veure la funció que juntament amb la funció “generate()” formen una estructura recursiva. Aquesta demana la posició x i y de la matriu, l’arxiu amb la informació de l’habitació i la posició de l’habitació de la que prové. A continuació, li dona la informació a l’arxiu amb la funció “getPosicionSpecial()” la qual demana la posició, si és l’inici, si és un “cul de sac” o si és una habitació gran, i la mida. Obre la porta amb l’habitació que la ha generat. La funció “getNormalRoom()” serveix per a fer saber al minimapa que aquesta habitació existeix. Finalment, s’intenta generar les habitacions en les diferents direccions amb la funció “generate()”, i si ho fa, obre una porta.

```
private bool generate(int x, int y, int[] prevPos)
{
    //Pre: position x and y of the grid
    //Post: generates a room in the grid and returns if a room is generated

    int[] pos = { x, y };

    if (generatedRooms == 0)
    {//generates the initial room

        preRoomGrid[x, y].getPositionSpecial(pos, true, false, false, size);
        miniMap.getNormalRoom(preRoomGrid[x, y], pos);
        generatedRooms += 1;

        int randomNumber = Random.Range(0, 4);

        switch (randomNumber)//only one door for the starting room
        {
            case 0:
                if(generate(x - 1, y, pos))
                {
                    int[] aux = { x - 1, y };
                    preRoomGrid[x, y].openDoor(aux);
                }
                break;
            case 1:
                if (generate(x + 1, y, pos))
                {
                    int[] aux = { x + 1, y };
                    preRoomGrid[x, y].openDoor(aux);
                }
                break;
            case 2:
                if (generate(x, y - 1, pos))
                {
                    int[] aux = { x, y - 1 };
                    preRoomGrid[x, y].openDoor(aux);
                }
                break;
            case 3:
                if (generate(x, y + 1, pos))
                {
                    int[] aux = { x, y + 1 };
                    preRoomGrid[x, y].openDoor(aux);
                }
                break;
        }

        return true;
    }
```

Figura 6.2.4: Funció “generate()” de la generació de nivells

La funció “generate()” és l’altra part de la recursivitat en l'algorisme. Com es pot veure en la Figura 6.2.4, aquest funció demana la posició x i y de la casella en la matriu, i la posició de l’habitació que la intenta generar. A més, es pot veure el codi per a generar l’habitació inicial. La primera part del codi és la mateixa que el ja explicat abans, amb l’excepció que aquí no obre cap porta ja que és la primera habitació i no n’hi ha cap que la precedeixi. A continuació, escull un número aleatori el qual decideix en quina direcció s’expandirà el mapa. Al fer-ho, la funció es crida a ella mateixa i, si tot funciona correctament, al final obre la porta.

```
else if(generatedRooms < maxRooms)
{
    if (!isInside(x, y)) { return false; } // off limits
    else if (preRoomGrid[x, y].used) { return false; } //already there's a room
    else
    {
        int random = Random.Range(0, 4);
        if (random != 0) //if 0 don't generate room
        {
            random = Random.Range(0, 5);
            if ((random == 0) && generatedRooms < maxRooms - 1)//try to generate big room, needs 2 rooms
            {
                return generateBigRoom(pos, prevPos);
            }
            else if (random == 1)//if 1 generate dead end room
            {
                preRoomGrid[x, y].getPositionSpecial(pos, false, true, false, size);
                preRoomGrid[x, y].openDoor(prevPos);
                miniMap.getNormalRoom(preRoomGrid[x, y], pos);
                generatedRooms += 1;
                if (preRoomGrid[x, y].deadEnd) { possibleEndRooms.Add(preRoomGrid[x, y]); }

                return true;
            }
            else
            {
                generation(x, y, preRoomGrid[x, y], prevPos);
                return true;
            }
        }
        else { return false; }
    }
}
else { return false; }
```

Figura 6.2.5: Funció “generate()” de la generació de nivells

En la Figura 6.2.5, es pot veure la segona part de la funció “generate()”. En aquesta, la funció intenta generar habitacions que no siguin la inicial. Primer de tot, mira si s’ha arribat al màxim d’habitacions, si és així retorna fals sense haver generat cap habitació. En canvi, si encara se’n poden fer, mira si la casella que li han passat està dintre de la matriu amb la funció “isInside()”, i si la casella ja està sent utilitzada per una altra habitació. Si passa aquestes condicions, genera un número aleatori el qual determina si es genera una habitació o no. A continuació, genera un altre número aleatori per saber quin tipus d’habitació generar, en el cas de que surti una habitació gran, ha d’assegurar-se que la habitació gran

més la habitació que generarà si o si després estan dintre dels límits. Un cop vist això, la intentarà generar amb la funció "generateBigRoom()". També pot ser que generi una habitació "cul de sac", on fa el procediment que hauria de fer a la funció "generation()" però sense expandir-se, a més, la afegeix a una llista d'habitacions "cul de sac" per a després poder triar una habitació final. Per últim, si genera una habitació normal, simplement executa la funció "generation()".

Quan més problemes vaig tenir amb aquest algorisme va ser l'implementar les habitacions grans, ja que aquestes no es col·locaven correctament en el mapa. La solució va ser emmagatzemar en un script, anomenat "BigRoomCapacity", les unitats que s'havia de desplaçar envers a la porta per on s'havia generat. Per una altra banda, les proves per a veure el funcionament d'aquest algorisme eren executar el joc per a revisar la forma del mapa i si estava ben generat, tal i com es pot veure, com a exemple, en la Figura 6.5.6.

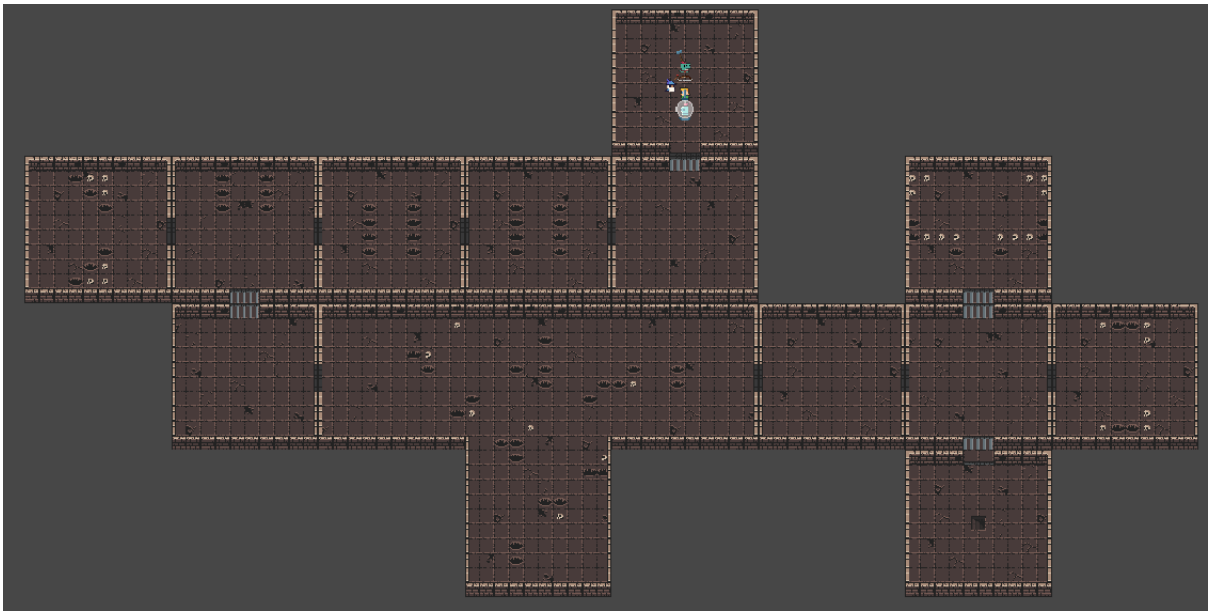

Figura 6.2.6: Exemple de masmorra generada proceduralment

6.3 Implementació de les Habitacions

Les habitacions, com he explicat abans, es troben en grups formats per les habitacions que tenen les portes en les mateixes direccions, per exemple, un grup amb habitacions amb una porta a l'esquerra. En canvi, les habitacions grans estan agrupades per tipus. Per exemple, un grup d'habitacions en forma de T. Per tant, aquests grups estan emmagatzemats cada un en un objecte, i aquest objecte està guardat a "RoomTemplates" de la "GameScene". Això ho he fet així per a poder seleccionar una habitació adient per a cada cas en la generació de nivells.

La organització d'una habitació és la que es veu en la Figura 6.3.1. En ella es pot veure l'element que conté els demés, el qual en aquest cas s'anomena "DL1", i que gestiona el que ha de fer l'habitació. A més, tenim el grid usat per a col·locar els objectes de l'escenari com parets, obstacles i terra. El "Tilemap_doors" hi és per a que el "NavMesh", element per a crear el mapa per on poden o no passar els personatges i enemics, no detecti que es pot passar per la porta. L'objecte "Enemies" gestiona els enemics de la sala i serveix com a pont de comunicació entre els enemics i altres elements. Per exemple, per comunicar a la sala que tots els enemics han mort i, per tant, les portes es poden obrir. Per altra banda, l'objecte "Black" és simplement una imatge de la forma de l'habitació, de color negre, perquè l'habitació no es vegi mentre el jugador està en un altra. Per acabar, tenim "Doors", la utilitat del qual és simplement per a que altres elements es puguin comunicar amb totes les portes simultàneament.

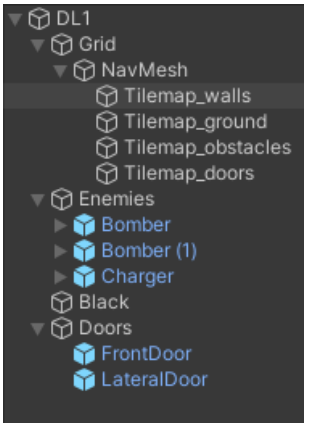

Figura 6.3.1: Organització de l'habitació anomenada DL1

El navmesh, és una eina de Unity3D la qual permet que objectes amb intel·ligència artificial, i el component NavMeshAgent, part de la eina navmesh, es puguin desplaçar per una zona creada amb anterioritat usant el navmesh. Aquesta eina transmet a l'objecte el camí òptim per a arribar al seu objectiu. A més, és una de les parts d'aquest projecte amb la que més problemes he tingut. Primer de tot, això és a causa que és el primer que tocava aquesta eina en Unity3D, i té algunes maneres de fer les coses que no són gaire intuïtives. Per exemple, a l'hora de crear un navmesh per a una habitació, el que s'ha de fer és rotar a -90º l'eix de les X de l'element que tingui el navmesh, i seguidament rotar els fills, objectes que determinen la forma de la navmesh, 90º en l'eix de les X. A més, hi ha habitacions, com la que es pot veure en la Figura 6.3.2 en forma de T, que generava un espai transitable a fora de l'habitació, el que causaba problemes amb les altres navmesh. Això es va solucionar a partir de canviar una opció que, per defecte, posés els espais en "no transitables". D'altra banda, al mirar la documentació i a l'anar fent proves en l'escena de proves, vaig poder tirar endavant.

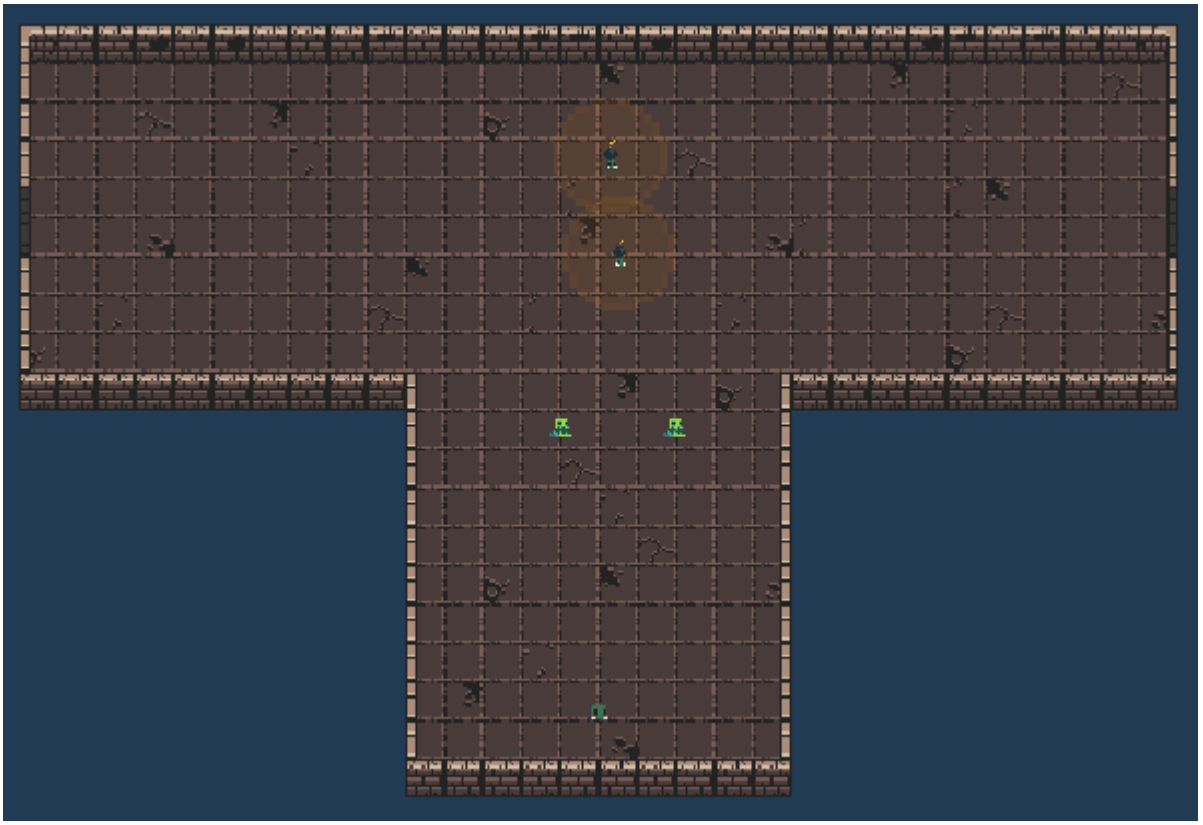

Figura 6.3.2: Habitació gran en forma de T

En aquest cas, no han sorgit problemes greus que hagi de comentar, i per fer les proves, la única manera que tenia per comprovar que estiguessin bé era jugar al videojoc i mirar si la quantitat d'enemics en l'habitació era adient, entre altres coses.

6.3.1 Implementació de les Portes

Les portes, a part de separar les sales les unes de les altres, són sensors que permeten saber on està el jugador principal. A través d'elles, l'habitació sap si té el jugador a dintre o no, a més, la càmera pot saber on enfocar.

D'altra banda, les portes estan formades per una imatge amb una animació que s'executa al obrir-se i tancar-se i un collider el qual és un trigger quan el jugador no està o ja ha completat l'habitació de la qual la porta en forma part. A més, conté dos scripts, un anomenat "InRoomBehaviour" i l'altre "DoorsInteraction".

El primer script, com es pot veure a la Figura 6.3.1.1, serveix per a les animacions de la porta i per a comunicar als enemics de l'habitació que el jugador hi ha entrat.

```
public void OpenDoor()
{
    animator.SetBool("opening", true);
    colid.isTrigger = true;
}

1 reference
public void CloseDoor()
{
    animator.SetBool("closing", true);
    colid.isTrigger = false;
}

0 references
public void Closed()
{
    animator.SetBool("closing", false);
    animator.SetBool("closed", true);
}

2 references
public void TeamInRoom(GameObject team)
{
    int numEnemies = 0;
    foreach (Transform child in enemies.transform)
    {
        numEnemies++;
        child.GetComponent<EnemyWaiting>().stopWaiting();
        if (child.GetComponent<EnemyMoveScript>()) { child.GetComponent<EnemyMoveScript>().getCharacters(team); }
        else if (child.GetComponent<ChargerMovementScript>()) { child.GetComponent<ChargerMovementScript>().getCharacters(team); }
    }
    if (numEnemies > 0)
    {
        foreach (Transform child in transform.parent)
        {
            child.GetComponent<InRoomBehaviour>().CloseDoor();
        }
    }

    room.getTeam(team);
    room.enterRoom();
}
```

Figura 6.3.1.1: Codi de l'script InRoomBehaviour

Les funcions que animen la porta no tenen una gran explicació, ja que és l'únic que fan. En canvi, la funció "TeamInRoom()", demana com a variable l'objecte "Team" que és el pare de tots els personatges jugables. En aquesta funció, primer es va a cada enemic de l'habitació se'l "desperta" i se li passa l'objecte "Team" per a que sàpiga a qui atacar. Llavors, es mira si hi ha algun enemic o si ja estaven morts quan el jugador ha entrat a l'habitació. Si hi ha enemics, li comunica a les altres portes que es tanquin. A més, li passa a l'habitació l'equip i executa la funció "enterRoom()" de l'script "RoomBehavior", el qual comunica a cada un dels aliats quins són els enemics de la sala, juntament amb fer passar el comptador en una unitat d'elements com el poder ser reviscut o els temps d'espera d'algunes habilitats com l'escut.
Per una altra banda, tenim el segon script, que s'encarrega de comunicar-se amb la càmera i de saber si el jugador ha canviat d'habitació, ja sigui entrant en ella o sortint.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        if (player == null) { player = collision.gameObject.GetComponent<PlayerMovement>(); }

        enterPosition.x = collision.gameObject.transform.position.x;
        enterPosition.y = collision.gameObject.transform.position.y;
    }
}
0 references
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        exitPosition.x = collision.gameObject.transform.position.x;
        exitPosition.y = collision.gameObject.transform.position.y;

        if (isVertical)
        {
            if (Mathf.Abs(exitPosition.x - enterPosition.x) > offset)//change of room
            {
                if (parent.leaveRoom()){ moveCamera(collision); inroomBehaviour.TeamInRoom(collision.transform.parent.gameObject); parent.enterRoom();}
            }
        }
        else
        {
            if (Mathf.Abs(exitPosition.y - enterPosition.y) > offset)//change of room
            {
                if (parent.leaveRoom()){ moveCamera(collision); inroomBehaviour.TeamInRoom(collision.transform.parent.gameObject); parent.enterRoom();}
            }
        }
    }
}
```

Figura 6.3.1.2: Codi de l'script DoorsInteraction

El que podem veure l'script en la Figura 6.3.1.2, són dues funcions que s'executen quan un objecte amb un Collider2D entra en el Collider2D de la porta, i quan surt. La funció que s'utilitza en la entrada del collider és per agafar informació. Per tant, s'aconsegueix al jugador, si és el que ha entrat per la porta, i el punt del collider en el que ho ha fet. Aquest punt d'entrada es guarda a causa que, després, també s'hi emmagatzema el punt per on surt l'objecte, per així, poder saber si el jugador ha sortit pel mateix lloc per on ha entrat, el que faria que no hagués sortit de l'habitació, o per si ha sortit per l'altre costat, sabent així que ha sortit de l'habitació.
Quan es surt de l'habitació, el que es fa és, primer de tot, executar la funció "leaveRoom()" de l'script "RoomDoors" que té l'habitació. El que fa és treure la capa negra o mostrar-la depenent de si el jugador està a l'habitació, a més, retorna si el jugador està a l'habitació o no. Si hi està, per tant, ha entrat a l'habitació, executa la funció "moveCamera()", la qual mou la càmera cap a l'habitació i, també executa la funció que s'ha vist abans "TeamInRoom()". Finalment, executa la funció "enterRoom()" del script "RoomDoors", que es comunica amb el minimapa per a que l'habitació es mostri en el minimapa.

Amb les portes, el problema que em vaig trobar era el d'identificar si el jugador està entrant en una habitació o sortint d'ella. Com ja he explicat, la solució ha estat guardar el punt d'entrada i el de sortida del personatge i amb això comprovar-ho a partir de la diferència entre les X o les Y, depenent de si la porta és vertical o horitzontal respectivament.

6.4 Implementació de la Intel·ligència Artificial

La intel·ligència artificial d'aquest projecte està implementada en els aliats i en els enemics. Cada un funciona de forma diferent, però tenen una cosa en comú i és el NavMeshAgent. Com ja he mencionat en l'Apartat 6.3, les sales tenen un NavMesh per on els NavMeshAgents, col·locats en els personatges i enemics, poden desplaçar-se. Això ho he tret d'una llibreria de Unity3D anomenada *NavMeshPlus*, i serveix per a que els objectes amb el NavMeshAgent que estiguin a sobre d'un NavMesh, puguin trobar el camí òptim cap l'objectiu.

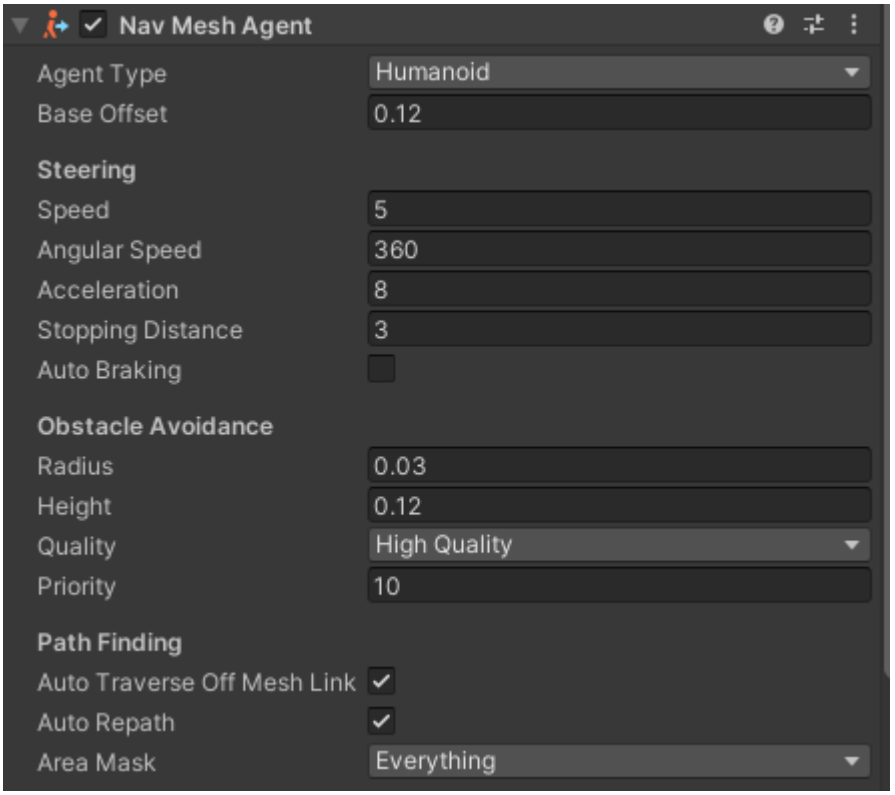

Figura 6.4.1: Component NavMeshAgent

Com es pot veure en la Figura 6.4.1, el NavMeshAgent demana diferents variables. Les més importants i les que més he hagut de remenar són les de l'apartat "Steering", on es demana la velocitat a la que anirà el personatge (Speed), la velocitat en la que gira (Angular Speed), l'acceleració que té (Acceleration), la distància a la que es para abans d'arribar l'objectiu (Stopping Distance) i si es vol que el personatge derrapi o no (Auto Braking). A més, si en aquest projecte hagués posat obstacles que es moguessin, l'apartat de "Obstacle Avoidance" m'hauria estat útil, però no és el cas.

6.4.1 Aliats

En aquest apartat explicaré com funciona la intel·ligència artificial dels aliats. No obstant, la idea general ja ha estat explicada l'Apartat 5.2.2, pel que obviaré aquest aspecte i em centraré en explicar la part de codi que mostri.

```
private void getTarget()
{
    //Pre: ---
    //Post: sets the target for te IA to follow/attack

    float minDistance = 99999.0f;

    if (target == null) { target = transform; }
    if (listEnemies == null || listEnemies.Count == 0) { target = player; targetIsPlayer = true; } //no en
    else //select nearest enemy
    {
        for (int i = 0; i < listEnemies.Count; i++)
        {
            if (listEnemies[i] != null)
            {
                float distance = Vector3.Distance(listEnemies[i].transform.position, transform.position);
                if (distance < minDistance)
                {
                    minDistance = distance;
                    target = listEnemies[i].transform;
                    targetIsPlayer = false;
                }
            }
        }
    }

    if (targetIsPlayer) { agent.stoppingDistance = 3; }
    else { agent.stoppingDistance = 0; }
}
```

Figura 6.4.1.1: Funció getTarget de l'script AgentScript

En la Figura 6.4.1.1, es pot veure la funció "getTarget()" del script "AgentScript", el qual s'utilitza en els aliats pel moviment. Aquesta funció és usada en la funció "Update()" de l'script i s'executa quan l'objectiu és null o cada 0.5 segons. Ara bé, el que fa és, primer de tot, posar-se com a objectiu a ell mateix en cas de que aquest sigui null, per si la funció acabés sense un objectiu. Llavors, es mira si la llista d'enemics és null o si el nombre d'enemics és igual a zero. En aquest cas, significa que està en una sala sense enemics i ha de seguir al jugador. En canvi, si hi ha enemics, es passa per cada un d'ells mirant quin és el més proper per considerar-lo l'objectiu. Per acabar, si l'objectiu és el jugador se li diu al NavMeshAgent que hi mantingui una distància de tres unitats, si no ho és, no ha de mantenir cap distància.

```
private void setDestination()
{
    //Pre: ---
    //Post: sets the destination

    if (target != null)
    {
        Vector3 destiny = transform.position;

        if (!ranged) { destiny = target.position; }
        else
        {
            RaycastHit2D hit;
            hit = Physics2D.Raycast(transform.position, (target.position - transform.position), Mathf.Infinity, layerMask);

            if (hit.collider != null && !hit.transform.CompareTag("Enemy"))//raycasts to the target, if the path is clear to
            {
                destiny = target.position;//it will be moving for the next searchCooldown period
            }
        }
        if (Vector3.Distance(transform.position, target.position) <= 1 && !targetIsPlayer) //flees from the enemy
        {
            Vector3 dirToEnemy = transform.position - target.transform.position;
            Vector3 newPos = transform.position + dirToEnemy;
            newPos.z = transform.position.z;
            destiny = newPos;
        }
        agent.SetDestination(destiny);
    }
}
```

Figura 6.4.1.2: Funció setDestination de l'script AgentScript

En la Figura 6.4.1.2, es veu la funció "setDestination()" de l'script anomenat "AgentScript". Aquest s'executa després de la funció "getTarget()", ja explicada. En aquest cas, el que fa aquesta funció és dictaminar el que ha de fer el personatge amb l'objectiu que se li ha donat. Primer de tot, se li dóna com a destí la seva mateixa posició, a causa que, si el personatge ataca a distància i té l'objectiu a tir, aquest es quedi en la seva posició. D'altra banda, si el personatge ataca cos a cos, se li assigna com a destí la posició de l'objectiu. En canvi. Si el personatge ataca a distància, aquest llança un Raycast cap a la posició de l'objectiu, si el personatge té una línia d'atac directa, no es mou. Si no la té, es mou cap a l'objectiu. Per acabar, si un personatge està a menys d'una unitat de distància de l'objectiu, essent un enemic, aquest es mou en direcció contrària a l'objectiu.

```
private void AIAttack()
{
    //Pre: ---
    //Post: AI tries to attack
    if (!shooted && ableToAttack && !cooldown)
    {
        if (!agentScript.targetIsPlayer)//we don't attack the player
        {
            if (distanceAttack) { shoot(); shooted = true;}
            else
            {
                float distanceToTarget = Vector3.Distance(agentScript.giveTarget(), finalPos.position);
                if (distanceToTarget <= 2.0f) { meleeAttack();  ableToAttack = false; }
            }
        }
    }
}
```

Figura 6.4.1.3: Funció AIAttack de l'script WeaponAttack

En la Figura 6.4.1.3 es pot veure la funció "AIAttack()" de l'script "WeaponAttack", el qual es troba en totes les armes dels personatges jugables. Per tant, és el mateix script que el jugador utilitza per atacar. El que fa la funció és el següent, primer de tot, confirma que es pot atacar, gràcies a les variables booleanes que es veuen en el primer *if.* Llavors, s'assegura que no s'estigui atacant al jugador mirant la variable "targetIsPlayer" de l'script "AgentScript". A continuació, mira si ha d'atacar a distància o cos a cos. Si és a distància, executa la funció "shoot()", la qual instancia un projectil i li passa la direcció i les habilitats que el personatge té, com podria ser la penetració. En canvi, si l'atac és cos a cos, s'executa la funció "meleeAttack()", la qual activa el collider de l'arma, i a més, posa a fals la variable "ableTotAttack", el que permet produïr el moviment de l'arma en la funció "Update()" de l'script.

6.4.2 Enemics

Ja hem explicat el funcionament general de cada enemic en l'Apartat 5.2.3. Per tant, en aquest apartat explicaré el que cregui que necessita una explicació amb més profunditat juntament amb el codi corresponent. Primer de tot, cal afegir que tan el slime normal, el bomber i el boss comparteixen el mateix script de moviment anomenat "EnemyMovement", mentre que els slimes que s'intenten fusionar i el charger tenen scripts de moviment que hereten de l'script ja mencionat. En l'script de "EnemyMovement", també hi ha una funció "getTarget()", que té la mateixa funció que té en els personatges, només que aquesta busca el personatge o el minion més proper. A més, ha de tenir en compte si el personatge està mort al terra o no, ja que quan els persontages busquen enemics, els que estan morts simplement desapareixen. D'altra banda, té una funció anomenada "movement()", la qual serveix per a moure l'enemic. En aquesta funció és on s'utilitza la funció "getTarget()", en el cas que sigui null, hagin passat 2 segons, o el personatge hagi mort. S'ha de seguir mirant si

l'objectiu és null a causa del minion, ja que aquest sí que desapareix al morir. Un cop obtingut l'objectiu, se li assigna la destinació i es posa a la imatge de l'enemic mirant cap a ell, tal i com es pot veure en la Figura 6.4.2.1.

```
public virtual void movement(float time)
{
    if (characters.Count > 0)
    {
        timer += time;
        if (timer >= targetTimer || target == null) { getTarget(null); timer = 0.0f; }
        else if (target.gameObject.GetComponent<CharacterDeath>())
        {
            if (target.gameObject.GetComponent<CharacterDeath>().isDead)
            {
                getTarget(null);
            }
        }
        agent.SetDestination(target.position);

        lookDirection(target.position);
    }
}
```

Figura 6.4.2.1: Funció movement de l'script EnemyMovement

D'altra banda, la funció "getTarget()" del charger funciona igual que en l'script "EnemyMovement", tot i que, necessita una línia recta lliure d'atac, el qual ho mira amb un Raycast. Quan té un objectiu, posa en verdader les variables "selected" i "charging". En canvi, posa la variable "walking" a fals, tal i com es pot veure en la Figura 6.4.2.2. En canvi, si no disposa d'un objectiu, aconsegueix una posició vàlida aleatòria del seu voltant.

```
float distance = Vector3.Distance(transform.position, player.transform.position);
if (distance < minDistance)
{
    RaycastHit2D hit;
    hit = Physics2D.Raycast(transform.position, (player.transform.position - transform.position), Mathf.Infinity, layerMask);

    if (hit.collider != null && !hit.transform.CompareTag("Background") && !hit.transform.CompareTag("Enemy"))//raycasts direct
    {
        selected = true;
        charging = true;
        walking = false;
        selectedPosition = new Vector3(player.transform.position.x, player.transform.position.y, transform.position.z);
        minDistance = distance;
        target = player.transform;
    }
}
```

Figura 6.4.2.2: part de la funció getTarget de l'script ChargerMovementScript

Les variables que just he esmentat s'utilitzen en la funció "movement()" del mateix script. Com es pot veure en la Figura 6.4.2.3, en aquesta funció hi ha tres opcions: quan camina, que es duu a terme quan no ha aconseguit un objectiu i es desplaça a una posició aleatòria. Quan està carregant, que es dóna quan el charger té un objectiu i dóna una embestida cap a la posició corresponent. Finalment, arriba l'estat en que, tan si estava caminant com carregant, l'enemic ha arribat a l'objectiu i s'ha d'esperar 1.5 segons per a aconseguir un nou objectiu.

```
public override void movement(float time)
{
    if (characters.Count > 0)
    {
        if (walking)
        {
            agent.speed = walkingSpeed;
            agent.SetDestination(selectedPosition);
            if (Vector3.Distance(transform.position, selectedPosition) < 1f) { walking = false; }
        }
        else if (charging)
        {
            agent.speed = chargingSpeed;
            agent.SetDestination(selectedPosition);
            if (Vector3.Distance(transform.position, selectedPosition) < 1f) { charging = false; }
        }

        if (!walking && !charging)
        {
            waitTimer += time;
            if (waitTimer >= waitTime) { waitTimer = 0.0f; getTarget(null); }
        }

        lookDirection(selectedPosition);
    }
}
```

Figura 6.4.2.3: Funció movement de l'script ChargerMovementScript

Per una altra part, l'script "SlimeMovementScript" és utilitzat pels slimes que s'intenten fusionar entre ells. En aquest, la funció "movement()", que es pot veure en la Figura 6.4.2.4, funciona de la següent manera. Primer de tot, marca com a destí l'altre slime amb el que s'ha de fusionar, aconseguit en l'script "SlimeGetHit", a no ser que l'altre slime hagi estat destruit. A continuació, si hi ha una distància menor a 0.7 unitats, s'inicia la animació de fusió i, si l'altre slime no ho ha fet encara, aquest primer es proclama com al slime que portarà a terme la fusió. Aquesta fusió és dóna lloc en la funció "fusion()" del mateix script. En aquesta funció, basicament s'instancia un slime normal i es destrueixen els dos petits. D'altra banda, si un dels dos slimes que s'han de fusionar mor abans de que la fusió passi, el que queda viu habilita l'script "EnemyMovement" i desactiva l'actual script.

```
public override void movement(float time)
{

    if (!charAttack && target != null) { agent.SetDestination(target.position); }

    if (target != null && !animating)
    {
        lookDirection(target.position);

        fusionTimer += time;
        if (Vector3.Distance(transform.position, target.position) < distance && fusionTimer >= launchTime)
        {
            animator.SetBool("fusioning", true);
            animating = true;
            if (!target.GetComponent<SlimeMovementScript>().fusioner) { fusioner = true; }
        }
    }
    else if (target == null && !launching) //in case the other slime is destroyed, it tries to attack the c
    {
        animator.SetBool("fusioning", false);
        animating = false;
        charAttack = true;
        gameObject.GetComponent<EnemyMoveScript>().enabled = true;
        Destroy(this);
    }
}
```

Figura 6.4.2.4: Funció movement de l'script SlimeMovementScript

Un slime, a l'hora de dividir-se, executa la funció "Split()" de l'script "SlimeGetHit", la qual es pot veure en les Figure 6.4.2.5, 6.4.2.6 i 6.4.2.7. Primer de tot, comprova si es pot dividir, només no pot si és un slime petit. Llavors, desactiva el collider, aconsegueix el prefab del slime i comprova si no ha arribat al límit de possibles divisions, la qual és de dos. Si es pot dividir, es selecciona la direcció en la que seran llançats els slimes, la qual és la direcció contrària en la que ha estat colpejat per últim cop. Llavors, es fa una iteració tres cops, un per cada slime petit. Per a cada un, primer de tot, se l'instancia, se li desactiven les colisions i se li afegeix un RigidBody per a que pugui ser llançat amb una força en la funció "Update()". A continuació, se li passa la llista d'enemics als que pot atacar i la direcció en la que és llançat anomenada "endPosition". Se li dona una unitat menys de vida que el màxim de vida de l'actual slime, i es determina si és un dels slimes que s'intentarà fusionar. Si és el cas, se li passa el company. En aquests dos, s'executa la funció "Regenerator()", la qual els activa l'script "SlimeMovementScript".

```
private void Split()
{
    if (canSplit)
    {
        GetComponent<BoxCollider2D>().enabled = false;
        slimePrefab = Resources.Load<GameObject>("Prefabs/Enemies/Slime");
        if (slimePrefab.GetComponent<SlimeGetHit>().life-1*familyTree > 0)//limit of times a slime can fuse
        {
            Vector3 direction = new Vector3(transform.position.x - hitPosition.x, transform.position.y - hitPosition.y, transform.position.z).normalized;
            int rotation = 100;
            GameObject firstRegenerator = null;

            for (int i = 0; i < 3; i++)
            {
                GameObject miniSlime = Instantiate(slimePrefab, transform.position, transform.rotation, transform.parent);
                SlimeGetHit miniHit = miniSlime.GetComponent<SlimeGetHit>();
```

Figura 6.4.2.5: Primera part de la funció split de l'script SlimeGetHit

```
miniSlime.GetComponent<BoxCollider2D>().enabled = false;
miniHit.AddRigidBody();
miniSlime.GetComponent<EnemyMoveScript>().characters = GetComponent<EnemyMoveScript>().characters;

miniSlime.transform.localScale = new Vector3(miniSlime.transform.localScale.x/2, miniSlime.transform.localScale.y/2, 1.0f);

endPosition = new Vector3(Mathf.Cos(rotation)*direction.x - Mathf.Sin(rotation)*direction.y,
Mathf.Sin(rotation)*direction.x + Mathf.Cos(rotation)*direction.y, direction.z);

endPosition = new Vector3(endPosition.x*5, endPosition.y*5, endPosition.z);
miniHit.getEndPosition(endPosition);

miniHit.splited = true;
miniHit.canSplit = false;
miniHit.life -= 1*familyTree;
miniHit.familyTree = familyTree + 1;
```

Figura 6.4.2.6: Segona part de la funció split de l'script SlimeGetHit

```
                if (i == 0) { rotation += 70; }
                else if (i == 1) { rotation -= 140; }

                if (i != 0)
                {
                    if (firstRegenerator != null)
                    {
                        miniHit.Regenerator(firstRegenerator.transform);
                        firstRegenerator.GetComponent<SlimeGetHit>().Regenerator(miniSlime.transform);
                    }
                    else { firstRegenerator = miniSlime; } //gets the first slime who wants wants to ge
                }
            }
        }
    }
    Destroy(gameObject);
}
```

Figura 6.4.2.7: Tercera part de la funció split de l'script SlimeGetHit

Per acabar, l'últim enemic que comentaré és el boss (no explico res del bomber a causa que amb el que ja s'ha vist d'ell ja n'hi ha suficient). Primer de tot, el boss té un script anomenat "BossAttackBehaviour", el qual gestiona quan fer els atacs a distància, fent que, quan

aquests atacs es duen a terme, l'enemic es quedi immòbil i no pugui atacar cos a cos. Un cop fet l'atac a distància, l'script no permet que faci un atac cos a cos durant 0.5 segons, i tampoc permet fer un atac a distància durant 5 segons més o menys un número aleatori de -1.5 a 1.5. D'altra banda, l'atac cos a cos és el mateix que fa un personatge amb una arma de cos a cos, només que ara l'enemic la apunta cap al jugador. En canvi, l'atac a distància es duu a terme en l'script anomenat "BossShooting". En aquest script, la funció "Shoot()", que es pot veure en la Figura 6.4.2.8, agafa una direcció, la qual és un vector cap a la dreta, i també agafa un angle aleatori entre 0 i 45. Llavors, fa una iteració vuit vegades, en les que en cada una crea un vector de direcció sumant-li un angle a la direcció anterior. A continuació, executa la funció "shootProjectile()", que, com es pot veure en la Figura 6.4.2.9, instancia un projectil i li dóna la direcció de dispar. Finalment, suma 45 graus per a la direcció del següent dispar.

```
public void Shoot()
{
    //Pre: ---
    //Post: shoot fireballs

    Vector3 direction = new Vector3(transform.position.x+1, transform.position.y, transform.position.z) - transform.position;
    int randomAngle = Random.Range(0, 45);
    float rotationDegrees = randomAngle;
    float rotationRads = randomAngle*Mathf.Deg2Rad;

    for (int i = 0; i < 8; i++)
    {
        Vector3 newDirection = new Vector3(Mathf.Cos(rotationRads)*direction.x - Mathf.Sin(rotationRads)*direction.y,
        Mathf.Sin(rotationRads)*direction.x + Mathf.Cos(rotationRads)*direction.y, direction.z);

        shootProjectile(newDirection.normalized, rotationDegrees);
        rotationRads += Mathf.PI/4;
        rotationDegrees += 45;
    }
}
```

Figura 6.4.2.8: Funció Shoot de l'script BossShooting

```
private void shootProjectile(Vector3 direction, float rotation)
{
    //Pre: ---
    //Post: instanciates the projectile and shoots it

    GameObject element = Instantiate(projectile, transform.position, Quaternion.Euler(new Vector3(0, 0, rotation)));
    element.GetComponent<ProjectileMovement>().getShotDirection(direction);

}
```

Figura 6.4.2.9: Funció shootProjectile de l'script BossShooting

No vaig tenir problemes que pugui destacar sobre la intel·ligència artificial, més enllà d'averiguar el funcionament del navmesh i el NavMeshAgent. En canvi, per a fer proves amb

ella, vaig utilitzar l'escena de proves utilitzant un navmesh de proves per a veure si els enemics i aliats es comportaven com devien.

6.5 Personatge Controlat pel Jugador i la Suma d'Habilitats

En aquest apartat parlaré sobre el personatge principal, controlat pel jugador, i de la mecànica de sumar les habilitats d'altres personatges al personatge principal.

Abans de res, el moviment del personatge principal es dóna quan el jugador prem les tecles w a s d, per anar endavant, a l'esquerra, endarrere i a la dreta respectivament. Quan una d'aquestes tecles és presionada, s'aplica el moviment en l'eix de coordenades indicat sobre el RigidBody del personatge en la funció "FixedUpdate()" de l'script "PlayerMovement". Aquest moviment, com ja he comentat amb antelació, es veu reduït a la meitat quan el jugador ataca.
Per una altra banda, els atacs del jugador es fan quan aquest prem el botó esquerre del ratolí. Això ocurreix en la funció "PlayerAttack()" de l'script "WeaponAttack". A més, aquesta funció s'executa en la funció "Update()" del mateix script. Com es pot veure en la Figura 6.5.1, la funció espera a que el jugador apreti el botó d'atac. Quan ho fa, comprova que pugui atacar, si pot, executa la funció "Attack()", la qual mira si ha de fer un atac a distància o cos a cos i l'executa, donant, si és necessari, les habilitats dels projectils al projectil llançat, com per exemple, cremar al enemic. A més, la funció "PlayerAttack()", relentitza al jugador quan ataca i li retorna la velocitat inicial quan aquest deixa d'apretar el botó dret del ratolí.

```
private void PlayerAttack()
{
    //Pre: ---
    //Post: the player tries to attack

    if (Input.GetButton("Fire1"))
    {
        if (!shooted && ableToAttack && !cooldown) { Attack(); }
        playerMovement.SpeedDown();
    }
    if (Input.GetButtonUp("Fire1")) { playerMovement.SpeedUp(); }
}
```

Figura 6.5.1: Funció PlayerAttack de l'script WeaponAttack

Canviant de tema, per explicar el pas de les habilitats dels personatges al personatge principal, primer he d'explicar com les emmagatzemo en cada personatge. Vaig decidir fer-ho a través de diccionaris, els quals tenen com a clau el nom de l'habilitat i com a valor

un booleà que determina si l'habilitat està activa en el personatge o no. Hi ha dos diccionaris d'aquest tipus, un que aplega les habilitats que afecten a les armes o els projectils, guardat en l'script "WeaponAttack", i el diccionari que guarda les demés habilitats en un script anomenat "CharacterStats". En el primer cas, l'script els hi passa als projectils, en l'script anomenat "ProjectileHit", i a l'script que tenen les armes cos a cos, anomenat "MeleeWeaponAttack", ja que aquests dos scripts són els encarregats de transmetre el dany als enemics i comunicar quins efectes secundaris tindran. Ara bé, en el segon cas, amb les habilitats com l'escut o la visió, diferents scripts fan referència a "CharacterStats" ja que, com podem veure en les dues habilitats que he posat d'exemple, una instancia un escut en el personatge, mentre que l'altre treballa amb el minimapa, en la Figura 6.5.2, es pot veure com l'arquer pot tenir l'escut.

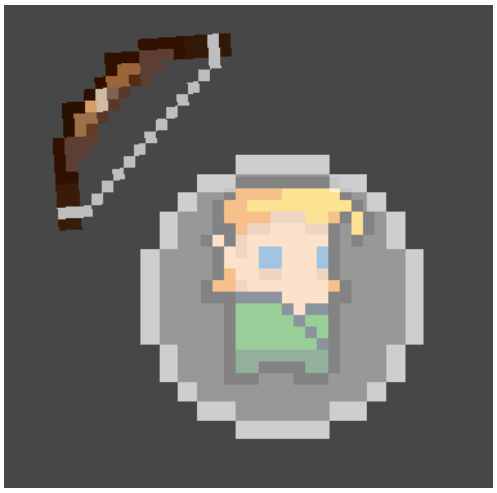

Figura 6.5.2: Arquer amb la habilitat de l'escut que ha aconseguit del cavaller

Explicat això, en el moment en que el jugador interactua amb l'escala que el trasllada al següent pis, apareixen els personatges com a opció per a ser seleccionats. Quan es selecciona un, l'script que conté el botó, anomenat "ButtonSacrifice", aconsegueix les dues habilitats del personatge en la funció "Sacrifice()" on, com es pot veure en la Figura 6.5.2, recorre els dos diccionaris per saber quines habilitats estan actives, guardant d'aquestes la clau, que ve a ser el nom. Un cop fet això, li comunica al objecte "Team", de l'escena principal, quin personatge ha d'eliminar, i seguidament, li passa les dues claus a l'script "Pop_UpSacrifice", el qual crea els botons i fa d'intercomunicador. A continuació, es creen els botons per seleccionar quina habilitat es vol quedar el jugador, amb les habilitats passades per la funció "Sacrifice()". Aquests, contenen l'script anomenat "SelectStat", el qual té al personatge principal com a variable, per així, poder-li passar l'habilitat. En la funció "Select()", que es pot veure en la Figura 6.5.4, li passa l'habilitat al jugador a través de les funcions "AddAbilities()", la qual gestiona si és necessari la adhesió de la habilitat, i també li passa l'habilitat a la funció "AddProjectileStat()", que fa el mateix que l'anterior. Llavors,

tanca la interfície on es mostren els botons. La interfície de la que he estat parlant es pot veure en les Figures 6.5.3 i 6.5.4.

Figura 6.5.3: Menú per seleccionar un personatge

Figura 6.5.4: Menú per seleccionar una habilitat

En la funció "AddAbilities()", que es pot veure en la Figura 6.5.5, compara la variable que li passen amb les habilitats conegudes. En cada cas es fa quelcom diferent, a excepció de posar el valor necessari a cert en el diccionari:

- **Visió:** s'executa la funció "showEndRoom()" de l'script "MiniMapDisplayer" que està en el minimapa i permet veure la habitació final.
- **Reviure:** afegeix un script anomenat "ReviveBehaviour" en el jugador.

- **Minion:** afegeix un script anomenat “SpawnMinionBehaviour” en el jugador.
- **Escut del darrere:** instancia l’escut de l’esquena com a fill del jugador.
- **Escut:** instancia l’escut com a fill del jugador.

En canvi, en la funció “AddProjectileStat()”, com es pot veure en la Figura 6.5.6, simplement torna a cert el valor de l’habilitat en el diccionari.

```
private void Sacrifice()
{
    string perk1 = "";
    string perk2 = "";

    foreach (Transform child in character.transform) //projectiles modifications
    {
        if (!child.CompareTag("Shield"))
        {
            foreach (Transform grandchild in child)
            {
                if (grandchild.GetComponent<WeaponAttack>() != null)
                {
                    WeaponAttack stats = grandchild.GetComponent<WeaponAttack>();
                    foreach (KeyValuePair<string, bool> page in stats.projectileStats)
                    {
                        if (page.Value)
                        {
                            if (perk1 != "") { perk2 = page.Key; }
                            else { perk1 = page.Key; }
                        }
                    }
                }
            }
        }
    }
    if (perk2 == "") //player abilities
    {
        Dictionary<string, bool> stats = character.GetComponent<CharacterStats>().playerAbilities;
        foreach (KeyValuePair<string, bool> page in stats)
        {
            if (page.Value)
                {
                    if (perk1 != "") { perk2 = page.Key; }
                    else { perk1 = page.Key; }
                }
        }
    }

    team.Reorganize(character);
    ui.getStats(perk1, perk2);
}
```

Figura 6.5.4: Funció Sacrifice de l’script ButtonSacrifice

```
private void Select()
{
    character.GetComponent<CharacterStats>().AddAbilities(selectedStat);

    foreach (Transform child in character.transform)
    {
        if (!child.CompareTag("Shield"))
        {
            foreach (Transform grandchild in child)
            {
                if (grandchild.GetComponent<WeaponAttack>() != null)
                {
                    grandchild.GetComponent<WeaponAttack>().AddProjectileStat(selectedStat);
                    break;
                }
            }
        }
    }
    ui.Close();
}
```

Figura 6.5.5: Funció Select de l'script SelectStat

```
public void AddAbilities(string ability)
{
    //Pre: valid ability name
    //Post: add the ability to the character

    if (ability == "vision") { playerAbilities[ability] = true; GameObject.Find("MiniMapCamera").GetComponent<MiniMapDisplayer>().showEndRoom(); }
    else if (ability == "revive") { gameObject.AddComponent<ReviveBehaviour>(); playerAbilities[ability] = true; }
    else if (ability == "spawner") { gameObject.AddComponent<SpawnMinionBehaviour>(); playerAbilities[ability] = true; }
    else if (ability == "backprotect")
    {
        foreach (Transform child in transform){
            if (!child.CompareTag("Shield") && !child.CompareTag("HitDetector"))
            {
                Instantiate(backProtect, child);
                playerAbilities[ability] = true;
            }
        }
    }
    else if (ability == "shield") { instantiatedShield = Instantiate(shield, transform); playerAbilities[ability] = true; }
}
```

Figura 6.5.6: Funció AddAbilities de l'script CharacterStats

```
public void AddProjectileStat(string type)
{
    //Pre: ---
    //Post: adds a new projectile stat

    if (projectileStats.ContainsKey(type))
    {
        projectileStats[type] = true;

        if (GetComponent<MeleeWeaponAttack>() != null)
        {
            GetComponent<MeleeWeaponAttack>().updateStats(projectileStats);
        }
    }
}
```

Figura 6.5.5: Funció AddProjectileStat de l'script WeaponAttack

Amb el jugador no em vaig trobar cap problema a l'hora d'implementar-lo. En canvi, amb la mecànica de suma d'habilitats em vaig trobar en que, en l'últim pis on es pot escollir una habilitat, la segona no sortia. Aquest problema vaig veure que es provocava a causa que, a l'hora de crear els botons, no ho havia fet correctament. Però al adonar-me'n va ser molt senzill corretgir-ho. Per una altra banda, per provar aquests dos elements, ho podia fer en l'escena de proves, o bé, des de l'editor, moure el personatge fins a l'escala de pas de nivell.

6.6 Implementació del Minimapa

El minimapa consta de tres parts, una imatge en el canvas de l'escena que retransmet el que la càmera grava, la càmera, i l'objecte que conté la càmera i les habitacions generades per a mostrar-se en el minimapa. Aquest es dibuixa en una zona de l'escena apartada del mapa, per així no interfer-hi directament.

De la primera part no hi ha res a explicar, ja que simplement és una imatge. La segona és una càmera ortogràfica que mostra el minimapa, enfocant-se en l'habitació que representa que és on està el jugador. Per últim, l'objecte que genera el minimapa i el conté. Aquest, disposa d'un script anomenat "MiniMapDisplayer", el qual conté les imatges per a poder generar el minimapa. Aquest script, quan es genera el mapa, va rebent les habitacions que ha de dibuixar en forma d'habitacions normals o habitacions grans. Quan rep una habitació, el que realment rep és l'script "RoomDoors" de l'habitació i la posició en la matriu del mapa, ja que el minimapa funciona en aquest tema igual a la generació procedural del nivell, només que en la matriu hi guarda objectes de la classe "miniRoom". Al rebre aquesta informació, crea una variable de la classe que hi ha en aquest script anomenada "miniRoom", que es pot

veure en la Figura 6.6.1, i que conté: les portes obertes de l'habitació, si s'ha visitat, un objecte que conté una imatge de l'habitació del minimapa i una llista de vectors que s'usa per saber amb quines habitacions estan connectades les habitacions grans. A continuació, es posa en la variable "miniRoom" si l'habitació ha estat visitada. Es fa per si és l'habitació inicial, i s'instancia l'habitació en el minimapa, guardant-la en la classe "miniRoom". Finalment, es guarda l'objecte "miniRoom" en la matriu del minimapa. Si l'habitació que ens passen és la final, es guarda en la variable endRoom, que s'utilitza si hi ha un personatge amb l'habilitat de visió. És un procés similar per a les habitacions grans, a excepció, que les habitacions adjacents a l'habitació gran es guarden la variable "bigRoomNexts" de la classe "miniRoom". Aquests dos processos es poden veure en les funcions "getNormalRoom()", visible en la Figura 6.6.2, i "getBigRoom()", que es pot veure en la Figura 6.6.3.

```
private class miniRoom{
    5 references
    public string doors; //XXXX == LRDT, If 1 it has a door in that side
    3 references
    public bool visited = false;
    12 references
    public  GameObject roomSprite = null;
    3 references
    public List<Vector2> bigRoomsNexts = new List<Vector2>();
}
```

Figura 6.6.1: Classe miniRoom de l'script MiniMapDisplayer

```
public void getNormalRoom(RoomDoors room, int[] pos)
{
    miniRoom auxRoom = new miniRoom();

    foreach (GameObject sprite in roomSprites)
    {
        if (!room.end)
        {
            if (sprite.name == "basicRoom")
            {
                auxRoom.roomSprite = sprite;
                auxRoom.visited = room.visited;
                auxRoom.roomSprite = Instantiate(auxRoom.roomSprite,
                new Vector3((pos[0] * roomSize)-10, (pos[1] * roomSize)-10, 0), Quaternion.identity, transform);

                miniMapGrid[pos[0], pos[1]] = auxRoom;
            }
        }
        else
        {
            if (sprite.name == "EndRoom")
            {
                auxRoom.roomSprite = sprite;
                Destroy(miniMapGrid[pos[0], pos[1]].roomSprite);
                auxRoom.roomSprite = Instantiate(auxRoom.roomSprite,
                new Vector3((pos[0] * roomSize)-10, (pos[1] * roomSize)-10, 0), Quaternion.identity, transform);

                miniMapGrid[pos[0], pos[1]] = auxRoom;
                endRoom = auxRoom;
            }
        }

    }
}
```

Figura 6.6.2: Funció getNormalRoom de l'script MiniMapDisplayer

```
public void getBigRoom(BiggerRoomCapacity room, int[] pos, int[] center)
{
    miniRoom auxRoom = new miniRoom();

    foreach (GameObject sprite in roomSprites)
    {
        if (sprite.name == room.roomName)
        {
            auxRoom.roomSprite = sprite;

            Vector2[] auxGrids = room.getCapacity(room.entranceSide);
            foreach (Vector2 grid in auxGrids) { auxRoom.bigRoomsNexts.Add(grid); }

            auxRoom.roomSprite = Instantiate(auxRoom.roomSprite,
            new Vector3(((pos[0] * roomSize) + ( (center[0]/10f)/2f ))-10f, ((pos[1] * roomSize) +
            ((center[1]/10f)/2f))-10f, 0), Quaternion.identity, transform);

            miniMapGrid[pos[0], pos[1]] = auxRoom;
        }
    }
}
```

Figura 6.6.3: Funció getBigRoom de l'script MiniMapDisplayer

A partir d'aquí, el que es fa és que les portes li comuniquen a l'script quina habitació s'està visitant, si aquesta no ho estava, es mostra. Això es duu a terme en la funció "show()" del mateix script i que es pot veure en les Figures 6.6.4 i 6.6.5. Aquesta funció, primer de tot, executa la funció "showRoom()", la qual habilita la imatge de l'habitació del minimapa per a que es pugui mostrar, a més, de pintar la imatge de color blanc i posar l'habitació com a visitada. A continuació, si és una habitació normal, comprova quines portes estan obertes. De les que sí que ho estan, executa la funció "nextToShown()", la qual si la conseqüent habitació no ha estat visitada, es mostra la imatge de l'habitació i es pinta de gris. En canvi, si l'habitació és gran, s'executa la funció "nextToShown()" amb les habitacions de la matriu que estiguin en les posicions de la variable "bigRoomsNexts" de la classe "miniRoom".

```
private void show(miniRoom room, int[] pos)
{
    showRoom(room);

    if (room.bigRoomsNexts.Count == 0) //normal size rooms
    {
        if (room.doors[0] == '1')
        {
            if (isInside(pos[0]-1, pos[1]) && miniMapGrid[pos[0]-1, pos[1]] != null)
                {
                    nextToShown(miniMapGrid[pos[0]-1, pos[1]]);
                }
        }
        if (room.doors[1] == '1')
        {
            if (isInside(pos[0]+1, pos[1]) && miniMapGrid[pos[0]+1, pos[1]] != null)
                {
                    nextToShown(miniMapGrid[pos[0]+1, pos[1]]);
                }
        }
        if (room.doors[2] == '1')
        {
            if (isInside(pos[0], pos[1]-1) && miniMapGrid[pos[0], pos[1]-1] != null)
                {
                    nextToShown(miniMapGrid[pos[0], pos[1]-1]);
                }
        }
        if (room.doors[3] == '1')
        {
            if (isInside(pos[0], pos[1]+1) && miniMapGrid[pos[0], pos[1]+1] != null)
                {
                    nextToShown(miniMapGrid[pos[0], pos[1]+1]);
                }
        }
    }
```

Figura 6.6.4: Primera part de la funció show de l'script MiniMapDisplayer

```
else //big rooms
{
    foreach (Vector2 position in room.bigRoomsNexts)
    {
        int[] auxPos = { pos[0]+(int)position.x, pos[1]+(int)position.y };

        if (isInside(auxPos[0], auxPos[1]) && miniMapGrid[auxPos[0], auxPos[1]] != null)
        {
            nextToShown(miniMapGrid[auxPos[0], auxPos[1]]);
        }
    }
}
```

Figura 6.6.5: Segona part de la funció show de l'script MiniMapDisplayer

Per últim, en el minimapa no em vaig trobar cap problema. Per provar-lo havia de jugar al videojoc i passar a través de les portes. En les Figures 6.6.6 i 6.6.7, es pot veure el seu funcionament.

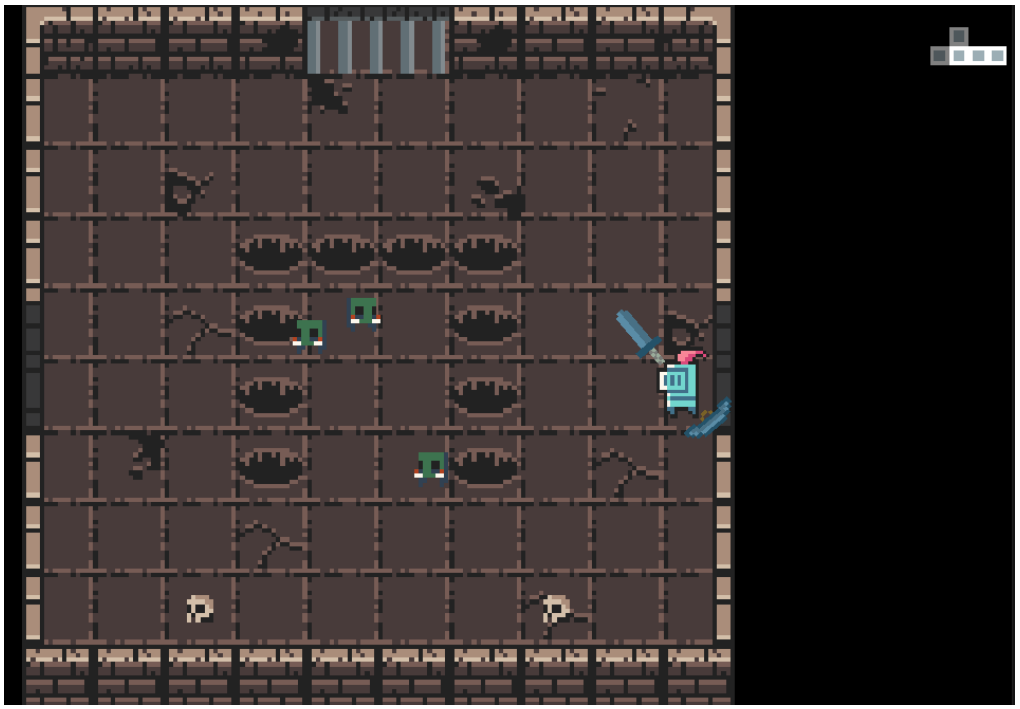

Figura 6.6.6: Imatge on es mostra el minimapa

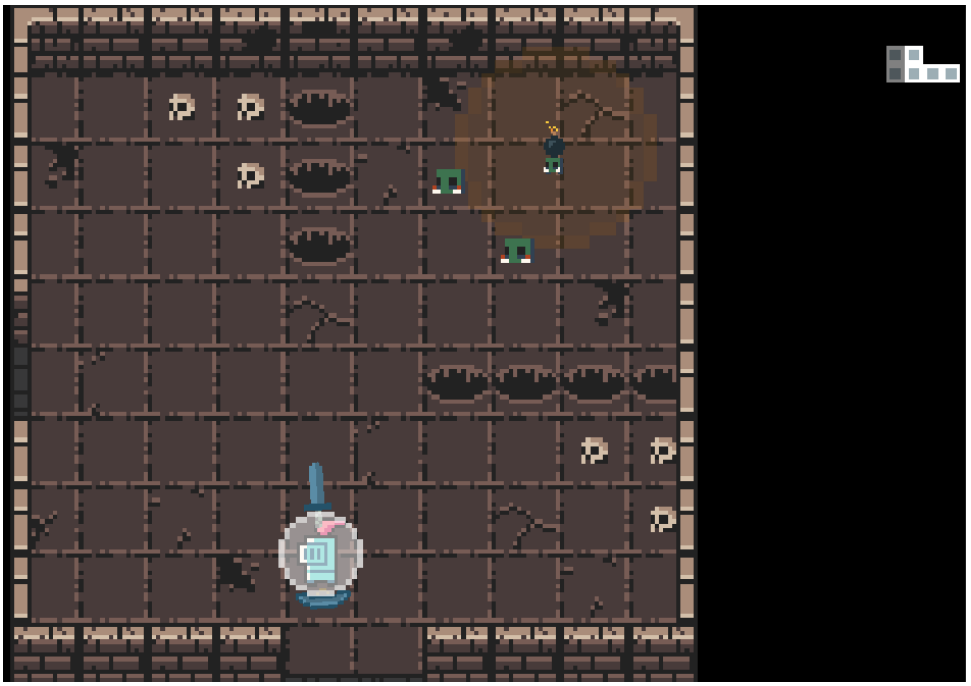

Figura 6.6.7: Imatge on es mostra el minimapa on el jugador ha avançat cap amunt

7. Resultats

7.1 Grau d'Assoliment

En aquest apartat es mirarà fins a quin punt s'han assolit els objectius inicials del projecte.

- **Disseny del videojoc:** en aquest sentit crec que he fet un bon treball dissenyant el videojoc en general, ja que té un sistema de combat estable, enemics que es poden fer difícils de combatre, una batalla final i una generació de mapa procedural. Tot i això i ser un prototip, crec que es podria haver intentat un sistema de recompenses com podria ser que al netejar una habitació d'enemics poguessis recollir monedes, o un sistema de puntuació que premiés el matar enemics i visitar habitacions.
- **Estudiar i implementar un algorisme de generació procedural per a la masmorra:** aquesta és la part de la que més content estic de tot el projecte. Després de mirar molts tipus d'algorismes, cap em va convencer, així que vaig decidir pensar en un de nou.
- **Implementar les mecàniques principals del jugador:** en aquest cas, les mecàniques són molt simples però són el que necessita aquest joc. No sempre és necessari fer-ho tot complexe, i en el cas de les mecàniques del jugador, la simplicitat ha funcionat bé.
- **Estudiar i implementar IAs pels aliats i pels enemics:** la IA dels aliats és molt simple, i crec que ha de ser així per a que el jugador s'hagi d'esforçar per a poder arribar a la lluita contra l'enemic final. En canvi, la IA dels enemics és més complexa, i estic orgullós del disseny. Això sí, podrien tenir més comportaments com la possibilitat de que esquivin alguns atacs.
- **Desenvolupament de la mecànica de prescindir d'un aliat i així obtenir una de les seves habilitats, per fer-les servir a partir d'aquell moment:** tot i que al principi pensava que seria una tasca complexa, de sobte va ser molt simple a l'utilitzar dicionaris. Per tant, crec que el desenvolupament d'aquesta mecànica ha estat completada, amb lloc per a poder implementar més habilitats juntament amb nous personatges.
- **Arrodoniment de la documentació:** crec que aquest document és prou complet, sense deixar-me res important pel camí sense explicar.

En les Figures 7.1.1, 7.1.2, 7.1.3 i 7.1.4 es poden veure diferents etapes d'una partida en la que en la primera, tot l'equip està viu, en la segona, ja han mort els companys, en la tercera, el mag ha adquirit una habilitat del cavaller, i en l'última, el jugador està lluitant contra el boss.

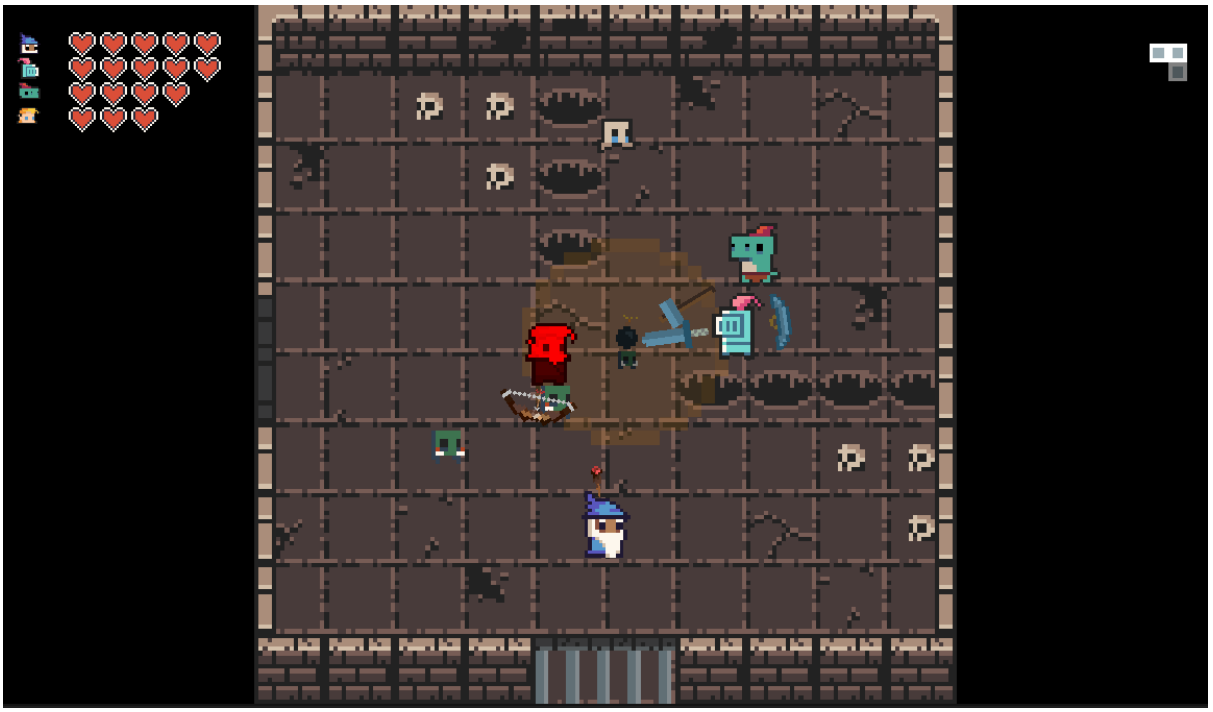

Figura 7.1.1: Imatge on tot l'equip està viu

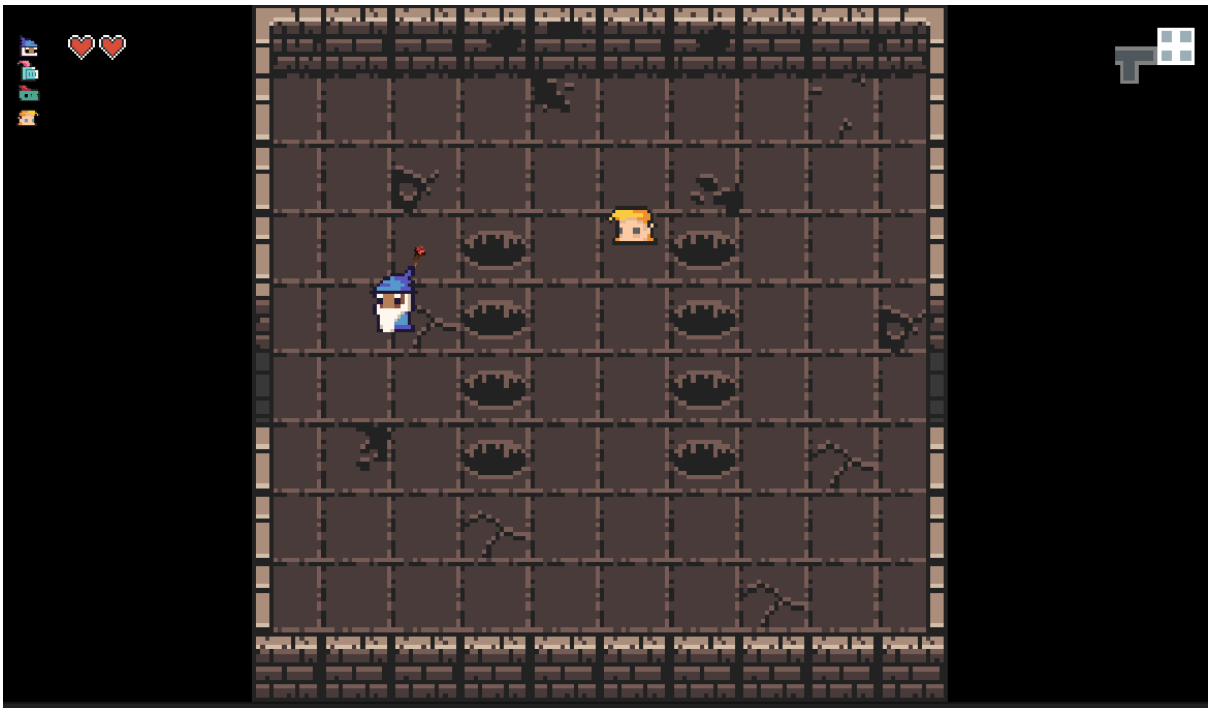

Figura 7.1.2: Imatge on l'equip ha mort

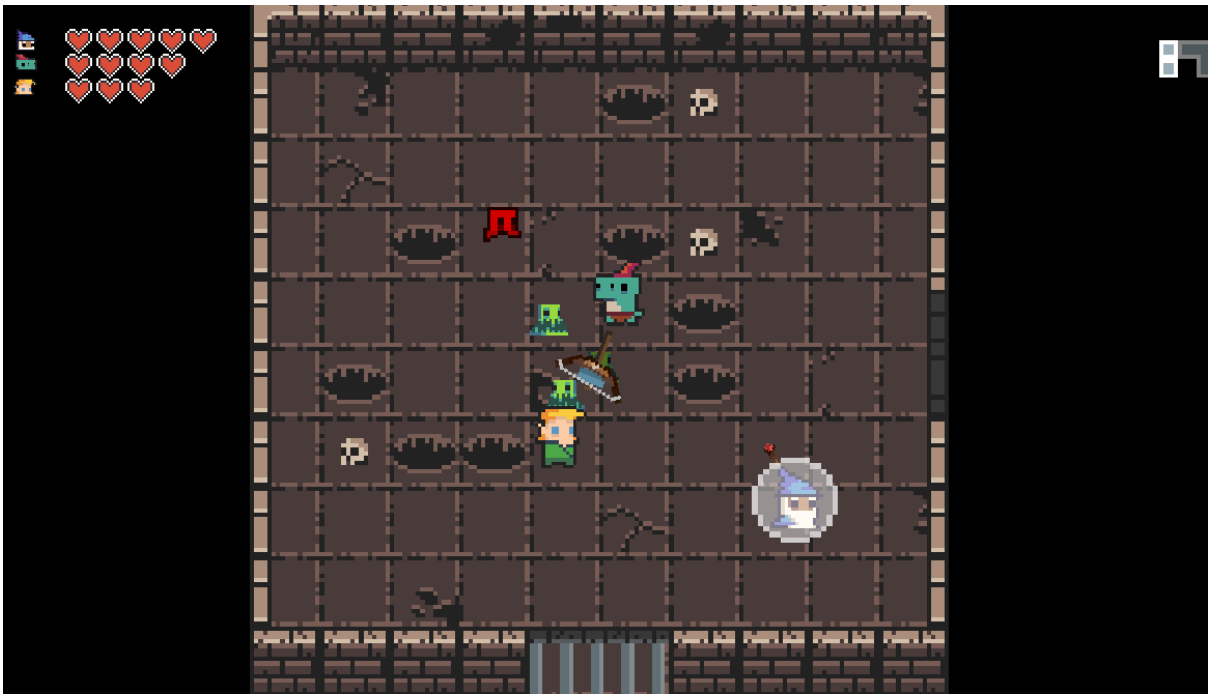

Figura 7.1.3: Imatge on el mag ha adquirit l'escut

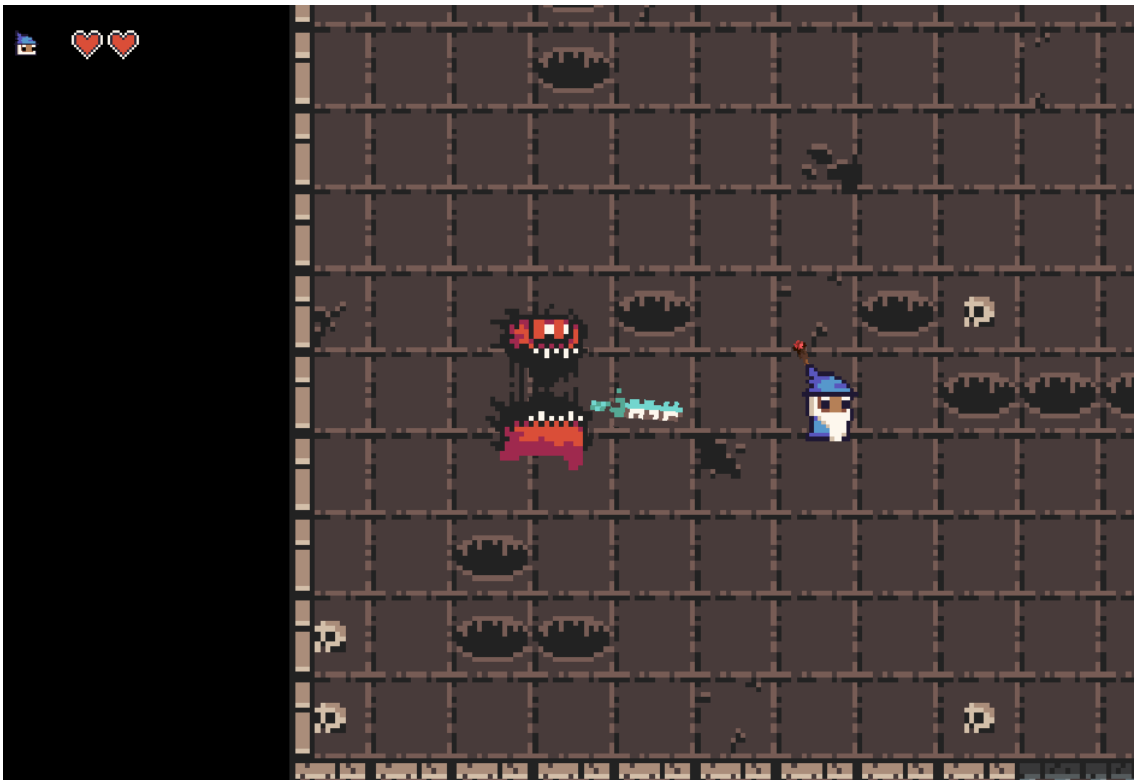

Figura 7.1.4: Imatge on el jugador està lluitant contra l'enemic final

7.2 Legislació i Normativa Vigent

El joc desenvolupat no presenta cap problema en aspectes legislatius. En cap moment es guarda informació de caràcter personal del jugador, per tant no s'aplica en cap situació la LOPD (Llei Orgànica de Protecció de Dades). Tampoc s'aplica la LSSICE (Llei de Serveis de la Societat de la Informació i Comerç Electrònic), ja que el projecte no suposa cap activitat econòmica.

Pel que fa als problemes de Copyright, la major part artística del projecte ha estat obtinguda de tercers però sempre en condicions d'ús lliure del software, art en aquest cas. Per una altra part, el joc ha estat desenvolupat completament per mi mateix, per tant no m'hauria de suposar cap problema si el volgués comercialitzar, encara que és una cosa que vulgui fer. Tot i així, si la comercialització del joc fos un èxit i es superés un cert llindar d'ingressos definit pels propietaris de Unity3D, em veuria obligat a comprar les llicències professionals del software per a no tenir problemes legals.

7.3 PEGI

El sistema Pan European Game Information (PEGI), és un sistema europeu de qualificació dels videojocs mitjançant icones. S'utilitza per a classificar els jocs per edat recomanada i descriure el contingut sensible que apareix en el joc, com ara la presència de violència, drogues o sexe. En cap moment les classificacions són de compliment obligatori, les classificacions són merament informatives.

En el cas d'aquest projecte, Pawn Sacrifice, l'únic que s'hauria d'advertir és de la violència, tot i que aquesta és lleu, a causa que es dóna en éssers fantàstics i poc realistes sense la aparició de sang. Per tant, hauria de tenir una classificació de PEGI 7.

Figura 7.3.1: Edats i etiquetes de PEGI

8. Conclusions

8.1 Valoració del Projecte

Amb aquest treball he pogut posar en pràctica moltes de les coses que he après en el grau de disseny i desenvolupament de videojocs, però, he vist que encara em queden moltes coses per aprendre.

La idea per a aquest projecte va sorgir un dia que estava parlant amb un amic, i vaig veure que tenia potencial pel que me la vaig guardar per a poder-la utilitzar en el treball de final de grau. Des d'un principi tenia clar quin seria el nucli del videojoc, no obstant, el procés de dissenyar i pensar en tots els demés aspectes del videojoc ha fet que gaudeixi molt més aquest treball, i que vegi lo complicat que pot arribar a ser dissenyar un videojoc.

Mentre desenvolupava el joc, se m'anaven ocurrent idees per implementar. També passava això en les tutories amb el meu tutor del projecte, en Gustavo Patow. Però, com és normal i lògic moltes d'elles només es van quedar en idees, a causa que, amb el temps limitat que tenia, no podia implementar tot el que se'm passés pel cap.

D'altra banda, aquest projecte és un prototip. Pel que es pot ampliar en grans mesures, sobretor essent un joc amb les característiques que té. Per exemple, té la possibilitat d'anar implementant personatges, per a disposar de més habilitats, més enemics i enemics finals, més pisos, recompenses… Com s'ha pogut veure, aquest prototip es podria allargar molt, i segurament ho faci, ja que tinc ganes de seguir-lo desenvolupant i arribar a deixar-lo en un estat de videojoc complet.

El que sí que he vist és que tinc una gran carència artística, no només per haver agafat l'art de tercers sinó perquè les interfícies d'usuari són molt millorables. És una cosa que he d'anar polint amb esforç i dedicació.

Per acabar, dir que estic content amb el resultat d'aquest prototip, compleix tots els objectius que em vaig posar i cap d'ells el vaig fer amb desganes. A més, he disfrutat molt del procés.

8.2 Desviacions de la planificació original

Els motius principals pels que s'ha desconfigurat la planificació inicial és el fet de no saber d'avantmà què tant complicada serà una tasca. A més, en alguns casos, com pot ser el disseny i implementació de l'algorisme de generació de mapa procedural, en dissenyava una part per, al cap d'un temps, acabar-ho. En l'exemple que he posat, vaig dissenyar l'algorisme

per a que tingués només habitacions normals, seguidament em vaig posar amb el disseny dels personatges i una part de la seva implementació. Això ho vaig fer a causa que necessitava poder provar que l'interior del mapa era funcional.
Per una altra banda, vaig buscar l'art per internet molt abans del que em pensava, ja que volia implementar les animacions en els personatges. També, he hagut de retocar les interfícies d'usuari en la penúltima setmana.

En la Figura 8.2.1 es pot veure com ha quedat la planificació.

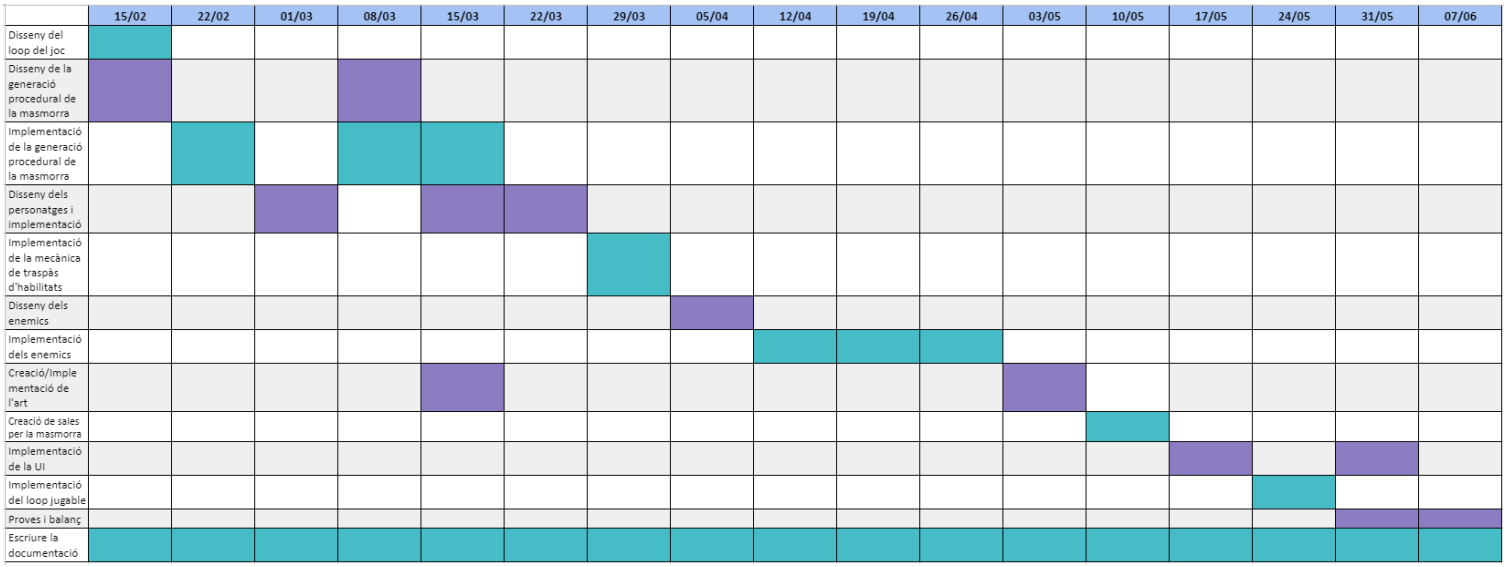

Figura 8.2.1: Planificació final del projecte

9. Treball Futur

Com ja he anat dient, m'agradaria fer d'aquest projecte un videojoc en un estat complet. De tal manera, se li hauria d'afegir:

- **Sons i música:** ja que ara mateix el joc careix d'aquests elements.
- **Més personatges:** la implementació de més personatges donaria més diversitat a les partides, ja que amb cada personatge afegit, s'estarien afegint dues habilitats.
- **Més enemics**: quants més enemics, més dinàmic i menys repetitiu es sentirà el combat.
- **Més layouts d'habitacions:** l'afegir nous layouts per a cada habitació s'estarà afegint més varietat al joc, pel que el sentiment de que cada partida és diferent s'incrementa.
- **Tipus de pisos:** fer que cada X pisos la temàtica d'aquests variï. Amb això aconseguim un altre cop més varietat en cada partida, i a més, trenquem la monotonia de sempre veure les habitacions semblants en quan a estètica.
- **Un sistema de recompensa:** bé podria ser un sistema de puntuació per a que el jugador es sentís incentivat, o bé podria ser implementar un sistema econòmic basat en monedes, les quals es podrien intercanviar per objectes en una tenda.
- **Objectes:** implementar objectes que es trobessin a l'hora d'investigar cada pis, els quals augmentessin característiques base com la velocitat o el dany. Això faria que els jugadors anessin a totes les sales de cada pis, buscant objectes, i que es sentissin més podersos per haver-ho fet.
- **Millorar l'art i les interfícies:** és necessària una millora en l'art del videojoc i en les interfícies visuals.

10. Bibliografia

- Unity Tecnologies. Unity Manual. https://docs.unity3d.com/
- Unity Tecnologies. Unity Answers. https://answers.unity.com/
- Unity Tecnologies. Unity Forum. https://forum.unity.com/
- GitHub, Inc. GitHub. https://github.com/
- Brackeys. Canal de youtube "Brackeys". https://www.youtube.com/brackeys
- Raymond Larabie (2008). "Joystix Font". https://www.1001fonts.com/joystix-font.html
- 0x72 (2019). "16x16 DungeonTileset II". https://0x72.itch.io/dungeontileset-ii

11. Annex

Com a annex he adjuntat tot el projecte juntament amb la memòria per a que es pugui mirar qualsevol script. També es pot donar un cop d'ull a l'art del projecte. Tots els Sprites es troben a: Assets/Resources/Sprites. Mentre que els scripts es troben a: Assets/Resources/Scripts.

12. Manual d'Usuari i d'Instal·lació

Per a instal·lar el videojoc, primer de tot, s'ha de descarregar l'arxiu .zip on es troba la carpeta amb el joc i la documentació. A continuació, es descomprimeix l'arxiu, obtenint, així la documentació del projecte i la carpeta on es troba l'executable del videojoc. Un cop tenim a l'escriptori o a la carpeta convenient, la carpeta del joc anomenada "Pawn Sacrifice", la obrim i fem doble clic a l'executable anomenat, també, "Pawn Sacrifice".

Per a poder jugar, simplement es fa clic al botó que posa "Play" i ja estem en una partida. LLavors els controls són els següents:

- **W:** moure's cap amunt.
- **A:** moure's cap a l'esquerra.
- **S:** moure's cap abaix.
- **D:** moure's cap a la dreta.
- **ESC:** per a posar el menú de pausa.
- **Clic esquerra del ratolí:** atacar o seleccionar un botó en els menús.

Es pot sortir del joc en qualsevol moment anant al menú de pausa i clicant a "Quit", o bé, clicant al mateix botó quan es guanyi o perdi la partida.

13. Agraïments

Per acabar, m'agradaria agrair sincerament les tutories amb en Gus, les quals m'han ajudat molt per a completar aquest projecte. També m'agradaria donar les gràcies a l'Oriol Perernau, company de classe, el qual em va ajudar a pensar la idea sobre la generació procedural del mapa. I finalment, gràcies a la meva família pel suport que m'ha donat al llarg de tot el projecte.