Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Desenvolupament d'un joc Endless Survival

Document: Memòria

Alumne: Antoni Martin Rabaneda

Tutor: Gustavo Patow

Departament: Informàtica, Matemàtica Aplicada i Estadística

Àrea: Llenguatge i Sistemes Informàtics

Convocatòria Juny/2021

Index

1 Introducció i objectius

1.1 Introducció

En aquesta memòria es descriu tota la feina realitzada individualment respecte al projecte de dissenyar i desenvolupar un videojoc des de zero; passant per diverses fases com l'elecció del tema especificant els objectius, un estudi per saber la viabilitat d'aquests i com s'han planificat, un marc de treball amb els conceptes previs a tenir en compte, i finalment el disseny del videojoc.

Durant aquesta última dècada, s'ha incrementat molt la importància dels videojocs dintre de la societat. Amb l'aparició dels dispositius mòbils, s'ha permès que aquests arribin a un major número de gent, sobretot persones no acostumades al sector. A causa d'això, s'ha sobreexplotat el mercat de videojocs en dispositius mòbils, on molts d'ells se centren en un disseny simple per atraure aquest públic tan poc familiaritzat, centrant-se en els gèneres i mecàniques que més atrauen i perdent la diversitat que tenen altres plataformes. Per aquest motiu, s'ha decidit crear un videojoc que aporti varietat al sector, així com sobresortir en aquest disseny estandarditzat.

Un dels gèneres que van sobresortir en gran manera dintre del sector, són els jocs 'Endless Runner'. Aquest gènere aporta un estil de videojoc semblant a les recreatives, on la base se centra en partides curtes amb l'objectiu d'arribar el més lluny possible, aconseguint la màxima puntuació, i tornant-ho a intentar una vegada perdut. Observant l'estudi de mercat, es pot veure com els videojocs d'aquest gènere solen tenir la mateixa estructura, sense arriscar-se a variar-les per tal d'oferir un producte diferent, o arribar a un nou públic.

El videojoc presentat en aquest projecte, és una variant d'aquest gènere, a la que s'ha denominat 'Endless Survival'. Aquest, agafa la idea principal dels 'Endless Runner', però afegint mecàniques diferents, afegint un toc de complexitat al joc. L'objectiu, però, no és avançar, sinó sobreviure el màxim temps possible a diferents onades d'enemics, utilitzant tant l'escenari com les habilitats de cada personatge.

Amb aquest projecte s'intenta oferir una experiència diferent de l'acostumada per les persones que juguen casualment, endinsant-los a mecàniques complexes i diferents del mercat actual, així com atraure el públic veterà a jocs més senzills i ràpids.

1.2 Motivació

Les motivacions en l'àmbit personal que em porten a voler desenvolupar aquest projecte es poden resumir amb les següents:

1. Poder treballar en cada apartat de la creació, des del disseny de les mecàniques principals, passant per la programació del joc, fins a la creació d'art.
2. Posar en pràctica totes les habilitats adquirides durant el grau, combinant-les i adaptant-les en un projecte més ambiciós als fets prèviament al curs.
3. Adquirir nous coneixements tant en el procés de creació, com de les eines utilitzades (Unity, Maya...)
4. Permetre tenir una base per continuar amb l'objectiu de publicar el videojoc una vegada completat.

1.3 Propòsit i objectius del projecte

El propòsit del projecte és el de crear un videojoc sencer, complet, de principi a fi, d'una qualitat remarcable. A més a més, es busca donar forma a una variació d'un gènere poc explorat, amb la finalitat d'aprendre a treballar sobre un llenç buit on poder donar-ho tot i mitjançant el qual poder desenvolupar els coneixements ja obtinguts. Així mateix, es busca trobar un lloc en el mercat dels videojocs per dispositius mòbils, per tal de sobresortir de la resta i aportar varietat en el sector.

De manera més concreta, els meus objectius són els següents:

1. Dominar Unity.
2. Treballar en el disseny per tal de crear unes mecàniques divertides i atractives per diferents públics del sector.
3. Crear un bon balanceig de joc.
4. Crear un sistema de classes amb una bona estructura, fàcilment modificable i escalable.
5. Trobar una concordança amb l'estil artístic dels elements que componen el videojoc.
6. Endinsar-se i aprendre sobre el sistema de música i sons.
7. Donar forma a una variació d'un gènere poc explorat.
8. Crear un sistema de joc amb la capacitat de poder-se jugar diverses vegades sense acabar en monotonia.

1.4 Distribució de tasques

Normalment un videojoc és creat per un equip multidisciplinari, però en aquest cas només una persona s'encarregarà de totes les diferents parts que l'engloben. A continuació es mostra un quadre (Taula 1) amb una valoració personal de quant d'esforç i valor es dedica a cada un d'aquests quatre elements

Estètica	15%
Narrativa	5%
Mecàniques	40%
Tecnologia	40%

Taula 1 Quadre d'autoavaluació

Un dels blocs on es gastaran més recursos és clarament el de les mecàniques. La proposta seleccionada està pensada per a oferir unes mecàniques innovadores i divertides a les ja establides en el gènere. Sumant-hi el bon balanç i la poca monotonia en cada partida. A causa d'això, es requerirà un temps en pensar i provar diferents opcions, per tal d'obtenir el millor resultat possible.

El segon blog més important és la tecnologia, ja que es treballarà amb el motor Unity, i la programació ha d'oferir una experiència adequada als objectius establerts. Requereix el seu temps per tal que el videojoc pugui funcionar correctament, així com poder aplicar totes les mecàniques anteriorment pensades.

L'estètica tindrà un pes molt baix, ja que només es dissenyarà i crearà els elements necessaris, extraient d'internet la resta de contingut. S'haurà d'intentar concordar tot l'estil artístic, així com afegir els efectes visuals amb l'objectiu d'atraure el màxim públic possible.

La narrativa és pràcticament nula, ja que és un videojoc centrat en partides ràpides i sense molta profunditat. La història només servirà com a excusa narrativa per donar-li sentit als objectius i al món, així com els personatges que l'habiten.

2 Estudi de viabilitat

2.1 Perfil dels jugadors

El joc està orientat a jugadors més acostumats a jugar videojocs per dispositius mòbils, que busquin mecàniques més complexes a les comunament jugades. El videojoc busca endinsar a aquest públic a més en profunditat en el sector, augmentant la dificultat però sense ser complicat d'aprendre. Així mateix, es busca atraure a un públic més experimentat a provar un videojoc de major senzillesa i simplicitat, desafiant-los a una experiència exigent.

Per definir més específicament el perfil de jugador del joc proposat, s'analitzarà la tipologia de jugadors definida per Richard Bartle i que s'ha estudiat durant el curs. Segons Richard Bartle, es pot classificar l'estil de joc dels jugadors en quatre diferents tipus de perfil.

> **Achievers:** tenen com a objectiu resoldre el nombre més gran de reptes i dificultats. Obtenen el plaer resolent situacions complexes. Interessats a actuar en el món.
> **Explorers:** els agrada explorar a fons el videojoc, descobrir els secrets i aprendre coses desconegudes. Interessats a interactuar amb el món.
> **Socializers:** els hi agrada més interactuar amb altres persones que el mateix rerefons del videojoc. Interessats a interactuar amb els jugadors.
> **Killers:** els hi agrada competir contra altres jugadors. Interessats a actuar sobre els jugadors.

En el tipus de videojoc proposat, se centra a desafiar al jugador en situacions desesperants, combinant diferents tipus d'enemics en infinites onades. A causa d'això el perfil objectiu són els 'Achievers', els quals buscaran sobreviure cada vegada més onades, provant diferents combinacions per aconseguir-ho. Per complir aquest objectiu, s'haurà d'estar segur que el joc sigui suficientment desafiant, evitant avorrir al jugador ràpidament.

2.2 Estudi de mercat

Un cop tenim una idea de com serà el joc que volem desenvolupar, i abans de prendre decisions importants del disseny i la implementació, hem de tenir en compte l'estat actual del mercat dels videojocs, especialment del gènere que serà el joc, i analitzar la competència. En aquest apartat es detallaran més aquests aspectes.

2.2.1 Estat de l'art

És important estudiar de forma exhaustiva els millors exponents del gènere per a identificar els aspectes positius que han portat el joc a l'èxit, i també detectar quins aspectes són millorables o canviaríem per a oferir una experiència de joc més satisfactòria. A més, hem de tenir en compte que el joc ha d'aportar alguna novetat perquè sigui interessant. Els principals jocs que s'han analitzat comparteixen gènere amb el projecte a desenvolupar, o que compten amb una temàtica i control similars. A continuació es comentarà breument els jocs analitzats i els aspectes més rellevants de cada un.

ReDungeon (Figura 2.2.1) és un videojoc denominat com a 'Endless Dungeon'. El jugador ha d'avançar por una masmorra que es va generant aleatòriament, esquivant enemics i trampes fins a morir. Compta amb una sèrie de personatges amb habilitats que es poden utilitzar en la partida. El jugador simplement pot moure's a una casella adjacent i utilitzar l'habilitat del personatge. És el videojoc que s'ha agafat com a referència per fer la base del projecte, ja que aporta una variació del gènere.

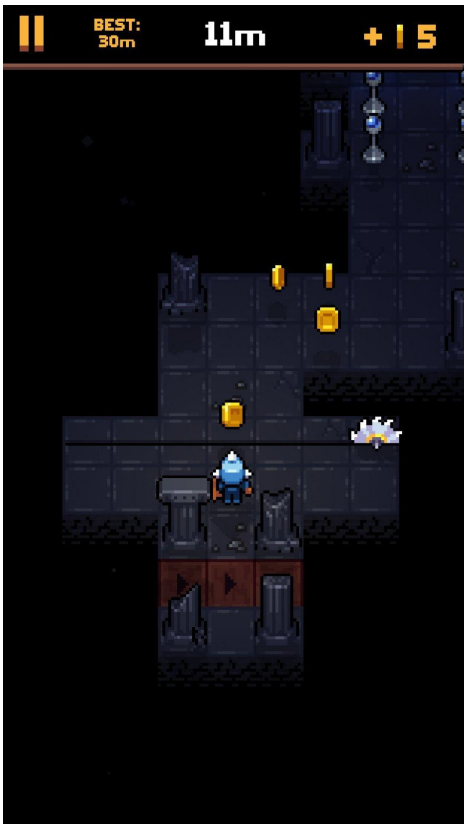


Figura 2.2.1 Captura ReDungeon

Temple Run 2 (Figura 2.2.2) és una seqüela d'un dels jocs més populars del gènere 'Endless Runner', sent un dels que va crear les bases. L'objectiu principal del joc és arribar la màxima distància, superant una varietat de temples. Ofereix una gran senzillesa de controls, permetent jugar simplement lliscant i fent clic sobre la pantalla. A més, aquesta seqüela presenta molt de contingut nou, com personatges, poders o temples.

Figura 2.2.2 Captura Temple Run 2

Molt semblant a Temple Run, mes tard apareix Subway Surfers (Figura 2.2.3) per fer-se un lloc en aquest gènere. Agafa les mecàniques principals i les simplifica més, aplicant un estil més 'Cartoon'. En una partida es pot canviar de via o saltar, podent pujar por sobre de trens, centrant-se en les plataformes. La gran variació és el gran contingut de personalització, permetent comprar diferents personatges i cosmètics per personalitzar-los.

Figura 2.2.3 Captura Subway Surfers

Oferint una experiència diferent, passant a les dues dimensions apareix Jetpack Joyride (Figura 2.2.4). Agafa les bases del gènere i simplifica encara mes les mecàniques, permeten jugant simplement amb un clic. A més, permet transformar els personatges amb diferents propulsors, obtenint diferents avantatges i inconvenients. Posa a disposició una gran varietat de cosmètics, tots fàcilment comprables amb els diners aconseguits en cada partida.

Figura 2.2.4 Captura Jetpack Joyride

2.2.2 Paràmetres avaluació

A continuació, compararem els principals elements que interessen per veure en què ens diferenciem del mercat, així com dels nostres punts forts i debilitats. Por cada paràmetre, assignarem un número del 0 al 10.

Els paràmetres són:

Originalitat: Interessa destacar l'originalitat d'aquest projecte, innovant en el gènere. Por això, es vol saber si som suficients originals respecte a la competència.

0: No aporta cap novetat en el gènere, repetint el que ja està creat.

10: Aporta una gran novetat respecte al mercat, destacant les mecàniques o contingut.

Simplicitat: Inclús amb els canvis que es volen aplicar, es vol mantenir en certa manera la simplicitat que caracteritza al gènere.

0: Presenta una jugabilitat molt complicada, dificultant l'aprenentatge.

10: Ofereix una experiència fàcil d'aprendre i de dominar, permetent a qualsevol gaudir del joc.

Varietat: Encara sent la repetició un element indispensable, oferir una varietat de jugabilitat entre partides pot ajudar a diferenciar-se.

0: Es presenta sempre la mateixa jugabilitat, fent-se repetitiu fàcilment.

10: Ofereix molta varietat d'opcions, fent de cada partida única.

Contingut: El diferent contingut del joc també pot ajudar a evitar avorrir al jugador en una partida llarga.

0: No ofereix molt de contingut, tenint els mateixos elements al llarg del joc.

10: Conté molt de contingut diferent, sorprenent al jugador inclús després de jugar-hi hores.

A continuació, assignarem una nota a cada videojoc i els compararem amb el nostre (Taula 2.1) per poder extreure una conclusió.

	ReDungeon	Temple Run 2	Subway Surfers	Jetpack Joyride
Originalitat	8	7	5	7
Simplicitat	6	7	8	9
Varietat	7	5	5	5
Contingut	5	6	6	7

Taula 2.1 Comparativa mercat

Per al projecte es consideren les següents notes:

Originalitat: 8. Un dels punts forts és l'originalitat, ja que s'ha variat la base per oferir una experiència nova, però sense canviar-la massa.

Simplicitat: 5. A causa dels canvis i mecàniques afegides, la simplicitat s'ha tornat el punt feble del projecte, augmentant la dificultat del gènere.

Varietat: 7. Gràcies a l'escenari canviant i l'aleatorietat d'aparició dels enemics, el joc ofereix diferencies entre cada partida, per evitar la repetició. Així i tot, es té com a objectiu afegir més varietat en un futur amb l'objectiu de millorar aquest punt.

Contingut: 8. Tant pels personatges i les mecàniques, com els diferents enemics, trampes i bonificacions, el joc ofereix una bona quantitat de contingut, permeten moltes opcions disponibles.

2.2.3 Conclusió

El projecte pot tenir un lloc en el mercat a causa dels seus punts forts que el fan diferenciar de la resta. Tot i que té alguns punts fluixos, com la simplicitat, ho compensa amb l'originalitat, la varietat i el contingut. A més, es podrien pensar canvis per tal de simplificar la jugabilitat sense perdre l'essència.

2.3 Viabilitat del projecte

En aquest capítol es valorarà la viabilitat del projecte des de diferents aspectes: els recursos tècnics, els recursos humans o el cost econòmic.

2.3.1 Recursos Tecnològics

Establim tots els elements tecnològics necessaris per permetre desenvolupar aquest projecte.

Software necessari:

Google Drive: Per la realització d'aquesta memòria i per altres documents realitzats durant el projecte, s'ha utilitzat l'editor de text Google Drive (Figura 2.3.1). S'ha decantat per aquest, ja que és una eina totalment gratuïta que permet guardar els fitxer en el núvol de forma molt senzilla.

Figura 2.3.1 Logo Google Drive

Unity3D: Unity (Figura 2.3.2) és un motor de jocs multiplataforma creat per Unity Technologies. Està disponible per a Windows, Mac i Linux, i té suport de compilació per a més de 20 plataformes, entre les quals es troben PC, les consoles principals, dispositius mòbils i WebGL.

Es basa en Mono, un entorn de desenvolupament integrat lliure i gratuït, dissenyat primordialment per al llenguatge C#. Compta amb eines i components per a crear jocs 3D i 2D. És totalment gratuït si no se supera un cert llindar d'ingressos, que si se supera, el desenvolupador ha d'adquirir una llicència pro (de pagament).

Figura 2.3.2 Logo Unity

Visual Studio: Com a editor de codi principal per a escriure el codi del joc s'ha utilitzat l'entorn Visual Studio (Figura 2.3.3), un editor de codi molt potent que permet depurar els projectes de Unity en temps real. Es va decantar per aquest perquè és l'editor predeterminat de Unity.

Figura 2.3.3 Logo Visual Studio

GitHub: Per a mantenir una còpia de seguretat actualitzada en el núvol i fer un seguiment de les modificacions del projecte de Unity, s'ha utilitzat l'aplicació GitHub Desktop (Figura 2.3.4). Permet sincronitzar i actualitzar el projecte amb el repositori de GitHub i fer totes les operacions que ens ofereix la web.

Figura 2.3.4 Logo GitHub

Blender: És un programari lliure multiplataforma, dedicat a l'edició d'elements 3D. Això inclou eines per al modelatge d'objectes, l'edició de materials, l'animació i el render. S'ha decidit Blender (Figura 2.3.5) per ser una de les millors opcions quant a programari 3D gratuït.

Figura 2.3.5 Logo Blender

InkScape: Per fer alguns elements d'interfície, s'ha utilitzat el software gratuït de dibuix vectorial InkScape (Figura 2.3.6). És l'opció gratuïta del programa de pagament Adobe Illustrator.

Figura 2.3.6 Logo Inkscape

Substance Painter: Software per la creació de textures a elements 3D. Conte totes les eines necessàries per al texturitzat. S'ha triat Substance Painter (Figura 2.3.7) a causa de la seva potència i flexibilitat.

Figura 2.3.7 Logo Visual Studio

Mixamo: Llibreria d'animacions creada per professionals de descàrrega gratuïta.
Mixamo (Figura 2.3.8) Ofereix una gran varietat d'animacions, poguent carregat un model propi i veure el resultat al moment. Es descarregaran les animacions necessàries per donar vida als personatges.

Figura 2.3.8 Logo Mixamo

Hardware necessari:

Un ordinador gamma mitjana per poder utilitzar el software prèviament mencionat i poder desenvolupar el projecte:

Processador: Intel® Core™ i3-9100
Targeta gràfica: Saphire Radeon RX 570
Memòria: 8 GB RAM

Un telèfon mòbil gamma mitja-baixa per fer les proves del prototip.

2.3.2 Recursos Humans

Com la majoria de videojocs, aquest treball requereix quatre perfils de persona:

Dissenyador principal: Serà l'encarregat de plantejar quines mecàniques es faran servir i com funcionaran. També tindrà la responsabilitat d'estudiar les eines que es necessitaran juntament amb la seva documentació. Finalment s'encarregarà d'elaborar la memòria del projecte.

Programador i tester: Serà l'encarregat de definir com haurà de ser l'algorisme, quines funcionalitats i mètodes haurà de tenir i com s'haurà de dur a terme per aconseguir-les. També tindrà la responsabilitat de comprovar que el joc funciona correctament.

Artista principal: Serà l'encarregat de dissenyar i produir l'art del joc, ja sigui dels escenaris, els personatges, les animacions, etc.

Cap del projecte: Serà l'encarregat de coordinar el projecte seguint la metodologia de treball escollida, ajustar els terminis de les diferents etapes de desenvolupament i aconseguir un bon funcionament en el treball.

Com que aquest projecte ha estat realitzat per una mateixa persona, és encarregat de dur a terme les tasques de tots els perfils, centrant-se més en programador i dissenyador. Pel que fa al perfil de cap de projecte, s'ha dut a terme juntament amb el tutor del projecte.

2.4.3 Econòmica

En aquesta secció definirem el cost econòmic dels recursos tècnics i dels recursos humans explicat en els apartats anteriors.

Pràcticament tot el programari és gratuït. L'única excepció seria amb Substance Painter, el qual s'hauria de pagar una llicència mensual de 19,90 €.

Relacionat a tot el hardware necessari, necessitaríem un ordinador per poder utilitzar el software, així com desenvolupar el videojoc. Així mateix, necessitarem un telèfon mòbil per fer proves del prototip.

Ja disposo d'un ordinador i un telèfon mòbil per desenvolupar i fer proves. Si s'hagués d'adquirir tot de zero, el preu seria semblant al següent:

Ordinador: 500 €
Mòbil: 200 €

Pel que fa als costos dels recursos humans, en cas que es realitzés una hipotètica estimació dels pressupostos dels integrants de l'equip, suposant que hem de contractar un professional de cada perfil principal, els costos per hora serien els següents (en Espanya, segons la pàgina web www.payscale.com/):

Dissenyador de joc: 16 €/hora
Programador: 13 €/hora
Artista: 13 €/hora

Un cop la planificació del projecte feta, es pot calcular el cost total estimat del projecte multiplicant les hores de les tasques de cada professional pel salari mitjà. Però això seria en cas hipotètic, ja que, com s'ha mencionat, el projecte l'ha realitzat una sola persona de manera gratuïta, per tant el cost total dels recursos humans realment és de 0 €.

2.3.4 Conclusions

El projecte es podria considerar viable donada la situació actual, ja que donada les circumstàncies, el cost totes seria de 19,90 € al mes, a causa del software 'Substance Painter'. Ja es disposen de tots els altres recursos necessaris, i tots els elements agafats per internet seran gratuïts i sense copyright.

3 Planificació

Aquesta etapa defineix l'estratègia seguida per arribar als objectius plantejats. Es descriurà breument el pla de treball, les tasques planificades, el temps estimat i els resultats esperats de cada tasca.

3.1 Abast del projecte

Per fer una bona planificació, primer s'ha de tenir molt clar l'abast del projecte. S'ha de tenir ben definits tots els elements que apareixeran en el videojoc, per tal de poder organitzar-se millor, fent una correcta extracció dels requisits del projecte, així com els punts més importants que ha de contenir.

Una vegada analitzat, aquests són els elements indispensables que ha de contenir el videojoc.

1. Gènere 'Endless Survival'.
2. Mecàniques atraients i entretingudes.
3. Menús necessaris per jugar una partida (Inici, Resultats...).
4. Diferents tipus d'enemics.
5. Varietat de trampes utilitzades per eliminar enemics.
6. Diferents personatges amb mecàniques diferenciables.
7. Escenari canviant amb l'objectiu de reduir la monotonia.
8. Mínim d'elements d'artístics, per tal de donar una ambientació.
9. Interfície amb la informació indispensable.

3.2 Paquets de treball

Amb l'objectiu d'organitzar efectivament el projecte, s'han creat una sèrie de tasques necessàries per a la realització. Aquestes tasques s'han agrupat en paquets de treball, disposant d'una duració i una finalitat determinada.

El projecte es comença a desenvolupar a data 8 de febrer del 2021, amb una data final d'11 de juny del 2021. És a dir, es disposa d'una durada d'unes dinou setmanes, les quals hem de dividir els paquets creats.
Els blocs que s'han decidit seguir són els següents:

Documentació

Tasques: Documentar memòria. Documentar codi. Revisar tasques.

Temporització: Dinou setmanes.

Finalitat: Informe del projecte.

Aprenentatge

Tasques: Repassar coneixements del programari (Unity3D, 3DSMax...). Cercar informació de codi o llibreries necessàries.

Temporització: Tres setmanes.

Finalitat: Coneixements base per desenvolupar el projecte.

Disseny

Tasques: Establir abast del projecte, Desenvolupament de la idea. Disseny de les mecàniques.

Temporització: Vuit setmanes.

Finalitat: Base del projecte correctament dissenyada.

Programació

Tasques: Implementar mecàniques, Incorporar interfície. Incorporar textos. Programar base.

Temporització: Dotze setmanes.

Finalitat: Prototip finalitzat.

Estètica

Tasques: Definir estil, Cercar elements artístics. Dissenys i creació dels elements artístics necessaris. Dissenyar interfície.

Temporització: Sis setmanes.

Finalitat: Art majoritàriament complet.

Test

Tasques: Provar el videojoc. Solucionar possibles errors. Balanceig del videojoc.

Temporització: Dues setmanes.

Finalitat: Prototip testejat.

3.3 Cronograma

A partir de la temporalització associada a les tasques planificades, s'ha fet un cronograma (Figura 3.1) on es dóna una visió global de tot el projecte.

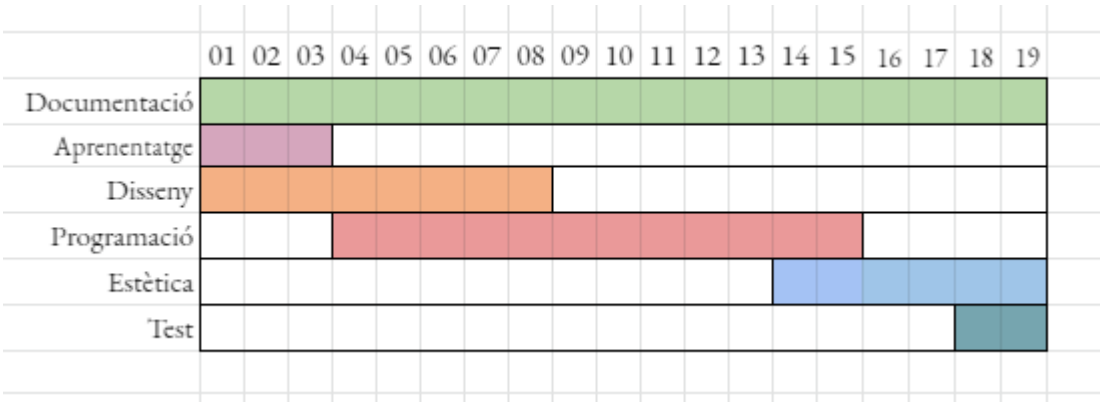


Figura 3.1 Diagrama de Gantt

La idea principal de l'organització, és començar el projecte dedicant un temps a repassar i aprendre les aptituds que es necessitaran al llarg del projecte. Al mateix moment, començar a preparar el disseny del joc, creant les regles que el compondran. Alhora que el projecte agafa forma, es començarà a programar la base, la qual es durà a terme pràcticament durant tot el transcurs del projecte.

Una vegada finalitat el procés de creació, siguent aquest el més important, s'acabarà amb la millora visual del projecte, intentant donar forma a tots els elements anteriorment creats. Per acabar, es faran les proves necessàries per evitar possibles errors, o problemes en el balanceig general. Desde l'inici fins al final, hi haurà un procés de documentació de tot el projecte, per tenir constància en tot moment de l'estat actual, així com donar a conèixer tota la informació rellevant.

4 Marc de treball i conceptes previs

Aquest punt servirà principalment per a preparar al lector i introduir els conceptes necessaris per a poder comprendre de forma correcta i fer un bon seguiment del document de disseny del videojoc.

En l'apartat 2, s'ha explicat en detall la competència actual del videojoc, així com els elements a destacar i comparar. És per això que, en aquest apartat, ens centrarem més en els conceptes per entendre el projecte.

Com s'ha mencionat anteriorment, aquest projecte ha estat desenvolupat per una sola persona, encarregada de tots els aspectes del videojoc. Gràcies al grau, té coneixements avançats de programació, així com algunes capacitats per dissenyar videojocs, modelar i animar models 3D.

El gènere el que ens basem és 'Endless Runner'. Les característiques principals d'aquest gènere, són:

Partides infinites
Una puntuació la qual intentar superar, normalment referint-se a la distància superada.
Mecàniques senzilles, amb poques accions per part del jugador.
Col·leccionables dintre de les partides.

De fer, comparteix moltes característiques amb els jocs Arcade dels 80s o 90s, prioritzant la puntuació obtinguda en jocs sense fi.

La variació d'aquest projecte, auto anomenada 'Endless Surival', agafa aquest concepte i els canvia lleugerament, afegint mecàniques més complexes. Entre aquestes mecàniques es troben:

Diferents personatges amb habilitats úniques, canviant la manera de jugar.
Variïs tipus d'enemics, cada un amb uns moviments o habilitats diferents.
Mecàniques extres, com bonificacions del jugador o
Diferents mapes, generats de manera semialeatòria.

Així mateix, es canvia l'objectiu i la puntuació, substituint la distancia pel nombre d'onades superades, convertint-lo en un 'Survival' més que un 'Runner'. Per tant, es manté la base de les partides infinites fins a perdre. També es mantenen els col·leccionables en mig de la partida, havent-hi monedes per desbloquejar els nous personatges.

5 Disseny del videojoc

5.1 Espai del joc

En els dissenys inicials del projecte, s'apostava per un escenari mitjanament gran, amb diversos camins i zones per explorar. Això estava pensat per centrar-se en l'element d'exploració dintre de cada partida, afegint secrets o premis al llarg de l'escenari. Això provocava partides més lentes i tranquil·les, separant-se bastant de la idea principal.

Al final, es va decidir centrar-se en una jugabilitat ràpida i frenètica, on el jugador quasi no pot parar quiet. Per aquest motiu, l'espai havia de ser més reduït, ocupant tot l'espai de trampes i enemics, per tal d'obligar al jugador a estar atent en tot moment i augmentant la dificultat de cada partida.

Per tal d'evitar la monotonia, s'han creat més d'un escenari, que es tria aleatòriament al principi de la partida. Aquest comparteixen una estructura similar, canviant simplement la forma i l'ambientació. Seran espais tancats amb tots els elements repartits al llarg del mapa, els quals sempre estaran en el mateix lloc. Aquests són: Cofre, Seccions i els límits. Actualment, hi ha implementats dos escenaris diferents (Figura 5.1.1).

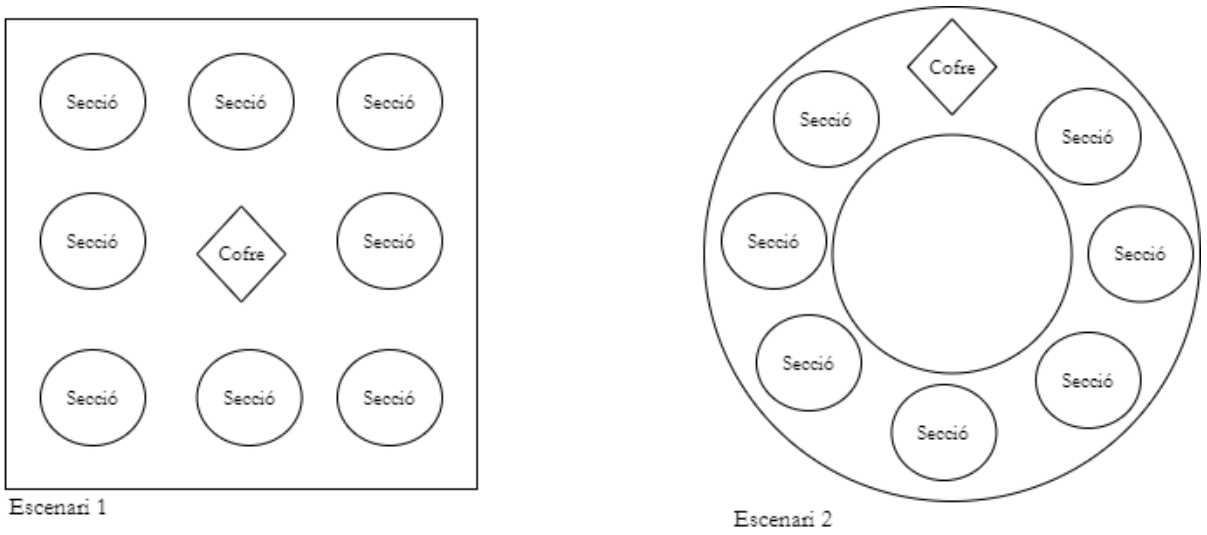


Figura 5.1.1 Escenaris del joc

Cada secció contindrà elements de decoració i algunes trampes. Les trampes poden aparèixer en unes posicions específiques, triant aleatòriament el tipus de trampa i la zona, havent-hi un mínim de dos en cada secció. A més, les seccions apareixen en una rotació aleatòria. Així doncs, tenim diverses versions de cada secció (Figura 5.1.2).

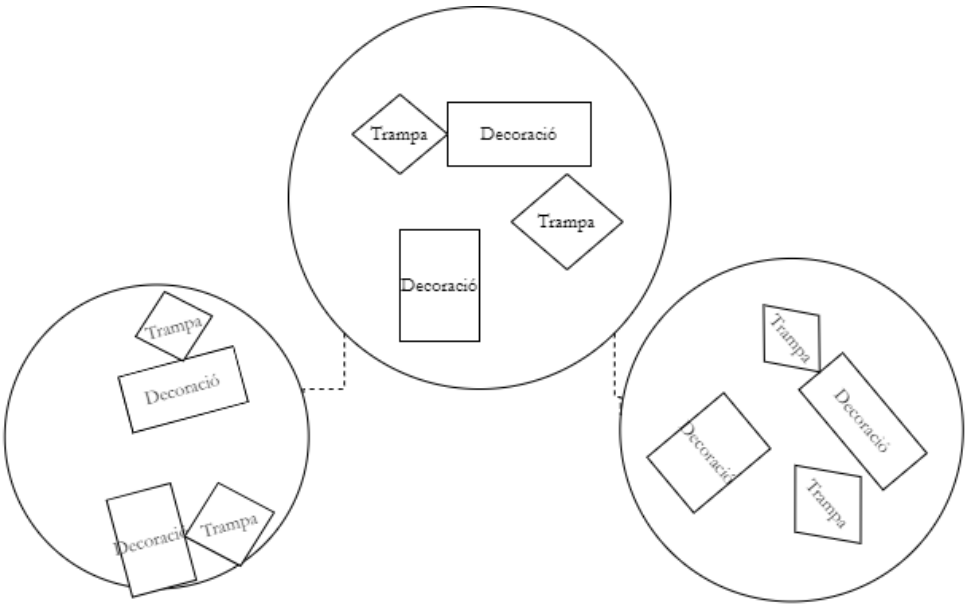


Figura 5.1.2 Possibles version d'una secció

5.2 Mecàniques

5.2.1 Objectes

Hi ha diversos objectes que tenen acció en una partida. Alguns d'ells fan la funció d'administrar la partida, encarregant-se del funcionament de les mecàniques relacionades amb l'escenari, la interfície o d'acabar el joc. Aquests no apareixen físicament en l'espai de joc. Els altres són els propis elements que tenen lloc a l'escenari, com personatges o enemics. Solen tenir un objecte pare en el qual hereten la base, alterant els atributs o funcions a mida.

5.2.2 Accions

Una vegada començada la partida, el jugador podrà interactuar amb el joc amb una sèrie d'accions. A continuació es mostrarà llista d'accions possible, així com els objectes el qual interactuen:

Moure: Joystick
Utilitzar habilitat: ManagerUI. ManagerGame. PlayerController.

5.2.3 Reptes

Encara que l'objectiu principal de joc sigui sobreviure, en la partida apareixen petits reptes que el jugador ha de superar per aconseguir-ho. Alguns d'aquests reptes necessiten altres reptes per ser superats.

A continuació es mostra una llista dels reptes amb les accions i reptes que ha de fer el jugador per superar-los:

Obrir cofre: Moure.
Agafar bonificació: Obrir cofre.
Activar trampa: Moure.
Esquivar enemic: Moure. Agafar bonificació. Activar habilitat.

Eliminar enemic: Activar trampa. Agafar bonificació. Activar habilitat.
Agafar Moneda: Eliminar Enemic.
Sobreviure: Moure. Eliminar Enemic, Esquivar enemic.

Seguidament es mostra un gràfic (Figura 5.2.2) per mostrar el desglossament de reptes.

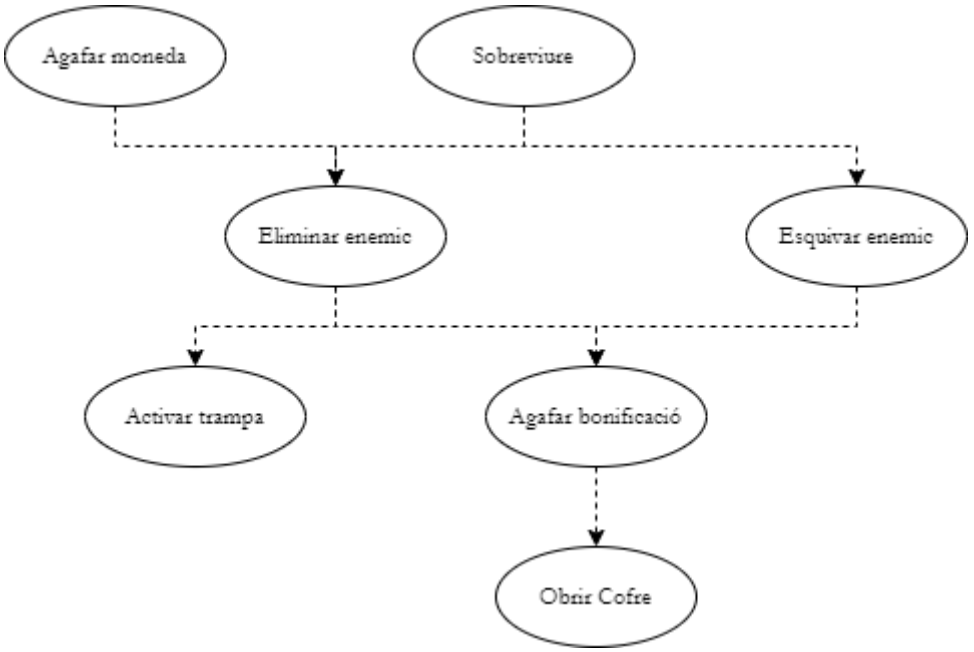


Figura 5.2.2 Diagrama de reptes

5.3 Economia del joc

L'economia és el sistema pel qual recursos i entitats són produïts, consumits, i
Intercanviats en quantitats quantificables. A continuació s'explica com els diferents elements del joc són controlats.

5.3.1 Recursos

El joc compta amb una sèrie de recursos que només serveixen dintre d'una mateixa partida, perdent el valor una vegada acabada. D'altres, es mantenen entre partides i queden guardades en tot moment.

Els recursos són els següents:

Onada: És el número de l'onada actual on es troba el jugador en la partida, augmentant la dificultat a cada increment. El rang de valors és entre 0 i 10.000.

Diners: Els diners aconseguit dintre de la partida, sense cap ús fins a acabar-la. El rang de valors és entre 0 i 10.000.

DinersTotals: Diners totals aconseguits al llarg del joc, utilitzar per desbloquejar tots els personatges. El rang de valors és entre 0 i 10.000.

OnadaRecord: La màxima onada obtinguda en una partida, per tenir un objectiu a superar. El rang de valors que pot tenir és entre 1 i 1000.

Combo: El nombre d'enemics eliminats a la vegada. Millor la recompensa deixada pels monstres. El rang de valors que pot obtenir és entre 1 i 1000.

5.3.2 Entitats

Per ser funcionals, aquest recursos han de ser continguts en entitats que interactuïn amb ells. Existirà una classe encarregada de totes les variables que internen en una partida. Aquestes són l'onada, el combo i els diners. En el cas de les variables externes a la partida, com els diners totals o el record, seran administrats per l'objecte encarregat de guardar i carregar partida.

5.3.3 Funcions

Hi ha diferents tipus de funcions que defineixen com s'interactua amb els recursos prèviament mencionats.

A continuació es mostren les diferents funcions, separades per tipus per veure el funcionament.

Source: Es creen en el món de joc, sense haver estat ja creat.

El número d'onada actual es crea en començar la partida amb valor 1. Aquest va incrementar en passar cert temps o eliminant tots els enemics actuals.

El combo es crea una vegada eliminat mínim a 1 enemic. Aquest incrementa per cada enemic eliminat.

Drain: Desapareixen del joc permanentment degut a la seva consumició.

Quan s'ha eliminat un enemic, passat un cert temps s'elimina el combo actual.

Si quan s'acaba la partida l'onada actual és més petita o igual que l'Onada record, desapareix amb la partida.

Converter: Es transformen en un altre tipus de recurs.

A l'acabar una partida, els diners passen a convertir-se en Diners Totals.

Conjuntament, si l'Onada actual és superior a la OnadaRecord, es transforma en aquesta última.

Trader: S'intercanvien per un altre element del joc.

El jugador pot desbloquejar personatges a canvi dels DinersTotals si supera el preu necessari per cada un.

5.4 Balanceig

5.4.1 Nivells de dificultat

En un joc equilibrat, la dificultat dels reptes presentats al jugador, o bé no hauria de canviar, o hauria d'augmentar, de manera que el jugador senti que els desafiaments posteriors presenten una dificultat més gran que els del principi. Per aconseguir-ho, s'ha de tenir en compte la creixent experiència del jugador i incorporar els augments adequats de dificultat.

El joc només compta amb un nivell de dificultat per defecte, el qual no es pot modificar, jugant cada partida amb les mateixes complicacions. En canvi, en el transcurs de cada partida, hi ha un augment de dificultat per evitar permetre jugar infinitament.

L'objectiu de balancejar el joc és mantenir el jugador en un estat de 'Flow', sense estressar-lo ni avorrir-lo (Figura 5.4.1). Com cada partida és infinita, és tot un repte mantenir-se en aquest estat, ja que no es pot controlar quina dificultat hi haurà a cada onada.

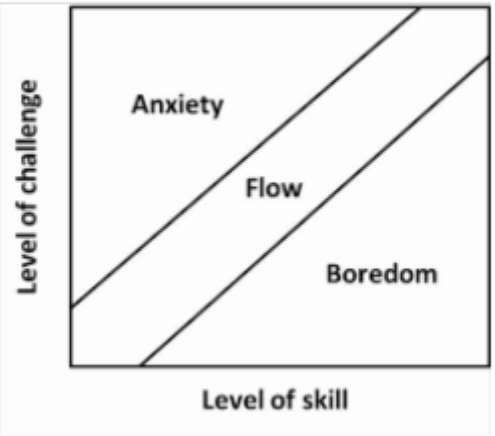


Figura 5.4.1 Diagrama sobre l'estat de 'Flow'.

Per solucionar això, s'han creat uns paràmetres que incrementaran a cada onada.
Aquests paràmetres són:

Tipus d'enemics.
Temps entre cada aparició.
Numero enemics.
Temps límit onada.

Aquests paràmetres augmenten una certa quantitat quan se supera una onada, fins a perdre.

Encara amb això, s'ha volgut afegir una mica més de profunditat per tal d'oferir al jugador una satisfacció en completar un repte complicat. A més, permetre tenir un augment no lineal i que el jugador no acabi en un estat d'avorriment arribat a cert punt de la partida.

S'han agrupat les onades en grups de 5, separant-los en nivells de dificultat. Dintre de cada grup. incrementaran els paràmetres prèviament explicats fins en passar del nivell 1 al nivell 5. Aquesta última onada representarà el repte més complicat dintre del grup i posarà al jugador en dificultats. Una vegada superat, els paràmetres es reiniciaran al nivell 1, començant el següent grup d'onades.

Fent això, el jugador entraria en un bucle, on una vegada superat el primer grup, els següents no seran cap repte. Per evitar això, els valors no es reinicien per complet, sinó que s'incrementen, tenint el nivell 1 una dificultat similar al nivell 3 del grup anterior. A més, s'afegeix la velocitat dels enemics com a paràmetre que incrementa constantment, sense reiniciar-se. Amb aquest afegit es vol donar al jugador una satisfacció i un petit descans cada vegada que supera el màxim nivell d'un grup, fent-lo sentit poderós en afrontar un nivell inferior (Figura 5.4.2).

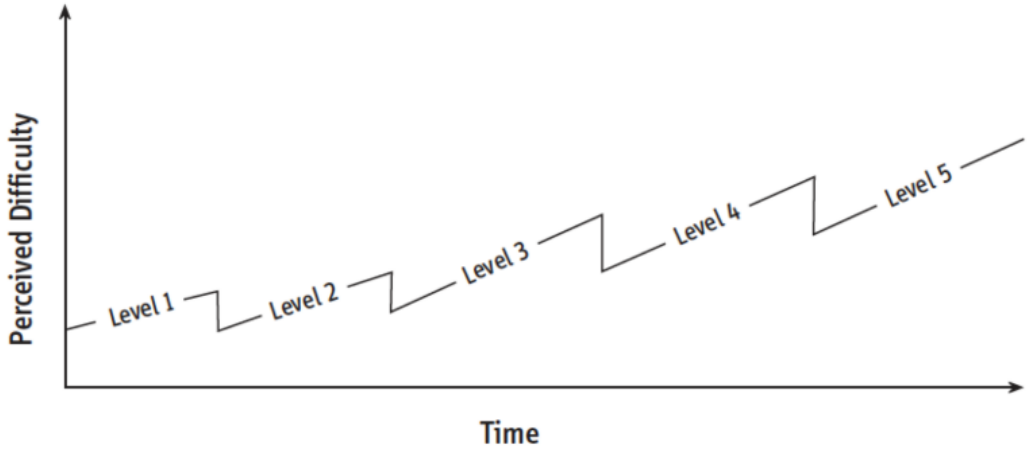


Figura 5.4.2 Diagrama de la dificultat variant.

A part del canvi de paràmetres, també s'incrementa la dificultat amb diferents enemics que apareixen arribant a cert grup d'onades. Aquest tindran unes habilitats que dificultaran la supervivència del jugador.

5.4.2 Feedback positiu

El feedback positiu dona lloc quan el jugador supera un repte que causa un canvi d'estat en el joc per fer el següent repte més fàcil, així fent menys difícil completar-lo. És una manera d'incitar al jugador a completar aquests reptes, a més de fer-li sentit que els seus esforços tenen una recompensa.

En el joc s'ha afegit amb un cofre que apareix al principi de la partida al costat del jugador. Aquest cofre dóna al jugador unes bonificacions que facilitaran la supervivència, així com l'eliminació d'enemics. La recompensa dependrà del nombre de monstres eliminats mentre el cofre estava tancat. Llavors, si el jugador

aguanta més temps sense utilitzar-lo, les recompenses seran més grans que si ho obre amb un mínim d'eliminats.

En un altre cas, s'aplica en el moment d'eliminar grans quantitats d'enemics. Quan més enemics s'eliminin a la vegada, s'incrementa la recompensa deixada en morir, incrementant la quantitat de diners obtinguda. Si el jugador s'arrisca i acumula enemics, obtindrà una recompensa major. En canvi, si elimina d'1 en 1, la recompensa serà sempre la mateixa. Amb aquests diners pot aconseguir més personatges, els quals tenen altres habilitats i jugabilitat.

5.4.3 Shadow cost

Quan es parla de 'Shadow cost', es refereix el fet d'integrar certs avantatges i desavantatges a diferents elements amb l'objectiu de balancejar el joc, fent que l'opció A sigui igual de bona que l'opció B. Amb això s'evita una opció guanyadora, permeten les mateixes possibilitats de completar el joc amb qualsevol de les opcions.

En el joc, hi han diversos personatges amb diferents habilitats i mecàniques que fan variar lleugerament la jugabilitat. Per evitar el problema anteriorment explicat, s'han creat uns desavantatges que cada personatge té en comparació dels altres. Encara després d'haver obtingut tots els personatges, el primer personatge té el mateix valor total que l'últim.
A continuació es mostra la llista de personatges amb el seu 'Shadow cost':

Maga

+ Permet fer un impuls per esquivar els enemics.
- No té cap altre habilitat. És el personatge més equilibrat.

Guerrer:

+ Permet tenir un escut per ser invulnerable a les trampes.
- La seva velocitat disminueix severament quan el porta.

Bufóna:

+ És el personatge més ràpid del joc.
- No pot quedar-se quiet, obligant a estar alerta en tot moment.

Rei:

+ Apareix un cofre que dona grans quantitats de diners.
- El cofre dels poders desapareix, prohibint utilitzar-los.

Golem:

+ Pot activar una habilitat atacar els enemics quan l'energia està al màxim.
- Quan no té energia disminueix bastant la seva velocitat, havent de carregar-la en un lloc concret.

Enterrador:

+ Pot transformar les plantes de l'escenari en ajudants que el protegeixen.
- Apareixen plantes que inhabiliten les trampes.

Amb les trampes que apareixen en el mapa, apliquem un sistema similar. Cada trampa té una àrea de dany diferent, eliminant més enemics a la vegada amb les més extenses. Per evitar l'abús d'aquestes, tarden més a activar-se, així com en recarregar-se per poder utilitzar-les de nou. El jugador podrà triar entre trampes més lentes i grans, o ràpides i petites.

5.5 Interfícies

Es compta amb una sèrie d'interfícies per indicar al jugador l'estat de la partida, conjuntament amb botons per permetre interactuar amb el joc

A continuació es mostren les diferents pantalles que apareixen:

Per començar tenim la pantalla inicial (Figura 5.5.1). Aquesta compta amb un escenari introductori amb un missatge de fer clic a la pantalla. Conjuntament apareix el nom del videojoc. En un futur, inclourà el logotip.

Figura 5.5.1 Pantalla inicial

Quan es fa clic a la pantalla, s'obre la pantalla de selecció de personatge (Figura 5.5.2). A dalt a l'esquerra es mostra els diners aconseguits actualment. A les cantonades apareixen fletxes per canviar de personatge, i a la part inferior central, un botó per jugar.

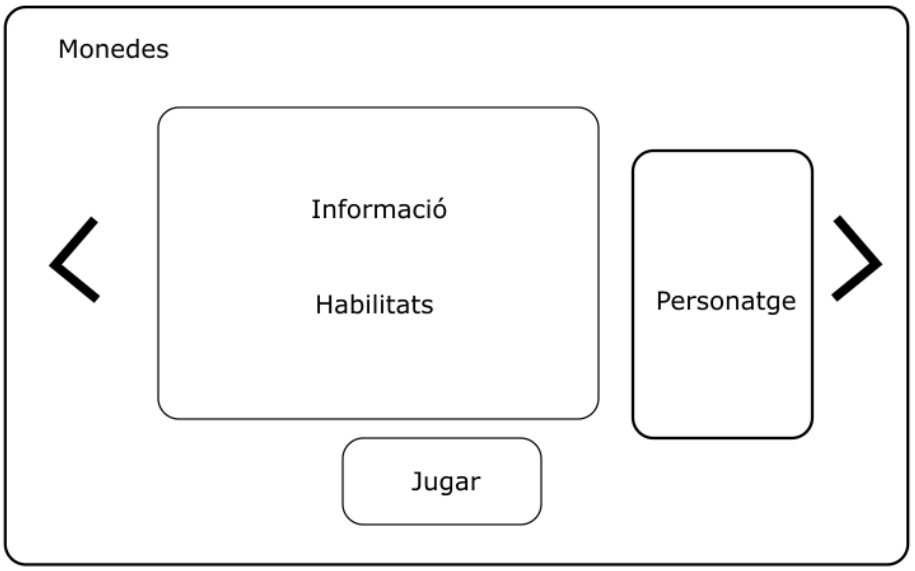


Figura 5.5.2 Selecció de personatge

A l'esquerra apareix la informació del personatge, amb una petita descripció i les habilitats les quals disposa. Clicant sobre aquestes habilitats, apareix la informació per sobre de tota la interfície (Figura 5.5.3). Tocant qualsevol part de la pantalla es torna a tancar.

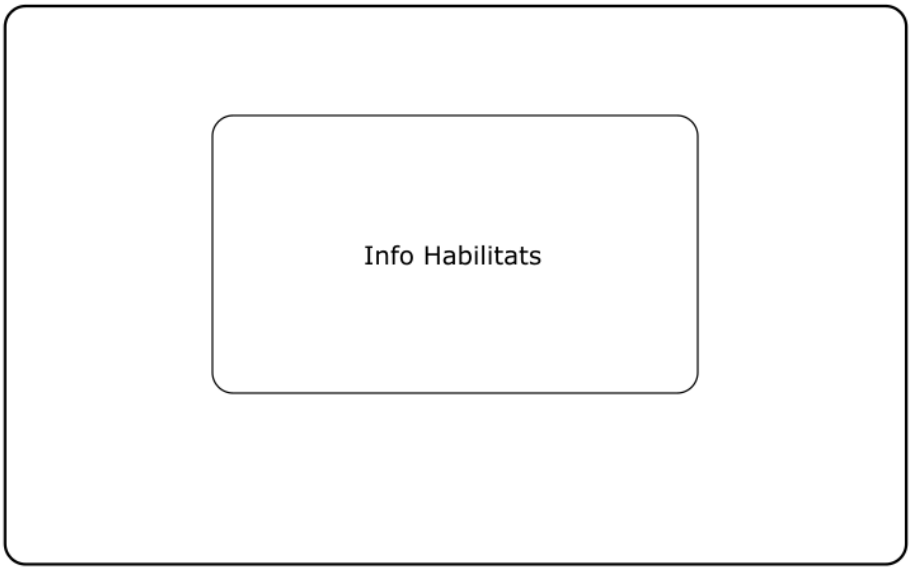


Figura 5.5.3 Informació habilitats

Si un personatge està bloquejat, apareixerà de color negre (Figura 5.5.4). A més, en comptes de les habilitats apareixeran els diners necessaris per comprar-lo, així com el boto fer-ho. Mentre estigui bloquejat, el botó de jugar queda inhabilitat, mostrant-se de color gris.

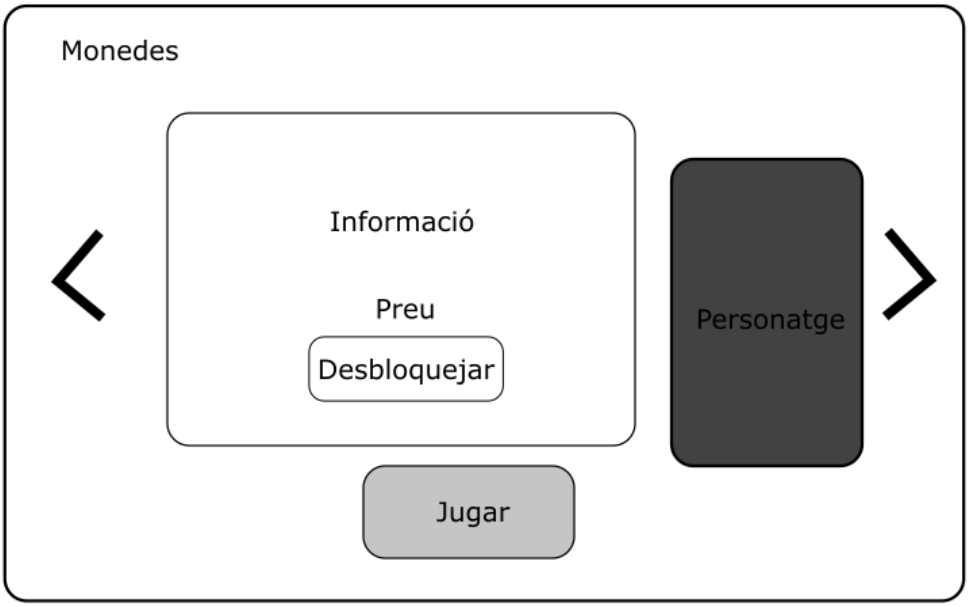


Figura 5.5.4 Personatge bloquejat

Una vegada en partida, a la part superior apareix tota la informació important, així com el botó per activar la pausa (Figura 5.5.5). A la part inferior, surten els botons per interactuar amb el personatge, des del 'Joystick' per moure fins al botó per activar l'habilitat. Quan es mata un enemic, a la part esquerra apareix el nombre d'enemics eliminats a la vegada

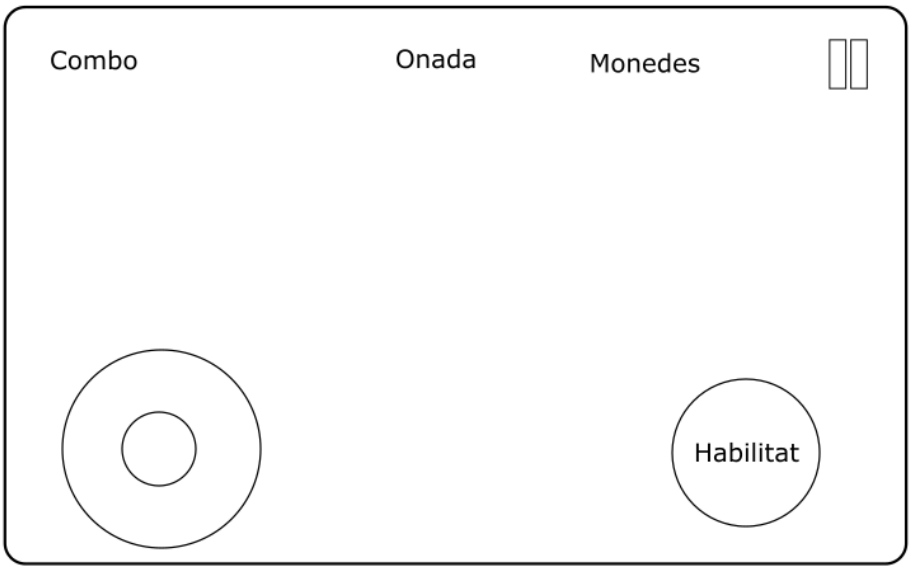


Figura 5.5.5 Interfície en partida

Hi ha alguns elements d'informació que es mostren en el món del joc. Un d'ells és el nombre d'enemics eliminats. Aquest apareix a la part frontal del cofre (Figura 5.5.6) i desapareix quan està obert, ja que no suma quan està en aquest estat.

Figura 5.5.6 Informació eliminats

Per alguns objectes de món s'ha afegit una línia de color al voltant del model. Aquesta canvia de color depenent l'estat. En el cas de les trampes, canvia a color vermell quan comença a atacar, o gris quan està recarregant (Figura 5.5.7). Pel cofre, canvia depenent del nivell de recompensa actual (Figura 5.5.8).

Figura 5.5.7 Trampa atacant

Figura 5.5.8 Cofre nivell 2

En el menú de pausa (Figura 5.5.9) apareix l'onada i monedes actuals, així com els botons per tornar a començar, triar personatge o continuar la partida.

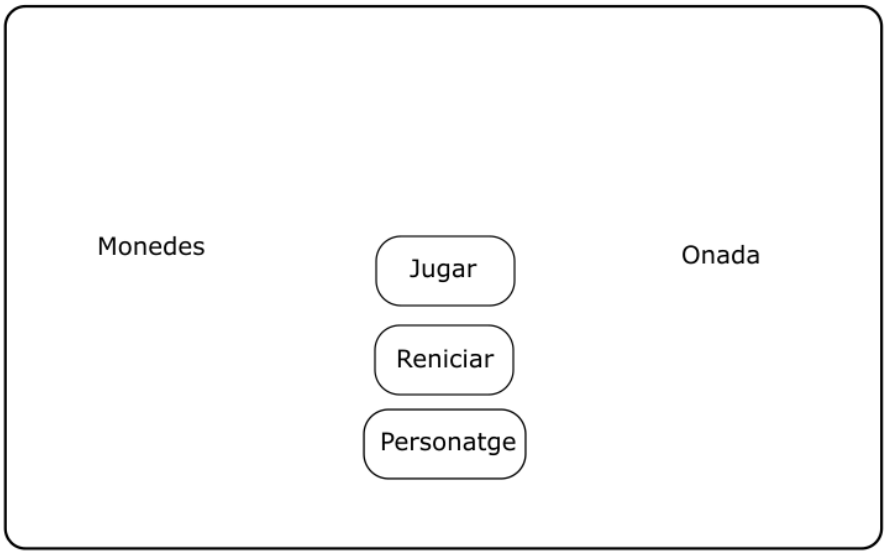


Figura 5.5.9 Menú pausa

En acabar la partida apareixen els resultats obtinguts (Figura 5.5.10), a més de la millor puntuació i el total de monedes. Es permet tornar a jugar o canviar de personatge.

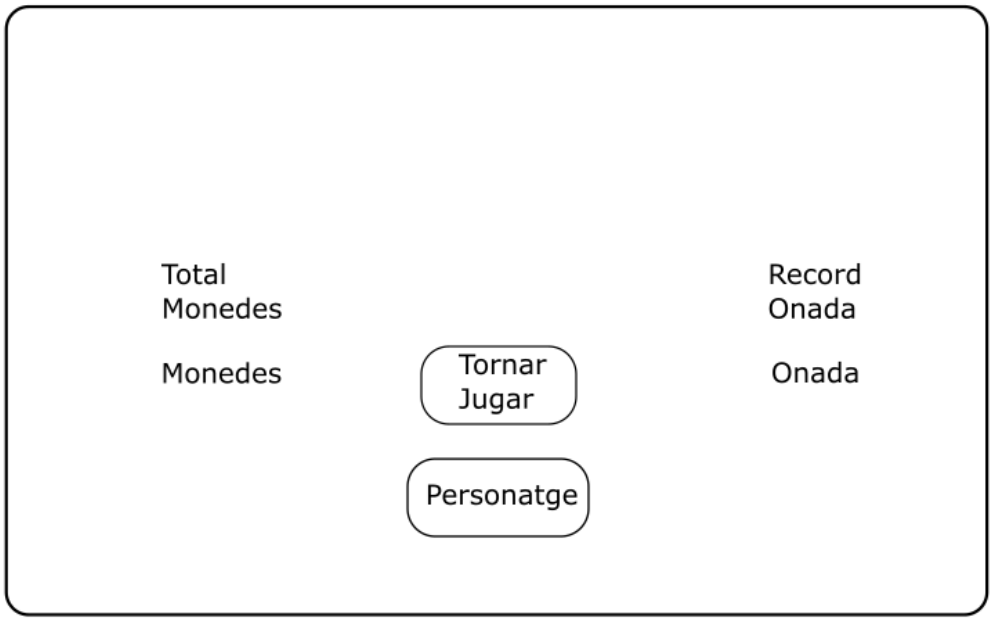


Figura 5.5.10 Menú fi de partida

5.6 Game layout chart

El joc segueix una estructura molt simple des que es comença a jugar fins a acabar la partida (Figura 5.6.1). Com que no hi ha un final, en acabar la partida es torna a jugar la següent o es tria un personatge diferent. La pantalla inicial només es mostra quan s'obre el joc.

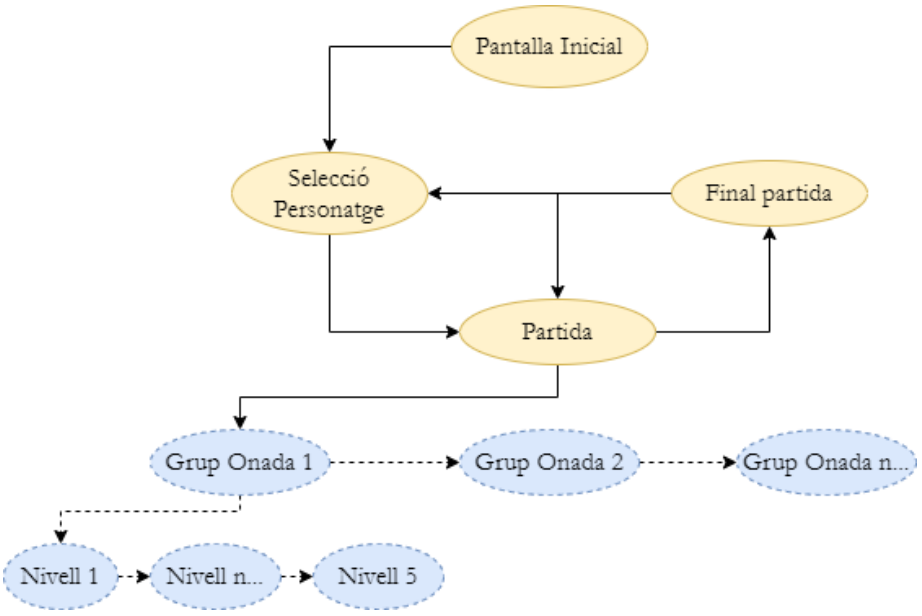


Figura 5.6.1 Estructura partida

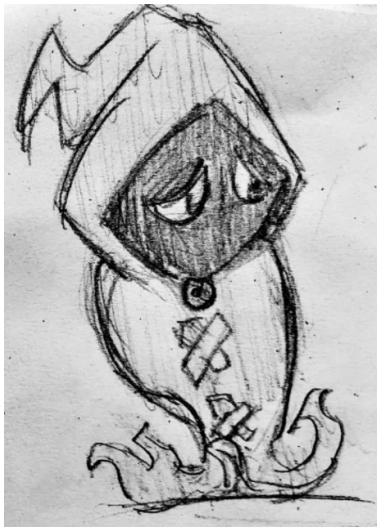
Es pot observar una estructura més detallada en el transcurs d'una partida. Es repeteixen els grups fins que el jugador és derrotat. El sistema de grup d'onades és explicat amb més detall al punt '5.4.1 Nivells de dificultat'.

5.7 Personatges

Els personatges del joc són els que més tenen un pes narratiu en la història, ja que són els únics als qui s'explica la seva història. A més, són els encarregats de fer que el jugador continuï jugant, ja que les monedes aconseguides són per desbloquejar-los. És per això que han de tenir una personalitat i estils diferenciables, que puguin tenir un pes en la jugabilitat. Tots els personatges contenen una habilitat que influeix en la jugabilitat. Aquesta està definida per la seva història, fent més forta la connexió entre les mecàniques i la narrativa. A més, algun conté vestimenta o objectes que encara el relacionen més amb l'habilitat.

A continuació analitzarem els diferents personatges:

Maga:
La maga és un personatge covard, tímid i malastruc. Per representar això, no es mostra res del seu aspecte, totalment coberta per una túnica de mag. Amb els ulls mostra por o preocupació, amb una postura reservada (Taula 5.1).

	Descripció física	Edat desconeguda Altura petita
	Descripció caràcter	Tímida Covard Trista
	Avantatges	Ràpida Àgil
	Desavantatges	Por constant Simple

Taula 5.1 Informació Maga

La seva història explica que va estudiar en una acadèmia de mags. Va aprendre com escapar del perill amb portals, però just el dia que explicaven els poders d'atacar, es va posar malalta. És per això que la seva única habilitat és activar un portal per fugir.

Els colors que el representen són el blau i el lila, representant màgia i místic.

Guerrer:

El guerrer és despistat, responsable i protector. Es mostra totalment tapat per l'armadura, amb una postura recta, preparat per enfrontar el perill (Taula 5.2).

	Descripció física	Edat desconeguda Altura mitjana
	Descripció caràcter	Despistat Protector
	Avantatges	Resistent Responsable
	Desavantatges	Lent. Oblidadís

Taula 5.2 Informació Guerrer

Segons la seva història, sempre perd l'espasa abans d'un combat. És per això que només porta un escut en mà, sense possibilitat d'atacar. La seva habilitat permet equipar o desequipar l'escut, protegint-se quan és necessari.

Els colors representatius són el gris, per l'armadura, i el color blau, indicant estabilitat i confiança.

Bufona:

La bufona és molt inquieta i impacient, la hiperactivitat en persona. A més, és extremament bromista. És el personatge més alt de tots. Té una postura informal, amb una animació de voler començar a córrer immediatament (Taula 5.3).

	Descripció física	Edat desconeguda Altura gran
	Descripció caràcter	Impacient Divertida
	Avantatges	Molt ràpida Inquieta
	Desavantatges	Irresponsable Bromista

Taula 5.3 Informació Bufona

Desde petita que va aprendre anar en monocicle abans d'aprendre a caminar. Desde llavors, sempre va a tot arreu en ell, i una vegada es comença a moure, l'adrenalina li impedeix frenar. Per això, és el personatge que té més velocitat, amb l'inconvenient de no poder estar quiet mai.

A causa de la seva altura i al estar sobre un monocicle, pot veure més enllà que els altres personatges. Per indicar la seva part bromista, la seva habilitat principal permet eliminar els enemics amb el foc que deixa quan va a molta velocitat, fent que ells mateixos es matin.

Els colors que la representen són vius i animat, sent el vermell i el blanc els principals, indicant els colors clàssics del circ.

Rei:
El rei és avariciós i orgullós, capaç de qualsevol cosa per aconseguir diners. Abusa del seu poder com a rei per aprofitar-se dels altres. Això si, ningú dubte de la seva valentia, enfrontant-se a qualsevol perill si hi ha benefici pel mig. És un personatge gran d'amplada però d'altura mitjana. Té una postura ferma i una mirada de superioritat (Taula 5.4).

	Descripció física	Edat Gran Altura mitjana
	Descripció caràcter	Avariciós Orgullós
	Avantatges	Valent
	Desavantatges	Egoista Abusador

Taula 5.4 Informació Rei

L'únic que se sap d'ell és la seva avarícia pels diners. És per això que la mecànica especial és canviar el cofre de bonificacions per un cofre de diners, convertint-lo en el personatge ideal per aconseguir monedes. A més, la seva habilitat principal és bàsicament pagar per guanyar, gastant les monedes aconseguides per eliminar els enemics.

Els colors principals seran el lila, blau i vermell, colors representatius de la reialesa.

Golem:

El golem és un ésser màgic amb l'objectiu de protegir el bosc. No es mostra cap gènere ni edat, al ser format per pedres i màgia. És un personatge pacifista i innocent, però molt fort en combat. Té una postura tranquil·la, sense estar en alerta, indicant que no hi ha cap indici d'agressivitat (Taula 5.5).

	Descripció física	Edat desconeguda Altura petita
	Descripció caràcter	Pacifista Protector
	Avantatges	Fort Resistent
	Desavantatges	Lent Innocent

Taula 5.5 Informació Golem

Ningú té cap informació respecte a desde quan existeix, ni que com va ser creat. Només que, mentre no s'ataqui el bosc, es mantindrà tranquil i no molestarà. Per això, és l'únic personatge que pot atacar directament els enemics que ataquen. Com és un ésser màgic, funciona amb una energia que ha de carregar en unes pedres màgiques repartides pel mapa.

Els colors representatius són el gris de la pedra, el blanc de l'energia (indicant puresa) i verd de l'herba que creix del seu cos (indicant naturalesa i calma).

Enterrador:

L'enterrador és un personatge corpulent i intimidant. Se centra en la seva feina i no es deixa distreure amb facilitat, però està fart de treballar. A més, va cobert amb roba llarga i caputxa, ocultant la seva cara. Té una postura encorbada i cansada, indicant que no té ganes ni de caminar.

	Descripció física	Edat desconeguda. Altura gran.
	Descripció caràcter	Intimidant Callat
	Avantatges	Adaptable Trempat
	Desavantatges	Mandrós Poca paciència

Taula 5.6 Informació Enterrador

L'enterrador ha estat sempre l'encarregat de cuidar del cementiri. Desde treure les males herbes, fins a evitar que els morts dominin la zona. Tants anys treballant amb la mort ha fet que sigui molt proper a ella. La seva habilitat fa que els morts que elimini facin créixer mala herba allà on moren. L'enterrador pot eliminar totes les plantes per invocar protectors que l'ajuden a eliminar els enemics.

5.8 Narrativa

El nivell de narrativa del joc és molt baixa, ja que no compta amb cap història, centrant-se simplement en la jugabilitat. L'única explicació de la història del món, està en la informació de cada personatge que apareix en la selecció i en la mateixa ambientació.

Cada un dels personatges apareix amb una petita descripció que el defineix, així com explicar el que el fa especial. A més, cada habilitat te a veure amb un aspecte seu, augmentant la concordança entre estètica i narrativa.

Tots els elements referents a l'ambient expliquen el perquè del joc. El jugador podrà observar que apareix en un bosc o cementiri en el qual apareixen tots tipus de monstres disposats a eliminar-lo. La raó per la qual es troba en aquell bosc, està a la imaginació de cada jugador.

5.9 Ambients

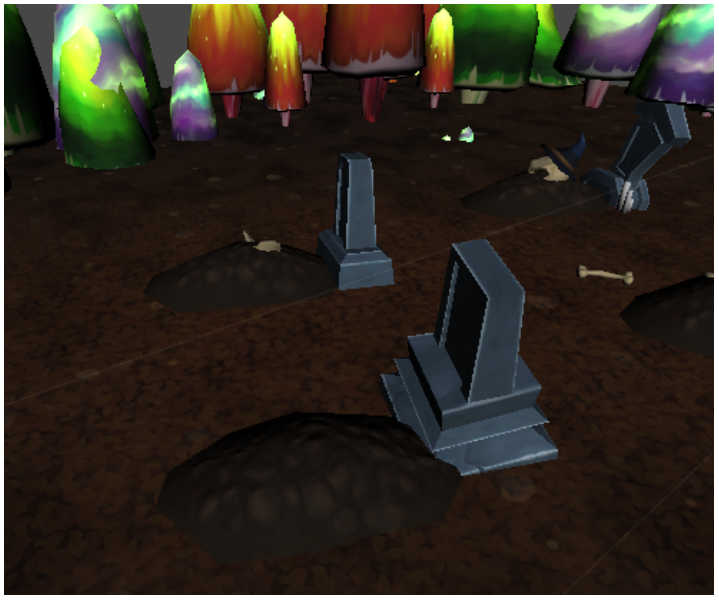
Per aconseguir endinsar al jugador en el món del joc, cal crear una ambientació adient a la temàtica. Tots els elements han de tenir una concordança per tal de combinar amb la narrativa. Per fer això, necessitem tenir en comptes diferents punts a l'hora de centrar-se en l'ambient de joc.

L'ambientació del joc ha de ser de temàtica Halloween, però sense arribar a fer por. Ha de ser una combinació entre elements de terror amb una temàtica amigable i graciosa. Aquest estil sol ser anomenat com 'Spooky', referint-se a una sensació de calfred més que de por.

Tots els objectes de decoració que apareixen tenen a veure amb aquesta temàtica (Figura 5.9.1). (Calaveres, carabasses, barret i escombres de bruixa...). Aquests elements estan repartits per l'escenari i a la pantalla inicial del joc.

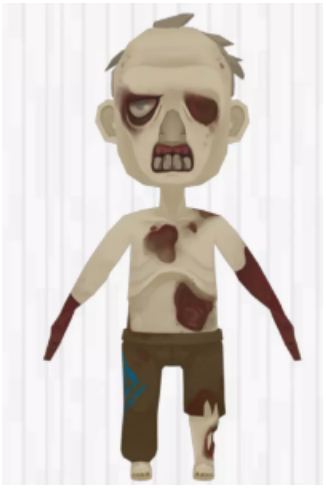
(Figura 5.9.1) Decoració 'Spooky'

La partida es desenvolupa en un cementiri enmig del bosc. Per això, tots els elements d'escenaris han d'estar relacionats (Bolets, arbres, tombes...). A més, cada escenari es jugarà en diferents parts del bosc (Figura 5.9.2). El primer nivell serà una zona plena de bolets, amb un estil més colorit. El segon mapa tindrà com a toc més fosc i amb temàtica de carabasses, afegint arbres secs en comptes de bolets.



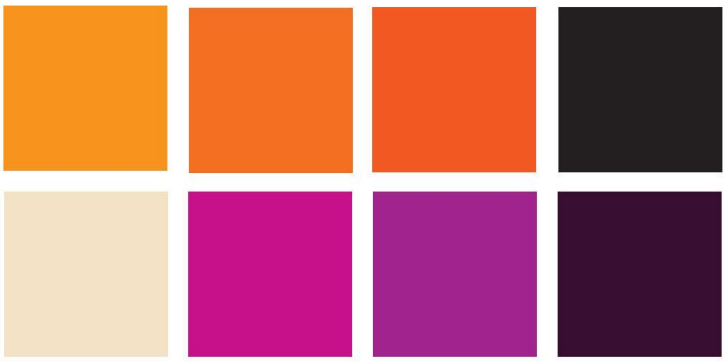
(Figura 5.9.2) Escenari de cementiri en un bosc

Tots els tipus d'enemics seran monstres clàssics de terror, com zombis, ratpenats o esquelets (Figura 5.9.3), però sense fer por, tenint una estètica més aviat infantil.



(Figura 5.9.3) Zombi d'exemple

Els colors representatius del joc, seran els que representen la festivitat de Halloween (Figura 5.9.4). Solen ser tons de taronges (Carabasses) i liles (Nit).



(Figura 5.9.4) Colors representatius de Halloween

Llavors, la llum de l'escena simularà ser de nit amb uns tocs de lila. Per contrastar, la interfície variarà per tons de taronja.

Per acabar de retocar els efectes visuals, s'afegiran unes partícules en alguns elements del joc (Figura 5.9.5). Aquests efectes seran del mateix tema i colors que les anteriors.

(Figura 5.9.5) Partícules

Les diferents músiques que apareguin agafaran les principals característiques de la música 'Spooky'. Combinarà tons greus que marcaran el ritme amb instruments aguts per crear la melodia. En el menú inicial, sonarà una música més lenta i misteriosa, mentre que en partida sonarà una més animada i ràpida.

5.10 Elements a desenvolupar

A continuació es mostra una llista amb els elements que s'han de desenvolupar per aconseguir el disseny explicat anteriorment:

Elements d'interfície:

- Informació personatges.
- Habilitats.
- Botons.
- Tipografia.
- Marcs.

Entorns:

- Elements d'escenari.
- Textures.
- Partícules.
- Efectes Llum.

Personatges:

- Models 3D.
- Textures.

Animació:

- Animacions de càmera.
- Animacions d'interfície.
- Animacions dels personatges i enemics.
- Animacions de les trampes.

Música i efectes sonors:

- Música Inicial.
- Música Partida.
- So dels personatges i enemics.
- So de les trampes.
- So interfície.
- Efectes sonors ambientació.

6 Implementació i proves

És aquest apartat s'explica com s'han implementat les idees dissenyades, així com els problemes trobats i com s'han solucionat. Es faran un repàs de tot el codi creat i com s'ha creat i implementat tot el tema artístic. Per acabar, s'explicaran les proves que s'han fet per comprovar que tot funciona com es pensava.

6.1 Estructura de classes

A continuació es mostra un gràfic amb les classes i objectes que intervenen en una partida (Figura 6.1.1).

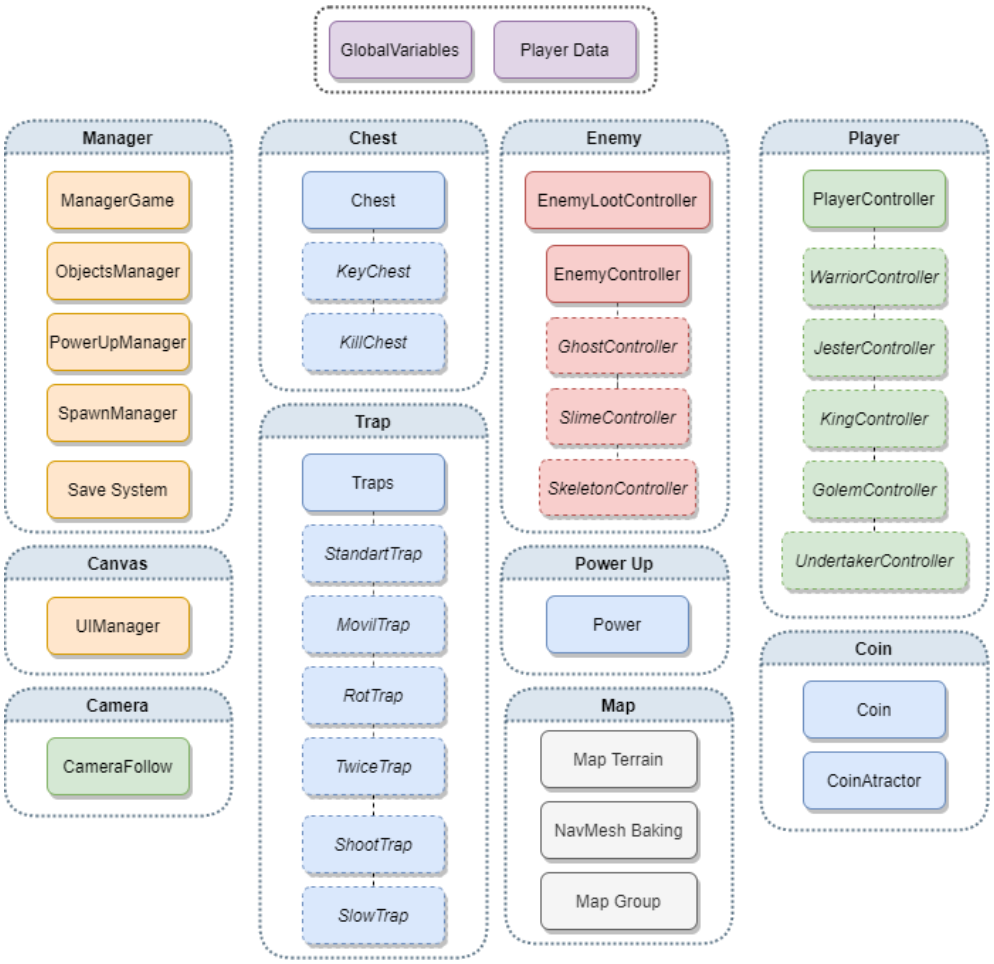


Figura 6.1.1 Gràfic d'objectes en partida

També tenim les classes que interactuen en el menú inicial i de selecció de personatge (Figura 6.1.2).

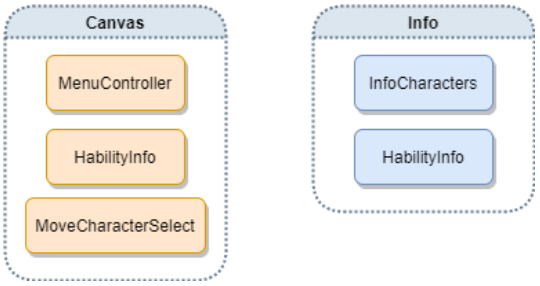


Figura 6.1.2 Gràfic d'objectes en el menú inicial

6.2 Dades joc (PlayerData, GlobalVariables)

S'han creat dues classes que serveixen només per contenir dades importants del joc.

Per tal de tenir més claredat en el codi, s'ha creat unes variables Enum (Figura 6.2.1) a la classe GlobalVariables, assignant a les variables classes de 'Personatge' i 'PowerUp' un nom el qual és comú per totes les classes.

```
public enum CharactersTypes { Mage, Warrior, Jester, King, Gollem, Undertaker};
public enum Powers {None, Fast, FastTraps, EnemyStun, EnemyDeath, Shield};
```

Figura 6.2.1 Variables Enum de GlobalVariables

Així, en comptes de comprovar si un personatge és el número 3, es pot comprovar mirant si és el tipus Mag (Figura 6.2.2). Amb això, es facilita molt l'escriptura i lectura del codi, fent-ho més comprensible i evitant recordar que vol dir cada número.

```
switch(classTried)
{
    case GlobalVariables.CharactersTypes.Jester:
        cameraScript.IncreaseSize(extraCameraSize);
        break;
}
```

Figura 6.2.2 Exemple ús de les variables

La classe PlayerData conté totes les dades que es volen guardar una vegada es tanca el joc (Figura 6.2.3). Des dels personatges desbloquejats, fins a les monedes actuals. Les dades es carreguen i guarden amb una classe auxiliar explicada més endavant.

```
public bool[] characterUnlock;
public int coins;
public int maxWave;

public PlayerData(bool[] _characterUnlock, int _coins = 0, int _maxWave = 0)
{
    coins = _coins;
    maxWave = _maxWave;
    characterUnlock = _characterUnlock;
}
```

Figura 6.2.3 Variables i constructor de PlayerData

6.3 Menú Inicial

El menú inicial consta de dues pantalles, una amb el títol del joc, i l'altre per seleccionar personatge. Aquestes dos estan integrades a la mateixa escena, la qual canvia d'un a l'altre amb una animació de càmera. Com és la primera imatge que veu el jugador, ha de ser visualment agradable i atraient.

6.3.1 Escena

A l'escena hi ha preparat un petit escenari amb els elements justos que sortiran per pantalla i que donaran l'ambientació necessària (Figura 6.3.1). Amb els efectes i angles de la càmera, no es notarà la falta de terra o objectes.

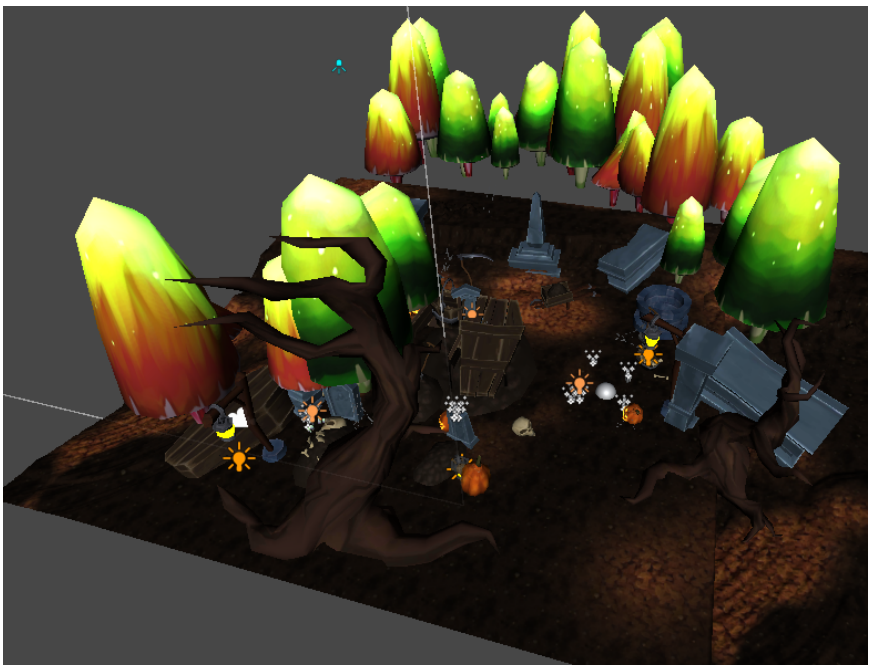


Figura 6.3.1 Escena inicial

La càmera principal està col·locada entremig d'aquest escenari (Figura 6.3.2) i està posicionada en un angle on es pot veure el personatge de fons.

Figura 6.3.2 Posició càmera inicial

Una vegada col·locat apliquem les llums necessàries i tenim el resultat final (Figura 6.3.3)

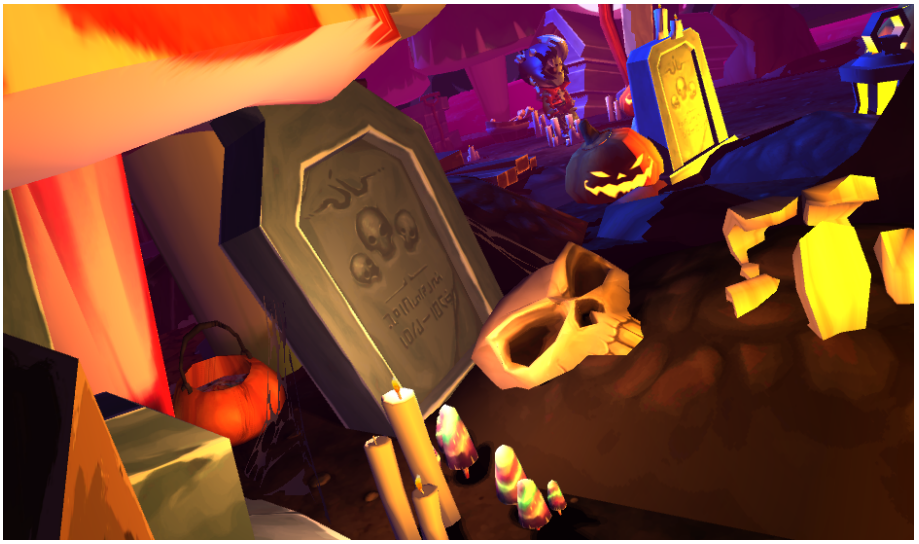


Figura 6.3.3 Efectes de llum

En el fons, hi ha un Game Object que serà la posició on apareixeran els personatges.

A l'escena hi ha un 'Canvas' per mostrar la informació (Figura 6.3.4). Està col·locat en el món del joc perquè sembli que els elements de l'escenari estan davant. A més, aplica una foscor per centrar l'atenció en el personatge.

Figura 6.3.4 Panell d'informació

6.2.2 Càmeres

La càmera principal del joc compta amb una animació (Figura 6.3.5) per canviar de menú. Aquesta es mou per tot l'escenari per situar-se davant del personatge (Figura 6.3.6), sempre evitant apuntar a una zona buida.

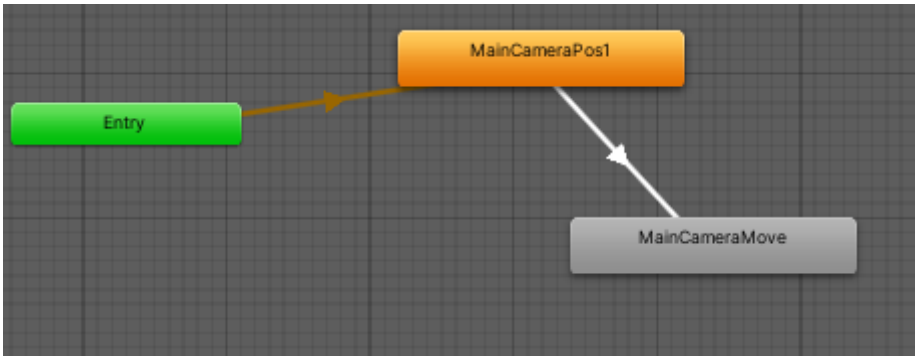


Figura 6.3.5 Animació càmera inicial.

Figura 6.3.6 Posició càmera selecció

A més, hi ha una segona càmera col·locada directament a la posició de triar personatge. Aquesta s'activarà quan s'hagi de jugat una partida desde que s'ha començat a jugar. Això es fa per no haver de veure l'animació de càmera quan es torna a triar personatge.

Per saber quina càmera activar es guarda una variable tipus 'Bool' en la classe PlayerData comentada anteriorment. Aquesta es posa a 'true' quan una partida comença, activant la segona càmera quan s'obre l'escena. En obrir el joc, es comprova aquesta variable abans de carregar les dades guardades, siguent 'false' per defecte i activant la primera càmera.

6.3.3 Personatges

Tot el codi dels menús i la selecció de personatge està inclosa un mateix script anomenat 'MenuController'.

Només començar la partida, llegeix mitjançant 'PlayerPrefs', una forma de guardar dades fàcilment, per saber quin és l'últim personatge amb el qual s'ha jugat. Si és la primera vegada que es tria el primer.

Quan ja se sap quin personatge ha sigut triat, es crea a l'escena amb la funció "Spawn" (Figura 6.3.7).
Aquesta elimina el personatge creat i les habilitats mostrades i seguidament crea el personatge i bloc d'informació indicats per una variable índex. Elimina l'script de moviment i el fa petit perquè quedi bé a l'escena. També afegeix un script per permetre moure el model arrossegant la pantalla del mòbil.

```
void Spawn()
{
    if (playerSpawned != null) Destroy(playerSpawned);
    if (habilitySpawned != null) Destroy(habilitySpawned);

    playerSpawned = Instantiate(chars[index], spawnChar.position, spawnChar.rotation);
    playerSpawned.transform.localScale = new Vector3(0.7f, 0.7f, 0.7f);
    Destroy(playerSpawned.GetComponent<PlayerController>());

    playerSpawned.AddComponent<MoveCharacterSelect>();

    if (!SaveSystem.characterUnlock[index]) playerSpawned.GetComponent<PlayerController>().Locked();

    DisableStart(!SaveSystem.characterUnlock[index]);
    EnableBuy(!SaveSystem.characterUnlock[index]);

    info.SetInfo(index);
}
```

Figura 6.3.7 Funció 'Spawn'.

Després, crida les funcions per saber si està desbloquejat i si es pot comprar.
Per mostrar la informació corresponent al personatge triat, un script anomenat 'InfoPersonatges' guarda el títol i la descripció, i s'encarrega de col·locar-los en pantalla (Figura 6.3.8).

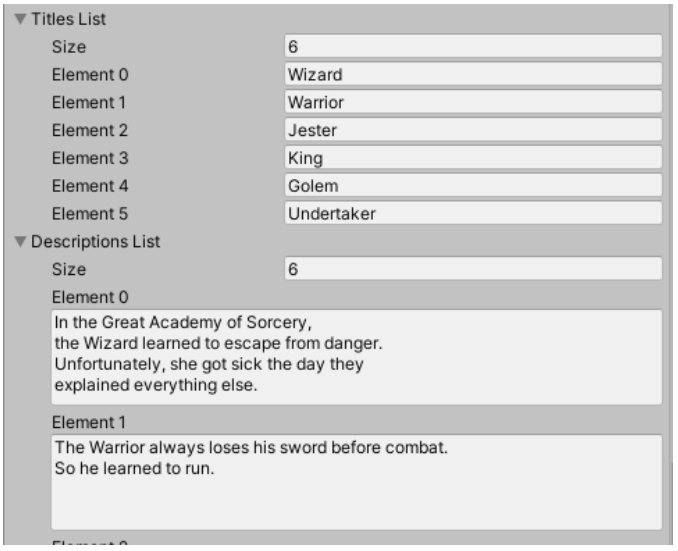


Figura 6.3.7 Variables 'HabilityInfo'

La funció 'EnableBuy' (Figura 6.3.8) activa els botons i textos necessaris si el personatge està bloquejat. A més, comprova si el jugador té els diners necessaris per comprar-ho. Si està desbloquejat, també mostra les seves habilitats.

```
void EnableBuy(bool ena)
{
    btnUnlock.gameObject.SetActive(ena);
    priceTxt.gameObject.SetActive(ena);

    if (ena)
    {
        priceTxt.text = minimumCoins[index].ToString();
        btnUnlock.interactable = SaveSystem.coins >= minimumCoins[index];
    }
    else
    {
        habilitySpawned = Instantiate(habilitiesMenu[index], SelectMenu.transform);
    }
}
```

Figura 6.3.8 Funció 'EnableBuy'.

Per desbloquejar el personatge, es prem el botó de desbloqueig i es marca com a comprat, tornant a cridar a la funció 'Spawn' perquè surti correctament (Figura 6.3.9). A més, es resten les monedes corresponents.

```
public void Unlock()
{
    SaveSystem.characterUnlock[index] = true;
    SaveSystem.coins -= minimumCoins[index];
    coinTxt.text = SaveSystem.coins.ToString();

    ///Animacio en un futur

    Spawn();

    SaveSystem.SaveData();
}
```

Figura 6.3.9 Funció 'Unlock'

Hi ha dues fletxetes als laterals de la pantalla per canviar de personatge. Quan es clica, canvia l'índex i torna a cridar la funció Spawn (Figura 6.3.9).

```
public void Left()
{
    index--;
    if(index < 0)
    {
        index = chars.Length - 1;
    }
    Spawn();
}
```

Figura 6.3.9 Funció 'Left'.

Finalment, quan es prem el botó play (Figura 6.3.10), es guarda el personatge triat per ser el que aparegui la pròxima vegada que es torni a seleccionar personatge. Per acabar, es carrega l'escena de joc.

```
}
public void Play()
{
    PlayerPrefs.SetInt("PlayerSelected", index);
    SceneManager.LoadScene("Joc");
}
```

Figura 6.3.10 Funció 'Play'.

6.3.3 Habilitats

Quan es canvia d'índex, apareix en pantalla un 'Prefab' amb els botons de les habilitats del personatge corresponent (Figura 6.3.10). Aquests estan col·locats a mà perquè quadrin a la pantalla, independentment del nombre d'habilitats.

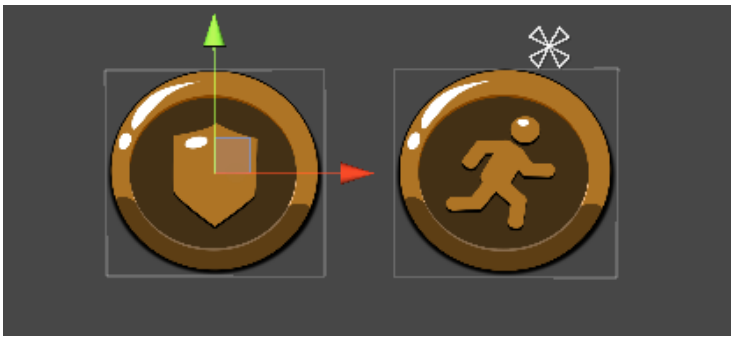


Figura 6.3.10 'Prefab' habilitats

Per cada personatge hi ha un 'Game Object' amb els seus botons. Aquests estan en un 'Array' que s'accedeix depenent de l'índex actual (Figura 6.3.11).

Figura 6.3.11 Inspector - Array habilitats.

S'han creat diferents 'Prefabs' de botons per cada habilitat amb un script anomenat 'Hability Button' (Figura 6.3.12) que guarda la informació que es mostrarà per pantalla.

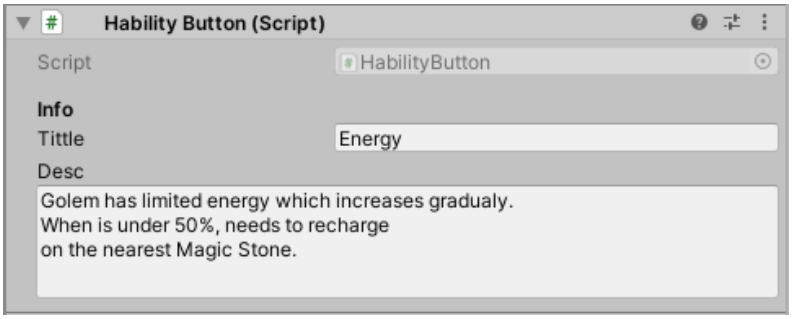


Figura 6.3.12 Script informació habilitat.

Quan es prem una habilitat, s'obre un panell amb tota la informació disponible (Figura 6.3.13), apareixen per sobre dels altres menús.

```
titleText.text = hab.GetComponent<HabilityButton>().tittle;
descText.text = hab.GetComponent<HabilityButton>().desc;
info.SetActive(true);
openedHab.transform.parent = info.transform;
```

Figura 6.3.13 Codi aparició informació

Per destacar l'habilitat la qual s'està mirant la informació, es posa temporalment com a fill del panell que ocupa la pantalla, quedant per sobre i evitant ser tapat (Figura 6.3.14).

Figura 6.3.14 Menú informació habilitat.

En el panell hi ha un botó invisible ocupant tota la pantalla que quan és premut, tenca el panell i torna a posar l'habilitat com estava.

Per crear l'art de les habilitats s'ha utilitzat el software Inkscape. Amb les eines que ofereix s'ha creat la rodona base que conté les habilitats i les icones d'aquestes. Per fer-les més visualment agradables, s'han aplicat diverses capes per aplicar profunditat amb ombres i brillantor (Figura 6.3.15).

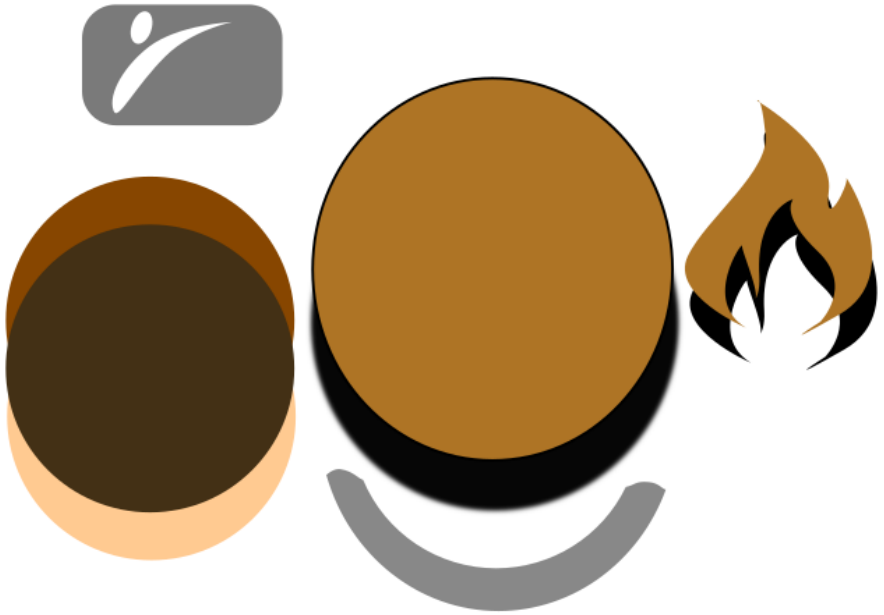


Figura 6.3.16 Botons desmotats en Inkscape.

6.4 Controladors

6.4.1 ManagerGame: Partida

La classe 'ManagerGame' és l'encarregada de connectar tots els elements de la partida amb els altres controladors. Cap classe pot interactuar amb una classe controladora sense passar per aquesta. A part d'això, s'encarrega de iniciar la partida, comptar les variables més importants i finalitzar el joc.

Inici partida

Al mateix instant que es carrega l'escena de la partida, s'executa la funció 'StartGame', la qual és encarregada de preparar el joc i organitzar a les altres classes. La primera part d'aquesta funció (Figura 6.4.1) carrega el personatge triat a l'escena anterior i el guarda a la variable 'ClassTried'. Seguidament crida a la classe 'ObjectManager' referenciada a la variable 'objMan' perquè crei tots els elements del mapa i obté la referència als spawns dels enemics i claus. A més, obté els scripts dels elements que ha creat per poder accedir-hi en qualsevol moment.

```
//GetPlayerSelected
if(!proves)classTried = (GlobalVariables.CharactersTypes)PlayerPrefs.GetInt("PlayerSelected",0);
//SpawnObjects
objMan.SpawnWorld();
enemyRootSpawn = objMan.GetEnemyRoot();
keyRootSpawn = objMan.GetKeyRoot();

chestScript = objMan.GetChestScript();
playerScript = objMan.GetPlayerController();
trapList = objMan.GetTrapList();
magicStoneList = objMan.GetMagicStonesList();
```

Figura 6.4.1 Primera part funció 'StartGame'

La següent part de la funció obté la superfície del terreny i l'aplica al 'NavMeshNavigation' (Figura 6.4.2), encarregat de crear el camí per on podran passar els enemics. Es generà amb la funció 'Bake'.

```
surface = objMan.GetSurface();
navAdmin.setSurface(surface);
navAdmin.Bake();
```

Figura 6.4.2 Segona part funció 'StartGame'

L'última part de la funció (Figura 6.4.3) assigna a la càmera el component 'Transform' del jugador per tenir un objectiu a seguir durant la partida. Conjuntament configura la classe 'UIManager' per canviar la interfície depenent del personatge triat. Inicialitza totes les variables i inicia la classe 'SpawnManager' referenciada a la variable 'spMan', assignant el personatge triat i els 'spawns' d'enemic i claus prèviament llegit de 'ObjectManager'. Per acabar inicialitza la interfície, mostrant les variables inicialitzades.

```
//Config GameObjects
cameraScript.ConfigureCamera(playerScript.transform);
UIManager.Instance.Config(classTried);

//Inicialize variables
lastEnemKillCombo = 0;
isPlaying = true;
gameCount = 0;
money = 0;
enemKillCombo = 0;
resetCombo = false;

spMan.StartGame(classTried, enemyRootSpawn, keyRootSpawn);

ConfigClass();
StartUI();
```

Figura 6.4.3 Tercera part funció 'StartGame'

La funció 'ConfigClass' (Figura 6.4.4) configura elements específics depenent del personatge triat. Actualment només actua quan el personatge és el bufó, incrementant la mida de la càmera i oferint més camp de visió.

```
void ConfigClass()
{
    switch(classTried)
    {
        case GlobalVariables.CharactersTypes.Jester:
            cameraScript.IncreaseSize(extraCameraSize);
            break;
    }
}
```

Figura 6.4.4 Funció 'ConfigClass'

Combo

La classe també s'encarrega de calcular el combo obtingut quan s'eliminen enemics. Aquest es calcula cada vegada que s'elimina un enemic (Figura 6.4.5), augmentant la variable per cada enemic eliminat.

```
public void EnemyDeath() //Called by: EnemyController(on death)
{
    if(resetCombo) resetCombo = false;
    enemKillCombo++;
    if (enemKillCombo > 1000) enemKillCombo = 1000;

    UIManager.Instance.UpdateCombo(enemKillCombo.ToString());

    if(classTried != GlobalVariables.CharactersTypes.King) chestScript.IncrementPunt();
}
```

Figura 6.4.5 Funció 'EnemyDeath'

A la funció 'Update' (Figura 6.4.5) el compta el temps desde l'últim enemic eliminat. Arribat a cert límit, el comptador de combo es reinicia i s'actualitza a la interfície. Quan es torna a eliminar a un enemic, torna a comptar.

```
void Update() ////Controls combo(enemies killed at a short time)
{
    if (isPlaying)
    {
        gameCount += Time.deltaTime; //Time in game
        if(!resetCombo)
        {
            countCombo += Time.deltaTime;
            if(countCombo>= timeResetCombo)
            {
                countCombo = 0;
                lastEnemKillCombo = enemKillCombo;
                enemKillCombo = 0;

                UIManager.Instance.UpdateCombo(enemKillCombo.ToString());
                resetCombo = true;
            }
        }
    }
}
```

Figura 6.4.6 Funció 'Update'

Final de partida

Quan s'acaba la partida indica a la classe 'UIManager' que s'han de mostrar els resultats i acabar el joc. Seguidament executa la funció 'SetEndData' (Figura 6.4.7) per guardar les dades obtingudes. Assigna les variables a la classe 'SaveSystem' i executa la funció 'SaveData'. El onada només la guarda si s'ha superat la màxima obtinguda. Per últim, mostra el resultat d'aquestes variables per pantalla.

```
void SetEndData()
{
    if(wave > SaveSystem.maxWave)
    {
        SaveSystem.maxWave = wave;
    }
    SaveSystem.coins += money;
    if (SaveSystem.coins > 10000) SaveSystem.coins = 10000;
    SaveSystem.SaveData();

    //To select character directly
   // SaveSystem.SelectCharacter = true;

    //Change ui;
    UIManager.Instance.UpdateBestWave(SaveSystem.maxWave.ToString());
    UIManager.Instance.UpdateTotalMoney(SaveSystem.coins.ToString());
}
```

Figura 6.4.7 Funció 'SetEndData'

6.4.2 UIManager: Interficie

La classe 'UIManager' s'encarrega de tota la interfície del joc, desde actualitzar els valors fins a executar la funcionalitat dels botons. A continuació es mostraran les funcions més importants de la classe, sense comptar les encarregades de canviar el text amb un valor donat (monedes, onades...).

La funció 'Config' (Figura 6.4.8) s'executa en començar una partida i canvia la interfície mostrada depenent del personatge triat. Si és el golem, mostra la barra d'energia a la part inferior de la pantalla. Si és el rei activa la imatge d'una moneda per indicar el cost de l'habilitat. Si és l'enterrador activa una imatge d'una planta per indicar quantes hi ha en cada moment. Per acabar, assigna la imatge corresponent al personatge.

```
public void Config(GlobalVariables.CharactersTypes type) //Called
{
    classTried = type;
    switch (type)
    {
        case GlobalVariables.CharactersTypes.Gollem:
            ShowEnergy();
            break;
        case GlobalVariables.CharactersTypes.King:
            KingImage.SetActive(true);
            break;
        case GlobalVariables.CharactersTypes.Undertaker:
            UndertakerImage.SetActive(true);
            break;
    }

    habilityButton.image.sprite = habilityImages[(int)type];
}
```

Figura 6.4.8 Funció 'Config'

Quan el jugador prem el botó de pausa s'amaga el menú de joc i es mostra el de pausa (Figura 6.4.9). A més, es para el temps del joc assignant 0 a la variable 'Time.timeScale' a la funció 'Pause'. També s'assigna el text de les monedes i onades del menú de joc al menú de pausa.

```
public void SetPause()
{
    pauseMoney.text = moneyTxt.text;
    pausePunt.text = waveTxt.text;
    Pause(true);
    gameScene.SetActive(false);
    pauseScene.SetActive(true);
}
```

Figura 6.4.9 Funció 'SetPause'

Quan es torna a la partida, s'assigna 1 al valor de 'Time.timeScale' i es tornen a intercanviar els menús (Figura 6.4.10).

```
public void ResumeGame()
{
    Pause(false);
    gameScene.SetActive(true);
    pauseScene.SetActive(false);
}
```

Figura 6.4.10 Funció 'ResumeGame'

Quan la classe 'ManagerGame' avisa que s'ha acabat la partida, s'obre el menú de resultats i s'assignen les variables corresponents en pantalla (Figura 6.4.11).

```
public void EndOfGame()
{
    gameScene.SetActive(false);


    score.text = waveTxt.text;
    money.text = moneyTxt.text;
    endScene.SetActive(true);
}
```

Figura 6.4.11 Funció 'EndOfGame'

Per últim, quan es torna al menú principal es desactiva la pausa i es carrega l'escena inicial (Figura 6.4.12).

```
public void MainMenu()
{
    Pause(false);
    SceneManager.LoadScene("MainMenu");
}
```

Figura 6.4.12 Funció 'MainMenu'

6.4.3 SpawnManager: Onades

La classe 'SpawnManager' s'encarrega de crear les onades d'enemics i configurar la dificultat en tot moment. Les onades estan dividides en grups d'onades. Cada vegada que se supera una onada, s'augmenta la dificultat. Quan es completa un grup, es reinicia la dificultat però augmentant-la lleugerament.

La funció 'Update' (Figura 6.4.13) compta el temps i crea els enemics quan toca. Una vegada han aparegut tots els enemics de l'onada, espera un temps i continu a la següent.

```
private void Update()
{
    if (gm.Playing())
    {
        switch (actualState)
        {
            case SpawnStates.Spawning:

                spawnCount += Time.deltaTime;
                if (spawnCount >= timeBtwnSpawn)
                {
                    spawnCount = 0;
                    Spawn();
                }
                if (monsterWaveCount >= WavEnem)
                {
                    FinishWave();
                }

                break;

            case SpawnStates.Waiting:
                waveCount += Time.deltaTime;
                if (waveCount >= timeBtwnWaves)
                {
                    NextWave();
                }
                break;
        }

        if(character == GlobalVariables.CharactersTypes.King) SpawningKeys();
    }
}
```

Figura 6.4.13 Funció 'Update'

Quan comença una nova onada, abans de crear els enemics s'executa la funció 'InitWave' (Figura 6.4.15) per configurar quins apareixeran. S'han creat tres 'arrays' que contenen els enemics, el percentatge d'aparèixer i l'onada mínima per crear-se. En el cas dels enemics, hi ha una llista on cada posició és un array de 'prefabs'.

Primer es fa un recorregut per tots els enemics a la inversa, començant per l'ultim i acabant en el segon. A continuació es comprova si s'ha arribat a l'onada suficient perquè aparegui. Si es pot, calcula quin percentatge d'enemics d'aquell tipus han d'aparèixer. Si han d'haver-hi 10 enemics i té un 20%, només apareixeran 2 d'aquell tipus en aquella onada. A cada creació es tria aleatòriament un enemic del grup i s'afegeix a la llista.

Una vegada han passat per tots els grups, s'emplena la resta d'enemics amb el primer de la llista, en aquest cas els zombis. És a dir, si de 10 enemics s'han afegit 6, es creen 4 zombis per emplenar. Després es barreja la llista perquè sigui aleatori l'ordre en el qual apareixen.

```
void InitWave() //Config which enem have to be spawned this wave
{
    int i, j, rand, perc, rest;

    actualState = SpawnStates.Spawning;
    monsterWaveCount = 0;
    enemToSpawn.Clear();

    //Spawn enemies depends on his percentage of the total spawneds
    for(i= enemyGroups.Count-1; i> 0;i--)
    {
        if (wave >= waveToappear[i])
        {
            perc = (int)(WavEnem * percentAppear[i] / 100);
            for (j = 0; j < perc; j++)
            {
                rand = Random.Range(0, enemyGroups[i].Length);
                enemToSpawn.Add(enemyGroups[i][rand]);
            }
        }
    }
    //Zombies to fill the rest of the wave
    rest = WavEnem - enemToSpawn.Count;
    for (i = 0; i < rest; i++)
    {
        rand = Random.Range(0, enemyGroups[0].Length);
        enemToSpawn.Add(enemyGroups[0][rand]);
    }
    Shuffle(enemToSpawn); //Shuffle all enemies to spawn randomly

}
```

Figura 6.4.14 Funció 'InitWave'

A continuació s'adjunta un exemple amb diferents onades i enemics (Figura 6.4.15).

Zombies	0%	0
Ratpenats/ Fantasmes	20%	5
Esquelet	10%	10

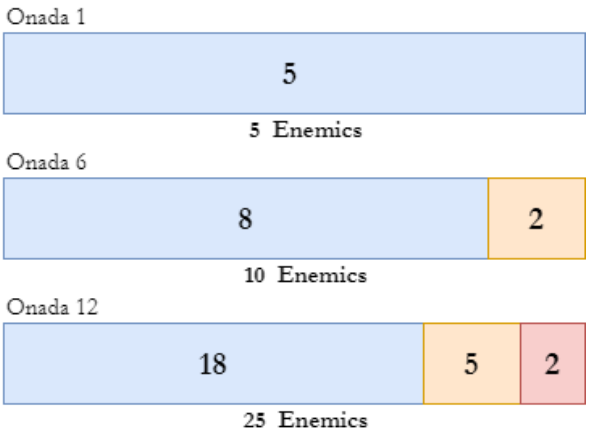


Figura 6.4.15 Exemple onades

A la funció 'Spawn' (Figura 6.4.15) es creen els enemics de la llista. Es tria un dels spawns disponibles i s'assigna un increment de velocitat que va augmentant amb el temps. Per determinar si un 'spawn' està disponible comprova si està fora de la càmera, evitant que apareguin al costat del jugador. L'enemic creat s'afegeix a la llista i s'augmenta el comptador que determinara quan han aparegut tots els enemics.

```
void Spawn()
{
    if (enemiesInGame < maxEnemies)
    {
        List<Transform> dispSpawns = AvaliableSpawns();

        if (dispSpawns.Count > 0)
        {
            num = Random.Range(0, dispSpawns.Count);
            enem = Instantiate(enemToSpawn[monsterWaveCount], dispSpawns[num].position, Quaternion.identity);

            //Increment speed
            enemScript = enem.GetComponent<EnemyController>();
            enemScript.VelIncrement(enemyIncrement);
            enemiesSpawned.Add(enemScript);

            monsterWaveCount++;
            enemiesInGame++;
        }
        else print("NO SPAWNS");
    }
}
```

Figura 6.4.15 Funció 'Spawn'

Quan s'acaba l'onada s'incrementa el comptador, s'augmenta la dificultat i s'inicialitza la següent (Figura 6.4.16).

```
void NextWave()
{
    wave++;
    if (wave > 10000) wave = 10000;

    gm.SetWave(wave);

    waveCount = 0;

    spawnCount = timeBtwnSpawn;

    IncreaseDifficult();
    InitWave();
}
```

Figura 6.4.16 Funció 'NextWave'

Quan s'incrementa la dificultat, es comprova si s'ha completat un grup d'onades (Figura 6.4.17). Això es comprova amb la variable 'NumDif' que conte de quantes onades està formada cada grup. Si encara no s'ha superat, s'incrementa la dificultat de l'onada. Si s'ha completat, s'incrementa la dificultat de grup i es reinicia la d'onada.

```
void IncreaseDifficult() //If wave is comp
{
    difficulty++;
    enemyIncrement += IncEnemVelWave;
    if (difficulty > NumDif)
    {
        difficulty = 1;
        IncreaseGroupIncrementals();
        ResetIncrementals();
    }
    else
    {
        IncreaseIncrementals();
    }
}
```

Figura 6.4.17 Funció 'IncreaseDifficult'

Per incrementar la dificultat s'augmente el nombre d'enemics, el temps entre cada aparició d'enemic i el temps entre onades (Figura 6.4.18).

```
void IncreaseIncrementals() //When wave is completed
{
    WavEnem += IncEnemNumWave;
    timeBtwnSpawn += IncTimeBtwnSpawn;
    timeBtwnWaves += IncTimeBtwnWaves;
}
```

Figura 6.4.18 Funció 'IncreaseIncrementals'

L'augment de dificultat del grup incrementa els valor que s'assignen per defecte quan es comença un grup, fent que cada vegada s'inicialitzi amb més dificultat (Figura 6.4.19).

```
void IncreaseGroupIncrementals() //When wave group i
{
    InitWavEnem += IncEnemNumGroup;
    InitTimeBtwnSpawn += IncTimeBtwnSpawnGroup;
    InitTimeBtwnWaves += IncTimeBtwnWavesGroup;
}
```

Figura 6.4.19 Funció 'IncreaseGroupIncrementals'

Per acabar es reinicien les variables assignant el valor inicial el qual ha anat incrementant amb cada grup superat (Figura 6.4.20).

```
void ResetIncrementals() //When wave group is completed
{
    WavEnem = InitWavEnem;
    timeBtwnSpawn = InitTimeBtwnSpawn;
    timeBtwnWaves = InitTimeBtwnWaves;
}
```

Figura 6.4.20 Funció 'ResetIncrementals'

6.4.4 ObjectManager: Mapa

La classe 'ObjectManager' s'encarrega de tota la creació dels objectes que apareixen en començar la partida, generant aleatòriament l'escenari i les trampes.

Per aconseguir aquesta aleatorietat, s'ha creat 2 terreny diferents que seran la base pel mapa. Cada un té una forma i decoració diferent. L'element en comú són els 'spawns' que hi han d'haver mínimament (jugador, cofre, enemics, claus...). A més, també s'assignen els 'spawns' dels grups d'objectes. Aquests grups contenen elements decoratius i trampes. Per poder accedir a tots aquests objectes desde la classe principal, s'ha creat una classe auxiliar anomenada 'MapTerrain' amb les variables referenciant els elements corresponents (Figura 6.4.21).

```
public Transform rootGroups;
public Transform playerSpawn;
public Transform chestSpawn;
public Transform rootEnemy;
public Transform rootMagicStone;
public Transform rootKey;
public GameObject Obstacles;
public NavMeshSurface surface;
```

Figura 6.4.21 Variables 'MapTerrain'

Els ‘spawns es mostren com a esferes de diferents colors que estan ocults quan es juga (Figura 6.4.22).

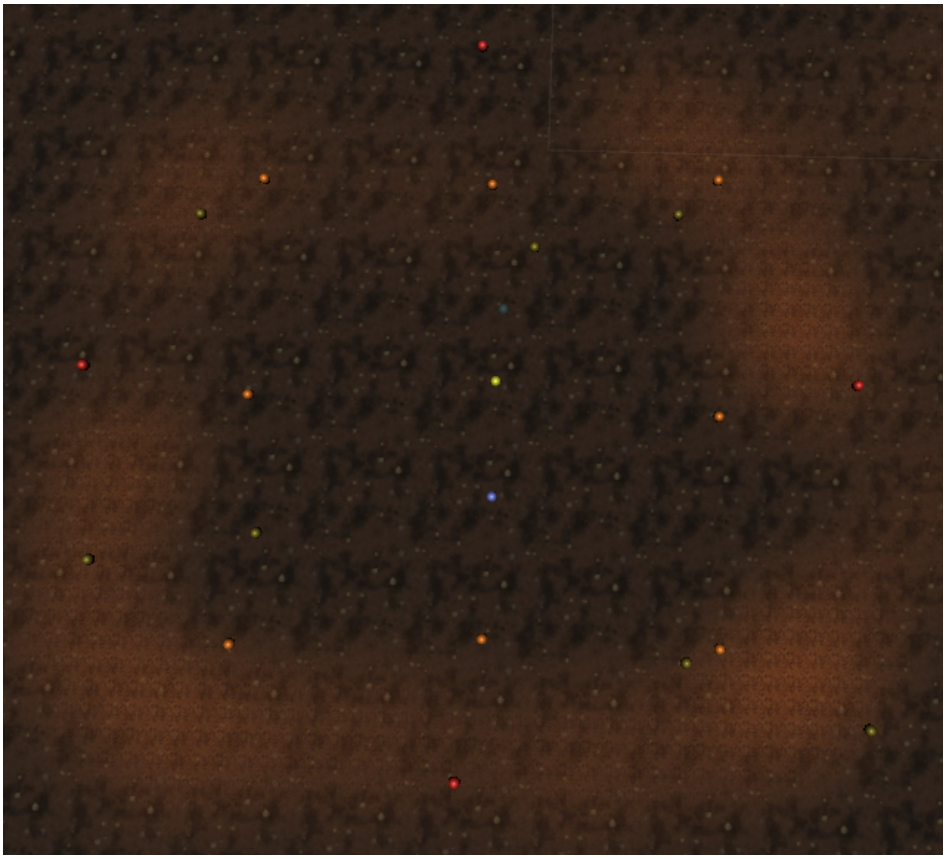


Figura 6.4.22 Spawns terreny

Apliquem una estructura similar per cada grup d’objectes. Són objectes buits que contenen decoració i ‘spawns’ on es generaran les trampes aleatòriament. S’assigna la classe ‘MapGroup’ per contenir l’objecte pare que conte tots els spawns de les trampes (Figura 6.4.23).

```
public Transform rootTraps;

public Transform GetRootTraps()
{
    return rootTraps;
}
```

Figura 6.4.23 Classe ‘MapGroup’

A continuació es mostra un exemple d'un grup (Figura 6.4.24), siguent les esferes blanques el punt d'aparició de les trampes.

Figura 6.4.24 Estructura secció de mapa

La funció 'SpawnWorld' (Figura 6.4.25) s'encarrega de crear tots els elements de l'escenari. També obté el personatge triat.

```
public void SpawnWorld() //Called by Manager(on configure game
{
    classTried = ManagerGame.Instance.GetClassTried();
    //Spawns
    SpawnTerrain();
    SpawnGroupMap();
    SpawnPlayer();
    SpawnChest();

    if (classTried == GlobalVariables.CharactersTypes.Gollem)
        SpawnMagicStones();
}
```

Figura 6.4.25 Funció 'SpawnWorld'

Per començar, tria un terreny aleatòriament dels assignats per paràmetre i el crea a la posició zero del món (Figura 6.4.26). Seguidament agafa tots els 'spawns' guardats en la classe 'MapTerrain' i les assigna a les seves variables.

```
void SpawnTerrain()
{
    rand = Random.Range(0, terrain.Length);

    if (!provisional)
        obj = Instantiate(terrain[rand], Vector3.zero, Quaternion.identity);
    else
    {
        obj = terrainProv;
    }
    map = obj.GetComponent<MapTerrain>();

    rootKeys = map.getRootKey();
    rootEnemies = map.GetRootEnemy();
    rootSpawnMagicStone = map.getRootMagicStone();
    playerSpawn = map.GetPlayerSpawn();
    chestSpawn = map.GetChestSpawn();
    mapSpawns = map.GetRootGroups();
    obstaclesRoot = map.getObstacles();
    surface = map.getSurface();
}
```

Figura 6.4.26 Funció 'SpawnTerrain'

A continuació, barreja l'array amb tots els grups disponibles i els crea en ordre fins a omplir tots els llocs disponibles (Figura 6.4.27). Amb això, a cada partida apareixeran grups diferents. Abans de crear-los, es gira en un angle aleatori en l'eix de les Y. A continuació es creen les trampes de la zona.

```
void SpawnGroupMap()
{
    Shuffle(mapGrups);
    for (int i=0;i<mapSpawns.childCount;i++)
    {
        randFloat = Random.Range(0, 360);
        obj =Instantiate(mapGrups[i], mapSpawns.GetChild(i).position,
                transform.rotation * Quaternion.Euler(0,randFloat,0), obstaclesRoot.transform);
        SpawnTraps(obj.GetComponent <MapGroup>().GetRootTraps());
    }
}
```

Figura 6.4.27 Funció 'SpawnGroupMap'

Per crear les trampes s'ha creat una estructura agrupada en objectes buits (Figura 6.4.28). 'TrapSpawn' és l'arrel el qual es recorrerà per crear les trampes. Després se separen en dos grups, les trampes que apareixeran en mig del mapa que atacaran en una àrea al voltant seu i les que estaran al costat d'un objecte de decoració, atacant en una direcció. Dintre de cada grup estaran els spawns. En cas de trobat un objecte buit amb alguns spawns com a fills, només es creara 1 trampa de tota la llista. Amb això no només canviaran les trampes que apareguin a cada partida sinó també la posició.

Figura 6.4.28 Estructura spawns trampes

Llavors es creen les trampes recorrent aquests grups i creant la trampa aleatòriament (Figura 6.4.28). La classe compta amb una llista d'arrays on cada un és un tipus de trampa diferent, sent actualment 2. Si detecta que és un objecte buit amb spawns com a fill, tria aleatòriament una trampa i la crea.

```
void SpawnTraps(Transform rootSpawnTrap)
{
    Transform spawnType, spawnGroup;
    for (int type = 0; type < rootSpawnTrap.childCount; type++) //Types of traps(lateral, middle,..)
    {
        spawnType = rootSpawnTrap.GetChild(type);
        for (int group = 0; group < spawnType.childCount; group++) //Types of groups of spawns
        {
            spawnGroup = spawnType.GetChild(group);

            if(spawnGroup.childCount > 0) //Is a group (randomize spawn)
            {
                rand = Random.Range(0, spawnGroup.childCount);
                InstantiateTrap(spawnGroup.GetChild(rand), type);
            }
            else //Is not a group
            {
                InstantiateTrap(spawnGroup, type);
            }
        }
    }
}
```

Figura 6.4.28 Funció 'SpawnTraps'

Per crear una trampa es tria una aleatòriament de la llista (Figura 6.4.29). Depèn del grup escull un array de la llista o un altre. Per acabar, l'afegeix a la llista per poder accedir-hi en qualsevol moment.

```
void InstantiateTrap(Transform spawn, int type)
{

    rand = Random.Range(0, trapTypes[type].Length);
    obj = Instantiate(trapTypes[type][rand], spawn.position, spawn.rotation, spawn);
    trapList.Add(obj.GetComponent<Traps>());
}
```

Figura 6.4.29 Funció 'InstantiateTrap'

Després de crear el mapa, crea el jugador corresponent al triat a la posició que pertoca (Figura 6.4.30). Guarda una referència del script 'PlayerController'.

```
void SpawnPlayer()
{
    GameObject obj = Instantiate(characters[(int)classTried], playerSpawn.position, chestSpawn.rotation);
    playerScript = obj.GetComponent<PlayerController>();
}
```

Figura 6.4.30 Funció 'SpawnPlayer'

Per ultim, crea el cofre corresponent depenent del personatge triat (Figura 6.4.31).

```
void SpawnChest()
{
    GameObject obj;

    if (classTried == GlobalVariables.CharactersTypes.King)
        { obj = Instantiate(kingChest, chestSpawn.position, chestSpawn.rotation); }
    else
        { obj = Instantiate(chest, chestSpawn.position, chestSpawn.rotation); }

    chestScript = obj.GetComponent<Chest>();
}
```

Figura 6.4.31 Funció 'SpawnChest'

6.4.5 PowerUpManager: Bonificacions

Quan un jugador interactua amb una bonificació, aquest crida la funció 'GetPower' de la classe 'PowerUpManager' perquè s'encarregui de tots els passos per activar i desactivar el poder. La funció comença per comprovar si ja hi ha un poder actiu, i en aquest cas, el desactiva abans de continuar (Figura 6.4.32). Després, assigna el poder a la variable 'actualPower' en cas de ser activada, o 'None' si es desactiva.

```
public void GetPower(GlobalVariables.Powers pow, bool activate)
{
    if (activate && powerActivate) GetPower(actualPower, false);

    powerActivate = activate;
    count = 0;

    if (activate) actualPower = pow;
    else actualPower = GlobalVariables.Powers.None;
```

Figura 6.4.32 Inici funció 'GetPower'

A continuació activa o desactiva el poder corresponent, executant un codi depenent del tipus (Figura 6.4.33). Tots els poders han de ser activat a través de la classe 'ManagerGame'. Conjuntament s'assigna un valor a la variable 'limit' la qual determina el temps de duració del poder. Els valor assignats s'han configurat per paràmetres. Quan és un poder sense duració s'assigna zero.

```
//Give power up depends on type.
switch(pow)
{
    case GlobalVariables.Powers.Fast: ///Increase Player Speed for a while

        limit = velIncDuration;
        if (activate) gm.ChangePlayerVel(velIncrement);
        else gm.ChangePlayerVel(-velIncrement);

        break;
    case GlobalVariables.Powers.EnemyDeath: //Kill all enemies

        limit = 0;
        if (activate) gm.KillAllEnemies();

        break;
    case GlobalVariables.Powers.EnemyStun: //Stun all enemies

        limit = 0;
        if (activate) gm.StunAllEnemies(stunDuration);

        break;
    case GlobalVariables.Powers.FastTraps: //Traps reload faster for a while

        limit = velTrapIncDuration;
        if (activate) gm.ChangeSpeedTraps(0.5f);
        else gm.ChangeSpeedTraps(2);

        break;
    case GlobalVariables.Powers.Shield: //Gives to player a shield.

        limit = 0;
        if (activate) gm.ActivatePlayerShield();

        break;
}
```

Figura 6.4.33 Final funció 'GetPower'

A la funció 'Update' (Figura 6.4.34) calcula el temps que ha estat actiu si hi ha un poder activat. Quan arriba al límit, es desactiva.

```
void Update()
{
    if(powerActivate)
    {
        count += Time.deltaTime;
        if(count >= limit)
        {
            GetPower(actualPower, false); ///Disable power
        }
    }
}
```

Figura 6.4.34 Funció 'Update'

6.4.6 SaveSystem: Guardar partida

La classe 'SaveSystem' s'encarrega de guardar el valor de les seves variables en un fitxer per poder carregar-ho a la següent partida i mantenir les dades. Per aconseguir-ho fa ús de la classe 'PlayerData' explicada anteriorment, transformant aquesta en un fitxer binari.

La funció 'SaveData' (Figura 6.4.35) crea una variable de tipus 'PlayerData' i la guarda en un fitxer binari amb nom 'Player.Data'. Fa ús de la classe 'BinaryFormatter' inclosa en el Unity per fer-ho.

```
public static void SaveData()
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Application.persistentDataPath + "/player.data";
    FileStream stream = new FileStream(path, FileMode.Create);

    PlayerData data = new PlayerData(characterUnlock,coins,maxWave);

    formatter.Serialize(stream, data);
    stream.Close();
}
```

Figura 6.4.35 Codi 'SaveData'

Quan es volen carregar les dades es busca aquest mateix fitxer i carrega el valor de les variables a les seves pròpies. Si no existeix el fitxer, s'assignen uns valors per defecte (Figura 6.4.36).

```
public static void LoadData()
{
    string path = Application.persistentDataPath + "/player.data";
    if(File.Exists(path))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        PlayerData data = formatter.Deserialize(stream) as PlayerData;
        stream.Close();

        characterUnlock = data.characterUnlock;
        coins = data.coins;
        maxWave = data.maxWave;
    }
    else
    {
        Debug.Log("Save file not found in " + path);

        for (int i = 0; i < characterUnlock.Length; i++) characterUnlock[i] = false;
        characterUnlock[0] = true;
    }
}
```

Figura 6.4.36 Codi 'LoadData'

6.5 Personatges

6.5.1 Càmera

S'ha creat un script per la càmera principal anomenat 'CameraFollow' el qual s'encarrega de seguir al personatge. (Figura 6.5.1). Perquè sigui visualment agradable, segueix al jugador suaument utilitzant la funció 'Vector3.Lerp', evitant la brusquedat.

```
if(target != null)
{
    Vector3 desiredPosition = new Vector3(target.position.x, transform.position.y, target.position.z + distanceZ);

    Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
    transform.position = smoothedPosition;
}
```

Figura 6.5.1 Codi 'CameraFollow'

La classe 'GameManager' li assigna el personatge a seguir i es col·loca a sobre amb unes distàncies com a paràmetres (Figura 6.5.2).

```
public void ConfigureCamera(Transform player) //Called by: Manager(Om configure game)
{
    target = GameObject.FindGameObjectWithTag("Player").transform;
    transform.position = new Vector3(target.position.x, target.position.y + height, target.position.z + distanceZ);
    cam.orthographicSize = height;
}
```

Figura 6.5.2 Funció 'ConfigureCamera'

6.5.2 Base

Els personatges tenen un script base anomenat 'PlayerController'. Aquest s'encarrega del moviment, habilitats i vida dels personatges. No apareix en el joc, ja que fa de plantilla pel codi de cada personatge, heretant tots d'aquest i mantenint la mateixa estructura.

Moviment

Es pot agrupar tot el codi respecte al moviment del personatge en la funció 'Play', anomenada en la funció 'Update' (Figura 6.5.3).

```
//Move and rotate character
void Play()
{
    if (ManagerGame.Instance.Playing())
    {
        // Store the input axes.
        float h = 0;
        float v = 0;

        GetDirection(ref h, ref v);

        AnimateMovement(h, v);
        // Move the player around the scene.
        Move(h, v);
        Rotate(h, v);
    }
}
```

Figura 6.5.3 Funció 'Play'

Comença calcular la direcció en la qual el personatge vol caminar (Figura 6.5.4). Les dades les agafa del 'Joystick' de la interfície i s'arrodoneixen per evitar problemes de números petits.

```
protected virtual void GetDirection(ref float h, ref float v)
{
    if(joystick != null)
    {
        h = Mathf.Round(joystick.Horizontal);
        v = Mathf.Round(joystick.Vertical);
    }

}
```

Figura 6.5.4 Funció 'GetDirection'

El joystick s'ha descarregat a través de 'UnityAssets' i s'ha col·locat en el 'Canvas' principal (Figura 6.5.5), afegint la referència en la variable 'joystick' de 'PlayerController'.

Figura 6.5.5 Joystick

Seguidament anima el moviment assignant la variable velocitat del controlador (Figura 6.5.6). La velocitat es calcula sumant el valor de les dues direccions, sumant zero si no es mou en cap.

```
void AnimateMovement(float h, float v)
{
    anim.SetFloat("Speed", Mathf.Abs(h) + Mathf.Abs(v));
}
```

Figura 6.5.6 Funció 'AnimateMovement'

S'assignen les direccions a la variable 'Movement' de tipus Vector3 (Figura 6.5.7). Normalitzem i ho multipliquem per la velocitat assignada per paràmetre. Per últim, movem el personatge en aquella direcció. Si té assignat una reducció de velocitat, també es dividira en el càlcul de la variable.

```
void Move(float h, float v)
{
    // Set the movement vector based on the axis input.
    movement.Set(h, 0f, v);

    // Normalise the movement vector and make it proportional to the speed per second.
    movement = (movement.normalized * actualSpeed * Time.deltaTime)/slowReduction;

    // Move the player to it's current position plus the movement.
    rb.MovePosition(transform.position + movement);
}
```

Figura 6.5.7 Funció 'Move'

Per últim, aplica una rotació al model del personatge depenent de la direcció (Figura 6.5.8). Es calcula l'angle al qual es vol anar amb la funció 'Mathf.Atan2' i s'aplica al jugador lentament amb 'Quaternion.Lerp'.

```
protected virtual void Rotate(float h, float v)
{
    float heading = Mathf.Atan2(h, v);
    if(h!=0 || v!=0)
    {
        //transform.rotation = Quaternion.Euler(0, heading * Mathf.Rad2Deg, 0);
        var desiredRotQ = Quaternion.Euler(0, heading * Mathf.Rad2Deg, 0);

        transform.rotation = Quaternion.Lerp(transform.rotation, desiredRotQ, Time.deltaTime * damping);
    }
}
```

Figura 6.5.8 Funció 'Rotate'

Habilitat

Quan el botó d'habilitat és premut, es crida la funció 'UsePower' (Figura 6.5.9) per activar el poder corresponent i desactivar-ho.

```
public void UsePower() //Called by GameManager(on button pressed)
{
    DisablePower();
    DoPower();
}
```

Figura 6.5.9 Funció 'UsePower'

Per desactivar el poder simplement posem a 'false' la variable 'canHability' (Figura 6.5.10), encarregada de saber si una habilitat està disponible. A més, s'avisa al 'ManagerGame' perquè desactivi el botó.

```
protected virtual void DisablePower()
{
    canHability = false;
    ManagerGame.Instance.EnableHability(false);
}
```

Figura 6.5.10 Funció 'DisablePower'

A cada frame es crida a la funció 'ReloadPower' (Figura 6.5.11.), la qual conta el temps desde que la habiliat ha estat activada i la torna a activar passat cert temps límit, variant entre cada personatge. 'EnablePower' té el mateix funcionament que 'DisablePower', però activant el botó i assignant 'true' a la variable 'canHability'.

```
void ReLoadPower()
{
    if (!canHability)
    {
        habilityCount += Time.deltaTime;
        if (habilityCount >= habilityCD)
        {
            habilityCount = 0;
            EnablePower();
        }
    }
}
```

Figura 6.5.11 Funció 'ReloadPower'

Vida

S'han creat diverses funcions per encarregar-se de la vida dels personatges. Aquestes es criden en les dues funcions per detectar col·lisions, 'OnTriggerEnter' i 'OnColissionEnter'. La primera (Figura 6.5.12) s'encarrega d'aplicar reducció de velocitat si s'interactua amb la trampa de lentitud, agafar les monedes i notificar-ho a la classe 'ManagerGame' per tenir un recompte. A part d'això, crida la funció 'GetDamage' quan interactua amb l'àrea de dany d'una trampa.

```
protected virtual void OnTriggerEnter(Collider other)
{
    if (other.tag == "Daño")
    {
        GetDamage(1, other.tag);
    }
    else if (other.tag == "Slow")
    {
        Slow(true);
    }
    else if (other.tag == "Moneda")
    {
        ManagerGame.Instance.IncreaseMoney(other.GetComponent<Coin>().GetValue());
        Destroy(other.gameObject);
    }
}
```

Figura 6.5.12 Funció 'OnTriggerEnter'

Com els enemics no tenen assignat un 'trigger', la seva col·lisió es crida amb 'OnCollisionEnter' (Figura 6.5.13). Si un enemic toca al jugador, es crida la funció 'GetDamage'. Amb l'excepció que sigui un enemic tipus Esquelet, el qual si està en procés de reviure, interactuar amb el jugador l'elimina per sempre i no fa cap mal al jugador.

```
private void OnCollisionEnter(Collision collision) //Get damage
{
    if (collision.transform.tag == "Enemigo")
    {
        EnemyController co = collision.gameObject.GetComponent<EnemyController>();
        if (co is SkeletonController)
        {
            SkeletonController sk = co as SkeletonController;
            if (sk.IsReviving())
            {
                sk.PermaDeath();
            }
            else
                GetDamage(1, collision.transform.tag);
        }
        else
            GetDamage(1, collision.transform.tag);
    }
}
```

Figura 6.5.13 Funció 'OnCollisionEnter'

La funció 'GetDamage' (Figura 6.5.14) comprova si el jugador està immunitzat i si ho està, no fa res. Després comprova si disposa de l'escut aplicat per la bonificació. Si el te, el desactiva i el posa en estat d'immunitat. Si no el te crida la funció 'ReduceLife'.

```
protected void GetDamage(int damage, string origin)
{
    if(!inmunity)
    {
        if(!shield)
        {
            ReduceLife(damage, origin);
        }
        else
        {
            ActivateShield(false);
            GetInmunity();
        }

    }
}
```

Figura 6.5.14 Funció 'GetDamage'

La funció 'ReduceLife' (Figura 6.5.15) disminueix la vida del jugador (actualment tots tenen 1) i si arriba a 0, mata al jugador. La variable 'Origin' passada per paràmetre conté l'etiqueta de l'objecte que ha fet mal al personatge. Actualment no té cap ús, però en un futur pot servir per canviar el comportament depenent de l'origen.

```
protected virtual void ReduceLife(int damage, string origin)
{
    actualLife-=damage;
    if (actualLife <= 0)
    {
        Death();
    }
}
```

Figura 6.5.15 Funció 'ReduceLife'

Per últim, la funció 'Death' (Figura 6.5.16) activa l'animació de mort i avisa que s'ha acabat la partida. S'utilitza la variable 'alive' per evitar que s'executi el codi més d'una vegada.

```
protected void Death()
{
    if(alive)
    {

        alive = false;
        anim.SetTrigger("Die");
        ManagerGame.Instance.EndOfGame();
    }

}
```

Figura 6.5.16 Funció 'Death'

Immunitat

Quan el jugador rep mal i sobreviu al mal o activa una habilitat especifica, s'aplica una immunitat. Aquesta s'aplica simplement afegint un condicional a la funció de rebre mal. Però per avisar al jugador que és invencible, ve acompanyada d'una animació.

La funció 'InmunityAnimation' (Figura 6.5.17) conta el temps que dura la immunitat i la desactiva quan supera el límit establert per paràmetres. A la vegada, cada cert temps crida la funció 'Blink' que desactiva el component 'mesh' del jugador, fent-lo invisible i el torna a activar passat un temps curt. Amb això es crea un efecte de parpadeig típicament utilitzat en altres videojocs.

```
void InmunityAnimation() //Blinks when inmunity
{
    if(inmunity)
    {
        inmunitycount += Time.deltaTime;
        blinkCount += Time.deltaTime;
        if(blinkCount>blinkTime)
        {
            blinkCount = 0;
            StartCoroutine(Blink());
        }
        if(inmunitycount > inmunityDuration)
        {
            StopInmunity();
        }
    }
```

Figura 6.5.17 Funció 'InmunityAnimation'

Per desactivar la immunitat (Figura 6.5.18) s'assigna false a la variable que ho comprova i es reinicien els comptador d'immunitat i parpadeig, perquè comencin per zero la pròxima vegada que es necessitin. A més, s'activa el 'mesh' per evitar la possibilitat que el jugador es quedi en invisible. No hauria de poder passar, ja que la funció 'Blink' l'activa sempre abans d'acabar.

```
protected void StopInmunity()
{
    inmunitycount = 0;
    blinkCount = 0;
    inmunity = false;
    EnableMesh(true);
}
```

Figura 6.5.18 Funció 'StopInmunity'

6.5.3 Mag

El personatge mag és controlat per l'script 'WizardController', fill de 'PlayerController'. Per això, les funcions explicades anteriorment s'apliquen automàticament al jugador. L'únic canvi és el poder. El seu poder permet fer un petit impuls d'una direcció i el qual l'atorga immunitat.

Per fer això, canviem la funció 'DoPower' (Figura 6.5.19) per afegir el codi necessari. Activa la immunitat i incrementa la velocitat a la guarda en la variable 'dashSpeed'. També assigna el valor 'true' a la variable 'dashing' per saber que està en procés d'impuls.

```
protected override void DoPower()
{
    GetInmunity();
    ChangeVel(dashSpeed);
    dashing = true;
}
```

Figura 6.5.19 Funció 'DoPower'

En la funció 'FixedUpdate' (Figura 6.5.20) s'afegeix un comptador per calcular quan ha superat el temps límit d'impuls. S'utilitza la variable 'immunityDuration' per calcular quan acabar l'impuls. Així l'impuls i la immunitat acabaran a la vegada. Una vegada acabat, es torna a assignar 'false' a la variable 'dashing' i es redueix la velocitat, tornant a la normalitat.

```
protected override void FixedUpdate()
{
    base.FixedUpdate();
    if (dashing)
    {
        dashCount += Time.deltaTime;
        if(dashCount >= inmunityDuration)
        {
            dashCount = 0;
            dashing = false;
            ChangeVel(-dashSpeed);
        }
    }

}
```

Figura 6.5.20 Funció 'FixedUpdate'

Per acabar, no volem permetre que el jugador pugui canviar de direcció durant l'impuls. Afegim un condicional a la funció 'GetDirection' (Figura 6.5.21) en el qual s'assigna sempre la direcció en què està mirant, caminant sempre recte.

```
protected override void GetDirection(ref float h, ref float v)
{
    if(!dashing)
        base.GetDirection(ref h, ref v);
    else
    {
        if (h == 0 && v == 0)
        {
            Vector3 dir = transform.forward;

            h = Mathf.Round(dir.x);
            v = Mathf.Round(dir.z);
        }
    }

}
```

Figura 6.5.21 Funció 'GetDirection'

6.5.4 Guerrer

L'habilitat del guerrer li permet tenir equipat un escut el qual el fa immune al dany de les trampes, però tenint la velocitat reduïda. Quan és tocat per un enemic, es destrueix l'escut i perd l'avantatge, recuperant la velocitat normal. Per aplicar aquest, afegim una variació a la funció 'ReduceLife' (Figura 6.5.22). Si té l'escut equipat, ignora el mal que provingui d'una trampa ('Daño'). Si ve de qualsevol altre element, és un enemics, pel qual traiem la protecció.

```
protected override void ReduceLife(int damage, string origin)
{
    if(hasArmor)
    {
        if (origin != "Daño")
        {
            HitWithArmor();
        }
    }
    else
    {
        base.ReduceLife(damage, origin);
    }
}
```

Figura 6.5.22 Funció 'ReduceLife'

Amb la funció 'HitWithArmor' (Figura 6.5.23) es treu l'escut i s'entra en estat d'immunitat durant una estona. Es desactiva el poder per sempre. S'assigna a la variable 'HabilityCD' (encarregada de calcular quan torna a estar disponible l'habilitat) el valor de 'timeRepairArmor' (El temps que tarda a reparar-se l'escut). Així podrà tornar a equipar-se l'escut passat un cert temps.

```
void HitWithArmor()
{
    LoseArmor();
    habilityCD = timeRepairArmor;
    GetInmunity();
    armorAvaliable = false;
    DisablePower();
}
```

Figura 6.5.23 Funció 'HitWithArmor'

Per treure l'armadura s'activa l'animació corresponent i es desactiva el model de l'escut (Figura 6.5.24). Seguidament s'incrementa la velocitat.

```
void LoseArmor()
{
    if(armorAvaliable)
    {
        anim.SetBool("Shield",false);
        armor.SetActive(false);
        hasArmor = false;
        ChangeVel(speedInc);
    }

}
```

Figura 6.5.24 Funció 'LoseArmor'

El model sense escut quedaria com es mostra la següent imatge (Figura 6.5.25).

Figura 6.5.25 Guerrer sense escut

Per recuperar l'armadura, s'assigna l'animació i s'activa el model de l'escut (Figura 6.5.26), reduint de nou la velocitat.

```
void RestoreArmor()
{
    if (armorAvaliable)
    {
        anim.SetBool("Shield", true);
        armor.SetActive(true);
        hasArmor = true;
        ChangeVel(-speedInc);
    }
}
```

Figura 6.5.26 Funció 'RestoreArmor'

El model amb escut quedaria com es mostra la següent imatge (Figura 6.5.27).

Figura 7.4.27 Guerrer escut

Perquè el jugador pugui equipar i desequipar l'escut quan vulgui, s'afegeix el codi a la funció 'DoPower' (Figura 6.5.28). Només pot canviar i no s'ha trencat l'escut, és a dir, que la variable 'hasArmor' estigui a 'true'.

```
protected override void DoPower()
{
    if (hasArmor)
        LoseArmor();
    else
        RestoreArmor();
}
```

Figura 6.5.28 Funció 'DoPower'

A part de desactivar i desactivar el model de l'escut, també s'han afegit animacions perquè camini diferent depenent de l'estat en el qual es trobi (Figura 6.5.29)

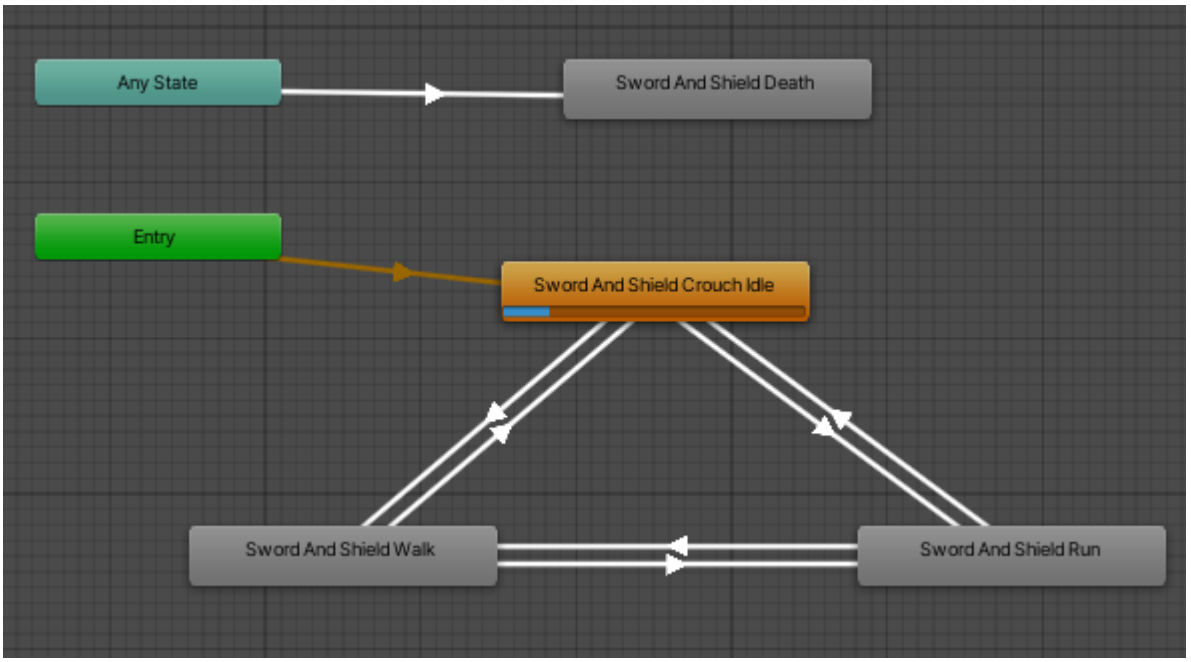


Figura 6.5.29 Controlador animacions Guerrer

6.5.5 Bufó

La característica principal del bufó és que no pot parar mai de córrer. Per aconseguir això, s'ha variat la funció 'GetDirection' (Figura 6.5.30). Igual que amb l'impuls del mag, s'ha fet que la direcció sempre sigui la que està mirant el model. Això s'aplica només quan es queda quiet, siguent les variable 'h' i 'v' igual a zero. El variable 'auto' té assignat 'false' quan comença la partida i es posa a 'true' quan es mou per primera vegada. Amb això s'evita que el jugador comenci ja moguen-se.

```
protected override void GetDirection(ref float h, ref float v)
{
    base.GetDirection(ref h, ref v);
    if (auto)
    {
        if(h == 0 && v == 0)
        {
            Vector3 dir = transform.forward;

            h = Mathf.Round(dir.x);
            v = Mathf.Round(dir.z);
        }
    }
    else
    {
        if (h != 0 || v != 0)
            auto = true;
    }
}
```

Figura 6.5.30 Funció 'GetDirection'

L'altra característica principal és que pot veure més enllà que els altres personatges. Per fet això s'ha augmentat la mida ortogràfica de la càmera en començar la partida. S'assigna el personatge triat a través de la classe 'ManagerGame' i canvia la càmera en cas de ser el bufó (Figura 6.5.31).

```
switch(classTried)
{
    case GlobalVariables.CharactersTypes.Jester:
        cameraScript.IncreaseSize(extraCameraSize);
        break;
}
```

Figura 6.5.31 Codi 'ManagerGame'

L'habilitat que pot activar el jugador és la capacitat de fer un esprint, augmentant la velocitat però reduint el camp de visió, fent difícil el seu control. A més, va deixant una estela de foc que fa mal a tots els personatges, inclòs el bufó. La funció 'DoPower' crida a la funcio 'ActiveFastMode' (Figura 6.5.32). Aquesta última canvia la velocitat i assigna l'animació que toca. També canvia el mida de la càmera a través del 'ManagerGame'.

```
void ActiveFastMode()
{
    powerActive = true;
    ChangeVel(velIncrement);
    anim.SetBool("Fast", true);
    ManagerGame.Instance.IncreaseCameraSize(-cameraSizeReduction);
}
```

Figura 6.5.32 Funció 'ActiveFastMode'

A la funció 'FixedUpdate' (Figura 6.5.33) compta el temps en el qual està amb el poder activat i el desactiva quan toca. També, crea uns petits objectes que fan de foc cada cert temps, assignant-los un temps de vida assignat per paràmetre.

```
protected override void FixedUpdate()
{
    base.FixedUpdate();
    if(powerActive)
    {
        fireCount += Time.deltaTime;
        if(fireCount > timeBtwFire)
        {
            fireCount = 0;
            obj = Instantiate(Fire, FirePosition.position, Quaternion.identity);
            obj.GetComponent<Fire>().SetLifeTime(fireDuration);
        }

        powerCount += Time.deltaTime;
        if(powerCount > powerDuration)
        {
            DisableFastMode();
        }
    }
}
```

Figura 6.5.33 Funció 'FixedUpdate'

Per saber la posició on ha d'aparèixer el foc, s'ha creat un objecte buit a darrere del personatge (Figura 6.5.34). Com apareixen molt seguidament mentre es mou, crea l'efecte d'una línia de foc.

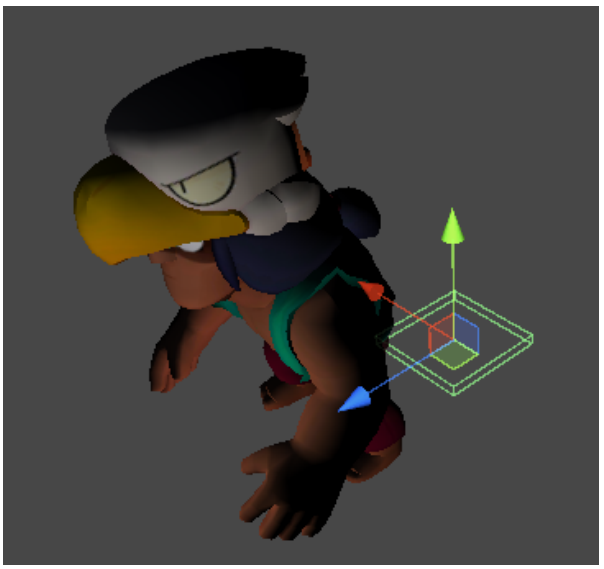


Figura 6.5.34 Posició aparició del foc

L'objecte de foc és un objecte amb unes partícules de foc (Figura 6.5.35). Té assignat l'etiqueta 'Daño' per fer mal com una trampa.

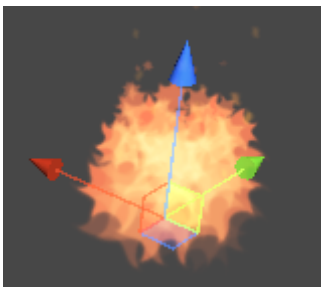


Figura 6.5.35 Particules foc

Quan s'acaba el temps del poder, es desactiva reiniciant els comptadors i tornar a posar l'animació normal (Figura 6.5.5). Es redueix la velocitat i la càmera torna a la seva mida original. A més, activa el poder un altre cop si ja s'havia passat el temps de recàrrega.

```
void DisableFastMode()
{
    powerCount = 0;
    fireCount = 0;
    powerActive = false;
    anim.SetBool("Fast", false);
    EnablePower();

    ManagerGame.Instance.IncreaseCameraSize(cameraSizeReduction);
    ChangeVel(-velIncrement);
}
```

Figura 6.5.36 Funció 'DisableFastMode'

El controlador d'animacions del bufó (Figura 6.5.37) afegeix una animació per quan està fent un esprint.

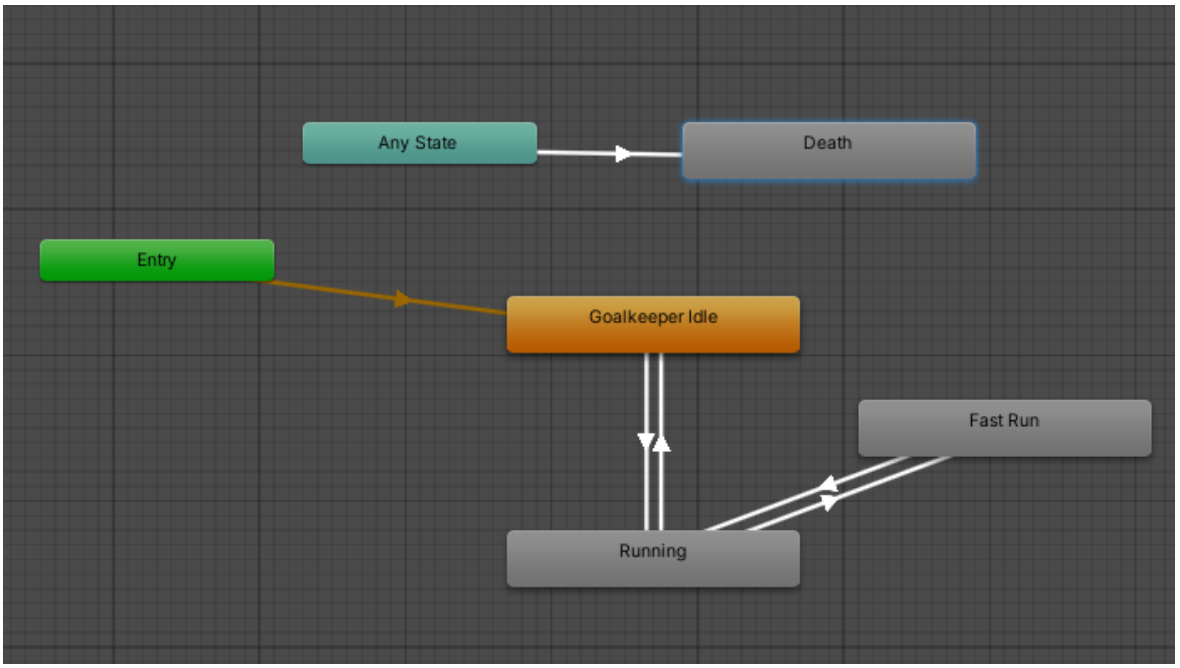


Figura 6.5.37 Funció 'Controlador Bufó'

6.5.6 Rei

L'habilitat especial del rei és que pot pagar certa quantitat de diners per eliminar a tots els enemics. Aquesta és la funcionalitat del codi de 'DoPower' (Figura 6.5.38). Si el jugador té els diners necessaris, simula aconseguir la bonificació 'EnemyDeath i resta el cost als diners totals. A més, a cada us incrementa el cost per tornar-ho a utilitzar i es mostra per pantalla.

```
protected override void DoPower()
{
    if(ManagerGame.Instance.GetCoins() >= actualminCoin )
    {
        ManagerGame.Instance.GetPower(GlobalVariables.Powers.EnemyDeath, true);
        ManagerGame.Instance.IncreaseMoney(-actualminCoin);

        actualminCoin += incCoin;
        ManagerGame.Instance.ChangeButtonText(actualminCoin.ToString());
    }
}
```

Figura 6.5.38 Funció 'DoPower'

Un canvi important és que s'intercanvia el cofre de bonificacions per un de diners. Això es canvia en crear el mapa mitjançant la classe 'ObjectManager' (Figura 6.5.39), assignant el personatge triat a través de 'ManagerGame'. Més informació a l'apartat "6.10 Cofres".

```
if (classTried == GlobalVariables.CharactersTypes.King)
    { obj = Instantiate(kingChest, chestSpawn.position, chestSpawn.rotation); }
else
    { obj = Instantiate(chest, chestSpawn.position, chestSpawn.rotation); }
```

Figura 6.5.39 Codi 'ObjectManager'

A més, en el mapa apareixeran unes claus (Figura 6.5.40) que serviran per obtenir millors recompenses en el cofre.

Figura 6.5.40 Model clau

D'això s'encarregara 'SpawnManager', fent el procés d'aparició de claus si el jugador és el rei (Figura 6.5.41).

```
if(character == GlobalVariables.CharactersTypes.King) SpawningKeys();
```

Figura 6.5.41 Codi 'SpawnManager'

El procés de la funció 'SpawningKeys' (Figura 6.5.42) és crear una clau cada cert temps. Aquest temps és entrat per paràmetre a la mateixa classe.

```
void SpawningKeys()
{
    keySpawnCount += Time.deltaTime;
    if(keySpawnCount > keySpawnCD)
    {
        keySpawnCount = 0;
        SpawnKey();
    }
}
```

Figura 6.5.42 Funció 'SpawningKeys'

Per cada clau es tria una de les posicions possibles aleatòriament i es crea aproximadament en el punt (Figura 6.6.43). Es calcula una variació per no aparèixer en la posició exacta i tapar una altra clau que ja hi havia creada.

```
void SpawnKey()
{
    int rand = Random.Range(0, keySpawnersRoot.childCount);
    float x = Random.Range(-2,2);
    float z = Random.Range(-2, 2);
    Vector3 pos = keySpawnersRoot.GetChild(rand).position;

    Instantiate(kingKey, new Vector3(pos.x+x, pos.y, pos.z + z), Quaternion.identity);
}
```

Figura 6.5.43 Funció 'SpawnKey'

Les posicions on pot aparèixer una clau estan posicionades en el terreny a mà (Figura 6.5.44). Cada cercle daurat és una posició possible.

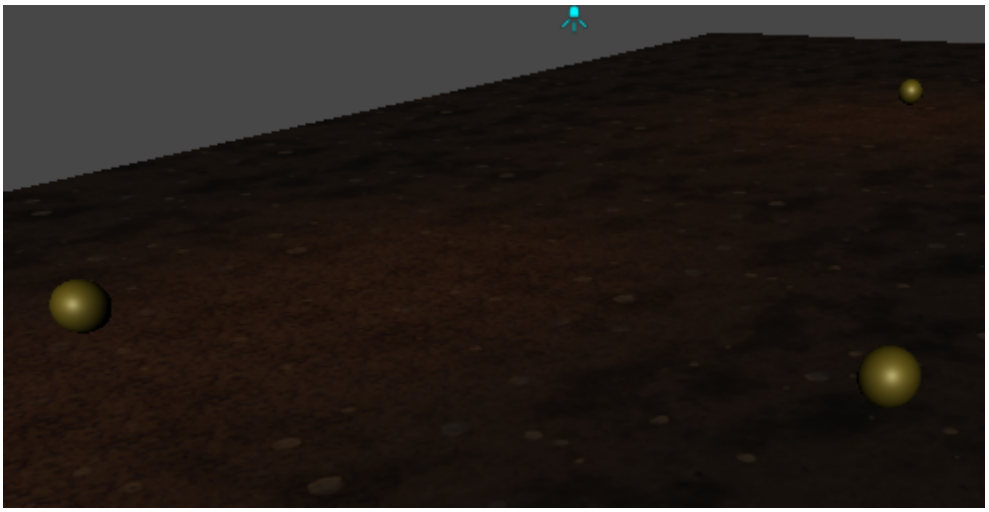


Figura 6.5.44 Posicions aparicio claus

Finalment, afegim que pugui interactuar amb les claus, incrementant el comptador a través de 'ManagerGame' (Figura 6.5.45).

```
protected override void OnTriggerEnter(Collider other)
{
    base.OnTriggerEnter(other);
    if(other.tag == "Key")
    {
        ManagerGame.Instance.GetKey();
        Destroy(other.gameObject);
    }
}
```

Figura 6.5.45 Funció 'OnTriggerEnter'

6.5.7 Golem

El golem conté una energia que determina el seu estat. Quan està al màxim pot activar el mode deu per atacar els enemics amb la seva habilitat, però disminuint l'energia fins a 0. Quan està per sota de la meitat està cansat i ha d'anar a carregar en una pedra màgica que apareix en el mapa. Quan és per sobre la meitat es carrega sol. La funció 'Controller' (Figura 6.5.46) s'encarrega d'executar el codi corresponent a l'estat actual.

```
void Controller()
{
    switch(actualState)
    {
        case State.Tired:
            break;
        case State.Recharge:
            ReloadEnergy();
            break;
        case State.Normal:
            IncrementEnergy();
            break;
        case State.God:
            GodMode();
            break;
    }
}
```

Figura 6.5.46 Funció 'Controller'

Per començar s'han de configurar les pedres màgiques. S'avisa a la classe 'ObjectManager' (Figura 6.5.47) perquè faci aparèixer aquestes pedres en punts específics del mapa quan el personatge sigui el golem. Aquestes posicions són col·locades en el terreny manualment.

```
if (classTried == GlobalVariables.CharactersTypes.Gollem)
    SpawnMagicStones();
```

Figura 6.5.47 Codi 'ObjectManager'

A continuació es canvia la funció 'OnTriggerEnter' (Figura 6.5.48) perquè passi d'estar cansat a estar carregant l'energia.

```
protected override void OnTriggerEnter(Collider other)
{
    base.OnTriggerEnter(other);

    if(other.tag == "MagicStone" && actualState == State.Tired)
    {
        TiredToRecharge();
    }
}
```

Figura 6.5.48 Funció 'OnTriggerEnter'

També editem la funció 'OnTriggerExit' (Figura 6.5.49) per parar la recàrrega quan surt del radi i no s'ha carregat suficient.

```
protected override void OnTriggerExit(Collider other)
{
    base.OnTriggerExit(other);

    if (other.tag == "MagicStone" && actualState == State.Recharge)
    {
        RechargeToTired();
    }
}
```

Figura 6.5.49 Funció 'OnTriggerExit'

La pedra és un model descarregat de la 'Asset Store' amb un 'trigger' i l'etiqueta 'MagicStone' assignada (Figura 6.5.50). També conté una animació d'activació quan el golem passa a estar cansat.

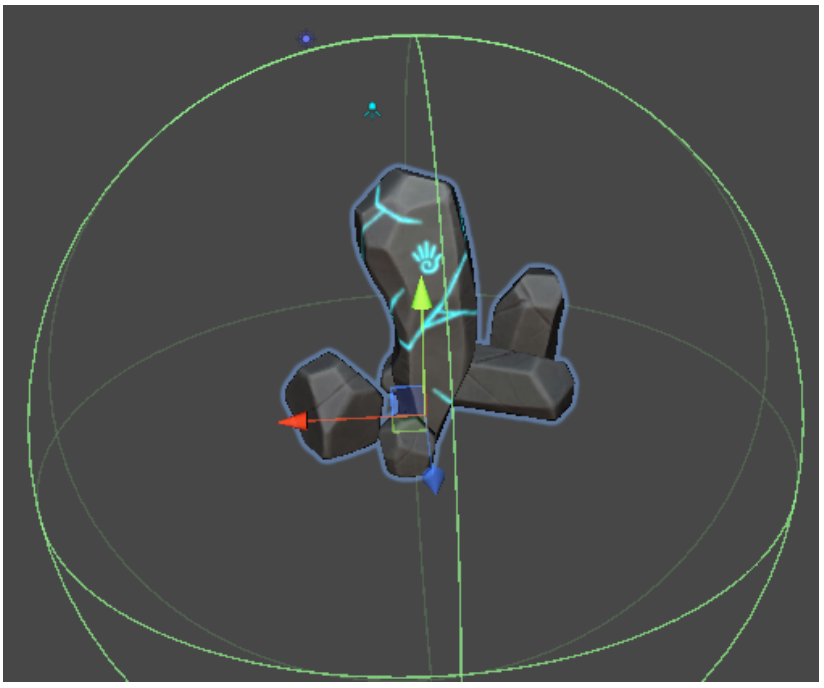


Figura 6.5.50 Model pedra màgica

A continuació cal mostrar l'energia actual per pantalla. Per fer això avisem a la classe 'ManagerUI' (Figura 6.5.51) per mostrar l'energia si el personatge és el golem.

```
case GlobalVariables.CharactersTypes.Gollem:
    ShowEnergy();
    break;
```

Figura 6.5.51 Codi 'ManagerUI'

En la mateixa classe creem una funció per anar actualitzant l'energia que es veu en pantalla (Figura 6.5.52). A més, s'afegeix un canvi de color depenent de en quin rang es trobi.

```
public void UpdateEnergy(float energy)
{
    GollemEnergy.value = energy;
    if(energy < 50)
    {
        fillEnergy.color = energyLow;
    }
    else if(energy >50 && energy < 100)
    {
        fillEnergy.color = energyMedium;
    }
    else
    {
        fillEnergy.color = energyHigh;
    }
}
```

Figura 6.5.52 Funció 'UpdateEnergy' de 'ManagerUI'

A la classe 'GollemController' (Figura 6.5.53) creem una funció per avisar a 'ManagerGame' d'actualitzar l'energia mostrada.

```
void ChangeEnergyUI()
{
    ManagerGame.Instance.ChangeEnergy(energy);
}
```

Figura 6.5.53 Funció 'ChangeEnergyUI'

La barra és un 'Slider' col·locat en el 'canvas' i lleugerament personalitzar perquè sigui visualment agradable (Figura 6.5.54).

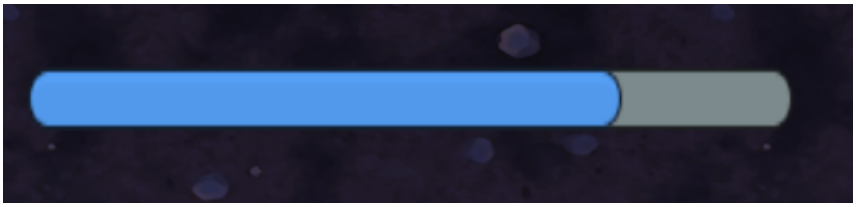


Figura 6.5.54 Slider energia actual

Quan el personatge està cansat i entra en contacte amb l'àrea de la pedra, canvia d'estat aplicant l'animació corresponent i activant unes partícules (Figura 6.5.55).

```
public void TiredToRecharge()
{
    ChangeState(State.Recharge);
    effect.Play();
}
```

Figura 6.5.55 Funció 'TiredToRecharge'

Mentre està en estat de recarregat, l'energia augmenta a la velocitat entrada per paràmetre (Figura 6.5.56). Una vegada arribat a la meitat, canvia a l'estat normal. La funció 'RechargeToNormal' té el mateix funcionament que 'TiredToRecharge', però desactivant les partícules.

```
void ReloadEnergy()
{
    energy += Time.deltaTime * reloadIncrement;
    if (energy >= 50)
    {
        energy = 50;
        RechargeToNormal();
    }
    ChangeEnergyUI();
}
```

Figura 6.5.56 Funció 'ReloadEnergy'

Quan activa l'habilitat i crida la funció 'DoPower', es passa a estat Déu (Figura 6.5.57), el qual ataca als enemics cada cert temps, aconseguint immunitat i augmentant la velocitat.

```
public void NormalToGod()
{
    fullLoaded = false;
    GetInmunity();
    ChangeVel(speedGodIncrement);
    attackCount = 0;
    ChangeState(State.God);
}
```

Figura 6.5.57 Funció 'NormalToGod'

Per atacar els enemics, s'ha creat una àrea als peus del personatge (Figura 6.5.58) amb l'etiqueta 'Daño' per simular el mal d'una trampa. Aquesta es manté inactiva fins a activar el mode Déu.

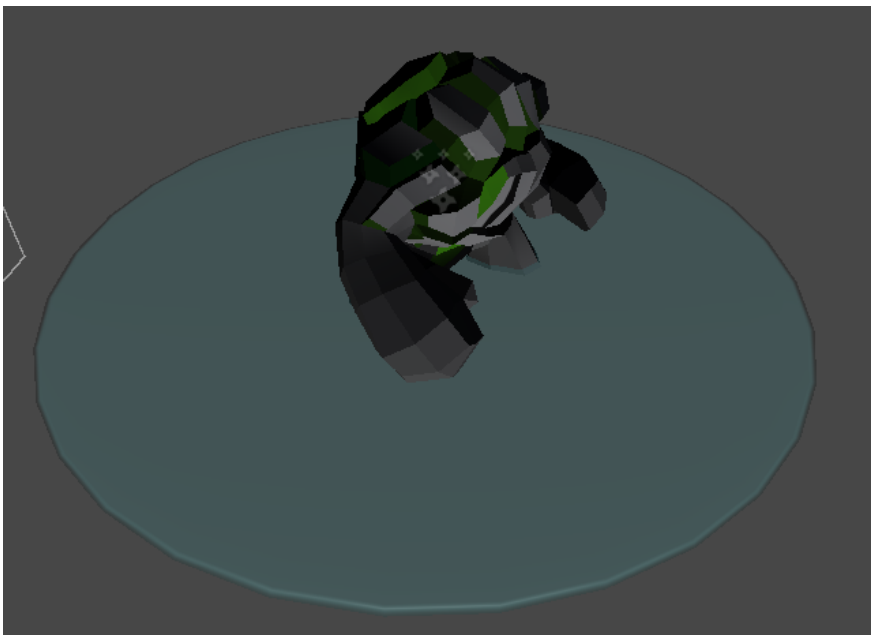


Figura 6.5.58 Àrea dany golem

En aquest estat va disminuint l'energia fins a 0 (Figura 6.5.59). De mentes, activa i desactiva l'àrea cada cert temps i elimina els enemics al seu voltant.

```
void GodMode()
{
    DecrementEnergy();
    if(energy > 5)
    {
        attackCount += Time.deltaTime;
        if (attackCount > attackSpeed)
        {
            attackCount = 0;
            damageZone.SetActive(true);
            StartCoroutine(disableDamage());
        }
    }

}
```

Figura 6.5.59 Funció 'GodMode'

Una vegada arriba l'energia a zero, s'acaba la immunitat i es canvia a estat de cansat (Figura 6.5.60). Es redueix l'augment de velocitat i seguidament s'aplica una reducció per simular el cansanci. S'avisa a la classe 'ManagerGame' per activar totes les pedres.

```
public void GodToTired()
{
    StopInmunity();
    ChangeVel(-speedGodIncrement);
    ChangeVel(-speedTiredDecrement);
    ChangeState(State.Tired);

    ManagerGame.Instance.EnableMagicStones(true);
}
```

Figura 6.5.60 Funció 'GodToTired'

La funció 'SetAnim' (Figura 6.5.61) canvia els paràmetres del controlador depenent de l'estat. Es crida cada vegada que es canvia d'estat.

```
void SetAnim()
{
    if (actualState == State.Tired)
    {
        anim.SetBool("Tired", true);
        anim.SetBool("Attack", false);
    }
    else if (actualState == State.Normal)
    {
        anim.SetBool("Tired", false);
    }
    else if(actualState == State.God)
    {
        anim.SetBool("Attack", true);
    }
}
```

Figura 6.5.61 Funció 'SetAnim'

S'han afegit diverses animacions al controlador base (Figura 6.5.62). A part de l'animació d'atacar, s'ha afegit un 'idle' i un caminant per quan està cansat, fent més visible el cansanci del golem.

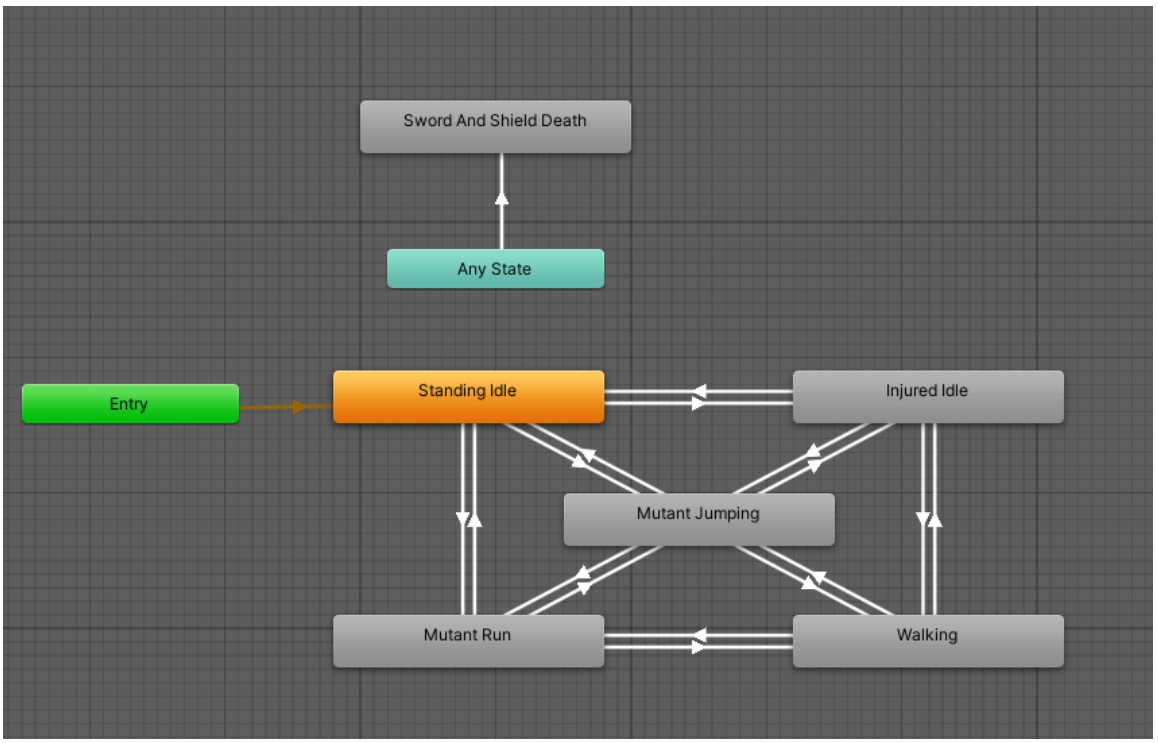


Figura 6.5.62 Controlador golem

6.5.8 Enterrador

El desavantatge de l'enterrador és que quan un enemic és eliminat en una trampa, apareix una planta que la bloqueja i no es pot tornar a utilitzar. Per aconseguir això hem d'afegir una funció en la classe 'EnemyLootController', encarregada de crear els objectes necessaris quan un enemic és eliminat.

La funció afegida es diu 'SpawnPlant' (Figura 6.5.63). Aquesta comprova si l'enemic a mort en una trampa amb la variable 'TrapOnDied' que s'assigna en eliminar l'enemic. També comprova si la trampa ja ha estat atrapada per evitar dues plantes en la mateixa zona. Per últim mira si es poden crear més plantes. Llavors invoca la planta en la posició on a mort l'enemic i li assigna la trampa per crear els esbarzers. A més, guarda la planta a una llista de 'ManagerGame' per poder eliminar-la quan toqui.

```
void SpawnPlant() //Only with Undertaker character
{
    if(trapOnDied != null && !trapOnDied.IsFrozen() && ManagerGame.Instance.CanSpawnPlant())
    {
        obj = Instantiate(plant, transform.position, Quaternion.identity);
        obj.GetComponent<Plant>().AssignTrap(trapOnDied);
        ManagerGame.Instance.SetPlant(obj.GetComponent<Plant>());
    }
}
```

Figura 6.5.63 Funció 'SpawnPlant'

Per veure si la planta es pot crear, es crida una funció de 'ManagerGame' (Figura 3.5.64) comprova si no hi ha cap ajudant a l'escena i que no se superi el màxim de plantes.

```
public bool CanSpawnPlant()
{
    UndertakerScript u = playerScript as UndertakerScript;
    return u.GetAssistantNum() <= 0 && PlantCount() < u.GetMaxAssistants();
}
```

Figura 6.5.64 Funció 'CanSpawnPlant' de 'ManagerGame'

El model de la planta ha sigut descarregat per 'Unity Assets' (Figura 6.5.65). Conté una animació per aparèixer i la mateixa animació a la inversa per desaparèixer.

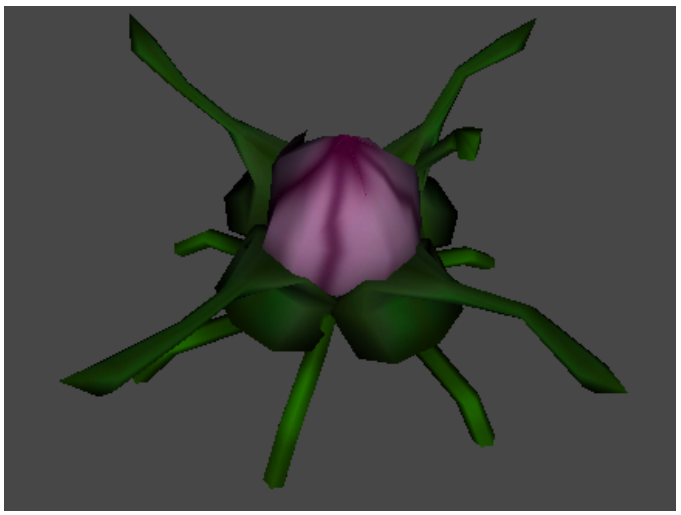


Figura 6.5.65 Model planta

La planta conté un script per fer aparèixer l'esbarzer en la mateixa posició de la trampa (Figura 6.5.66). A més, també aplica l'estat de congelat a la trampa, evitant que es recarregui. Al final l'elimina quan la planta desapareix.

```
void FrozeTrap(bool Froze)
{
    trapAttached.Froze(Froze);
    if (Froze)
    {
        brambleObj = Instantiate(bramble, trapAttached.transform.position, Quaternion.identity);
        brambleAnim = brambleObj.GetComponent<Animator>();
    }
}
```

Figura 6.5.66 Funció 'FrozeTrap'

L'esbarzer venia inclòs amb el model de la trampa (Figura 6.5.67). Igual que la planta, té una animació per aparèixer i desaparèixer, simulant sortir de sota terra.

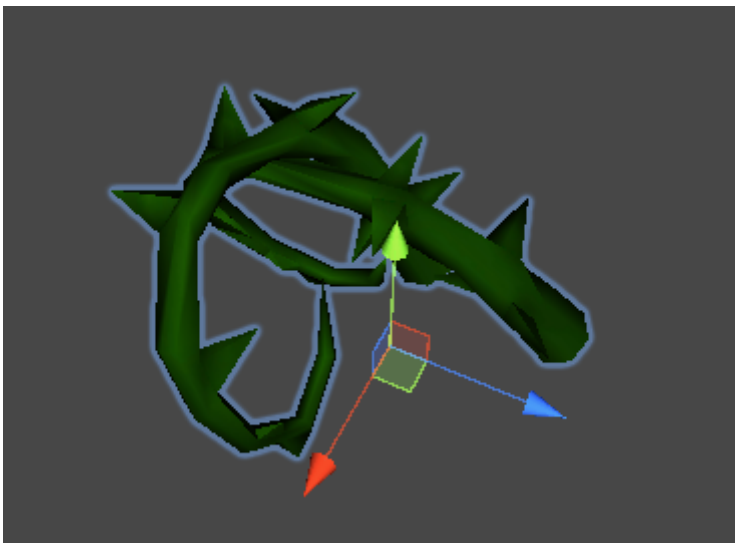


Figura 6.5.67 Model esbarzer

Aquest és el resultat final d'una trampa atrapada per una planta (Figura 6.5.68).

Figura 6.5.68 Resultat final planta

L'habilitat de l'enterrador és poder eliminar totes les plantes i fer aparèixer uns ajudants al seu voltant per cada una. Així que posem el codi corresponent a la funció 'DoPower' (Figura 3.5.69), activant el poder només si hi ha almenys una planta en el mapa.

```
protected override void DoPower()
{
  if(ManagerGame.Instance.PlantCount() > 0)
        SpawnAssistants();
}
```

Figura 6.5.69 Funcio 'DoPower'

La funció ‘SpawnAssistants’ (Figura 6.5.70) elimina totes les plantes, cridant la seva animacio abans de desaparèixer i crea els ajudants.

```
void SpawnAssistants()
{
    assistantNum = ManagerGame.Instance.PlantCount();
    ManagerGame.Instance.DestroyPlants();

    CreateEnemiesAroundPoint(assistantNum, transform.position, assistantRadius);
}
```

Figura 6.5.70 Funcio ‘SawnAssistants’

Per fer aparèixer les bombes es calcula la posició on li toca. Depenent del nombre d’enemics canvia la posició perquè rodegin totalment al personatge (Figura 6.5.71). La circumferència d'un cercle és de 2 * Pi * r, però podem ignorar el radi en aquest cas, ja que només volem la distribució al voltant del cercle, fent la funció 2 * Pi. Diguem que tenim 5 enemics. Això significa que cada enemic ha de distanciar-se del cercle (radians) de 2 * Pi / 5 l'un de l'altre. Utilitzarem num per indicar el nombre d’enemics que formen la fórmula: 2 * Pi / num. Si passéssim per cada enemic, la posició de l'actual al voltant del cercle seria la posició anterior més 2 * Pi / num. Per determinar els radians exactes podem utilitzar 2 * Pi / num * i.

Quan ja sabem la posició, fem aparèixer els enemics i els assignem un ‘GameObject’ buit anomenat ‘asGroupObject. Després l’afegim a la llista i els rota perquè mirin cap a la direcció contrària al jugador.

```
void CreateEnemiesAroundPoint(int num, Vector3 point, float radius)
{

    for (int i = 0; i < num; i++)
    {

        /* Distance around the circle */
        var radians = 2 * Mathf.PI / num * i;

        /* Get the vector direction */
        var vertical = Mathf.Sin(radians);
        var horizontal = Mathf.Cos(radians);

        var spawnDir = new Vector3(horizontal, 0, vertical);

        /* Get the spawn position */
        var spawnPos = point + spawnDir * radius; // Radius is just the distance away from the point

        /* Now spawn */
        var enemy = Instantiate(assistant, spawnPos, Quaternion.identity) as GameObject;
        enemy.transform.parent = asGroupObject.transform;
        enemy.GetComponent<MiniAssistant>().SetParent(this);
        listAssistants.Add(enemy.GetComponent<MiniAssistant>());

        /* Rotate the enemy to face towards player */
        enemy.transform.LookAt(2*spawnPos - transform.position);

        /* Adjust height */
        enemy.transform.Translate(new Vector3(0, enemy.transform.localScale.y / 2, 0));
    }
}
```

Figura 6.5.71 Funcio 'CreateEnemiesAroundPoint'

Aquest seria el resultat en el cas de apareixer quatre enemics (Figura 5.5.72).

Figura 6.5.72 Cercle ajudants

Perquè els ajudants segueixin al jugador només s'ha de canviar la posició de l'objecte buit (Figura 6.5.73) i en ser tots fills d'aquest, es mouran mantinguen la formació en cercle.

```
protected override void FixedUpdate()
{
    base.FixedUpdate();
    if (asGroupObject != null)
        asGroupObject.transform.position = transform.position;
}
```

Figura 6.5.73 Funcio 'FixedUpdate'

Quan el jugador es mogui, canviara l'animació de tots els ajudants perquè caminin (Figura 6.5.74).

```
public void AssistantWalk(bool w)
{
    for(int i=0;i<listAssistants.Count;i++)
    {
        listAssistants[i].Walk(w);
    }
}
```

Figura 6.5.74 Funcio 'AssistantWalk'

Per últim, quan un ajudant mort es treu de la llista (Figura 6.5.75). Si el nombre d'ajudants arriba a 0 i ja ha passat el temps per tornar a activar l'habilitat, es destrueix l'objecte buit i es permet utilitzar el poder de nou.

```
public void AssistantDied(MiniAssistant go)
{
    assistantNum--;
    listAssistants.Remove(go);
    if (assistantNum <= 0 && habilityCount >= habilityCD)
    {
        Destroy(asGroupObject);
        asGroupObject = null;
        EnablePower();
    }
}
```

Figura 6.5.75 Funcio 'AssistantDied'

El model de l'ajudant s'ha descarregat de 'UnityAssets' (Figura 6.5.76). Té un 'trigger' per detectar quan un enemic és a prop i així activar-se per explotar. A més, té una línia al voltant per saber quan està explotant.

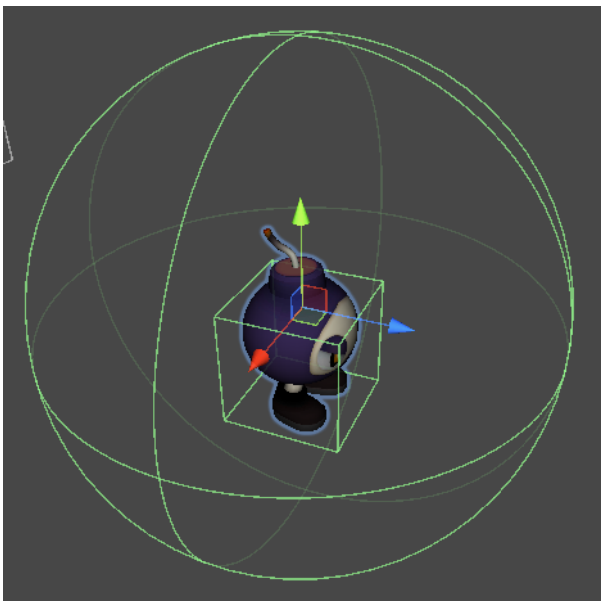


Figura 6.5.76 Model ajudant

Afegim la funció 'OnTriggerEnter' per activar-se quan un enemic col·lideix amb la zona (Figura 6.5.77). També s'aplica el color vermell a la línia del voltant i s'assigna 'true' a la variable que detecta quan s'ha d'atacar.

```
private void OnTriggerEnter(Collider other)
{
    if(other.tag == "Enemigo")
    {
        enemyTarget = true;
        outl.OutlineColor = Color.red;
    }
}
```

Figura 6.5.77 Funcio 'OnTriggerEnter'

Quan ha detectat un enemic, activa un comptador (Figura 6.5.78) per explotar passat un cert temps.

```
void Update()
{
    if (enemyTarget)
    {
        lifeCount += Time.deltaTime;
        if (lifeCount > lifeTime)
        {
            lifeCount = 0;
            enemyTarget = false;
            Explosion();
        }
    }
}
```

Figura 6.5.78 Funcio 'Update'

Quan explota activa l'animació que toca i destrueix l'objecte passar un temps (Figura 6.5.79).

```
void Explosion()
{

    anim.SetTrigger("attack01");
    StartCoroutine(DestroyObject());
}
```

Figura 6.5.79 Funcio 'Explosion'

6.5.9 Models i animacions

Tots els models dels personatges han sigut descarregats per 'Sketchfab' o 'Unity Assets', així com les textures corresponents (Figura 6.5.80).

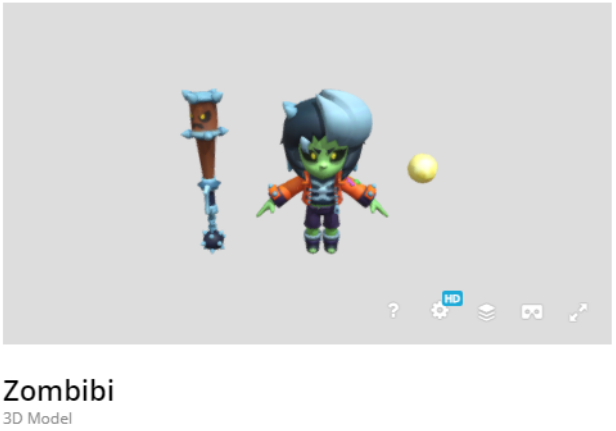


Figura 6.5.80 Model Zombibi

Seguidament, s'han passat a Blender per eliminar els accessoris i treure el 'Rigging' ja creat, per tal de poder pujar-ho a Mixamo sense problema. Per acabar, s'han exportat en format '.Obj' i s'han pujat a Mixamo (Figura 6.5.81), on s'han triat i descarregat les animacions que es necessitaven.

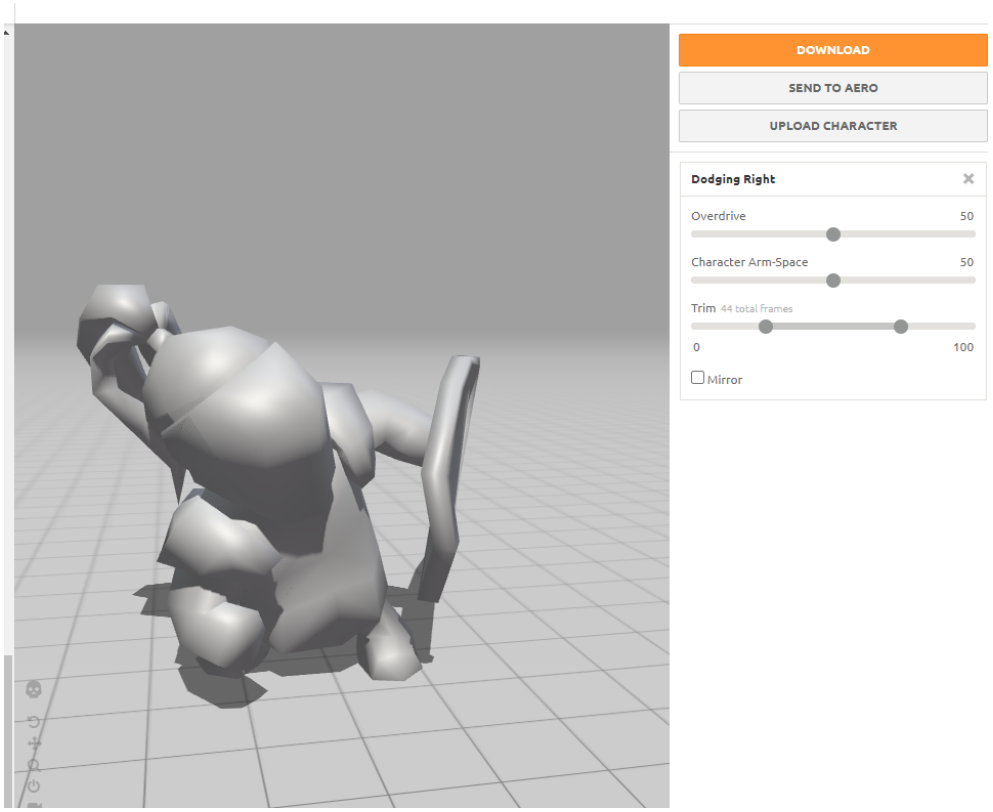


Figura 6.5.81 Exemple Mixamo

Una vegada amb totes les animacions, s'han aplicat sobre el model i s'ha creat el controlador que les conté (Figura 6.4.82). Cada Personatge té un controlador diferent assignat adaptat a les seves necessitats.

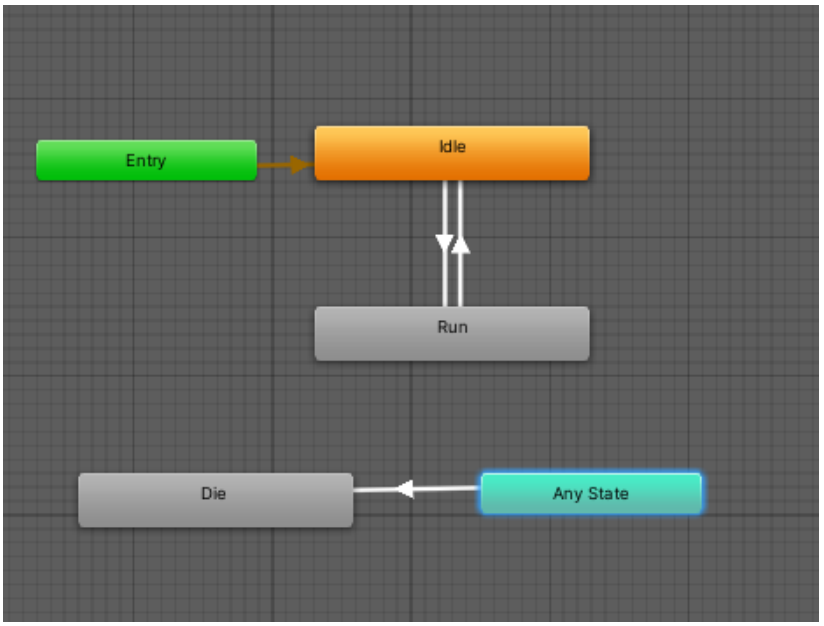


Figura 6.5.82 Controlador animacions base

Cada personatge té un cercle transparent que simula l'escut que t'atorga una de les bonificacions. Aquest es modifica a mà per tal de quadrar amb el model (Figura 6.5.83) i seguidament es desactiva. Quan s'agafa el poder, activem el cercle i tenim l'escut protector.

Figura 6.5.83 Escut protector

6.6 Enemics

6.6.1 Base

Moviment

Tots els enemics tenen assignat el component 'NavMeshAgent' (Figura 6.6.1). Aquest s'encarrega del moviment per l'escenari esquivant els obstacles i triant el camí més ràpid fins al seu objectiu. El component conté uns paràmetres per configurar la seva velocitat, velocitat angular i acceleració.

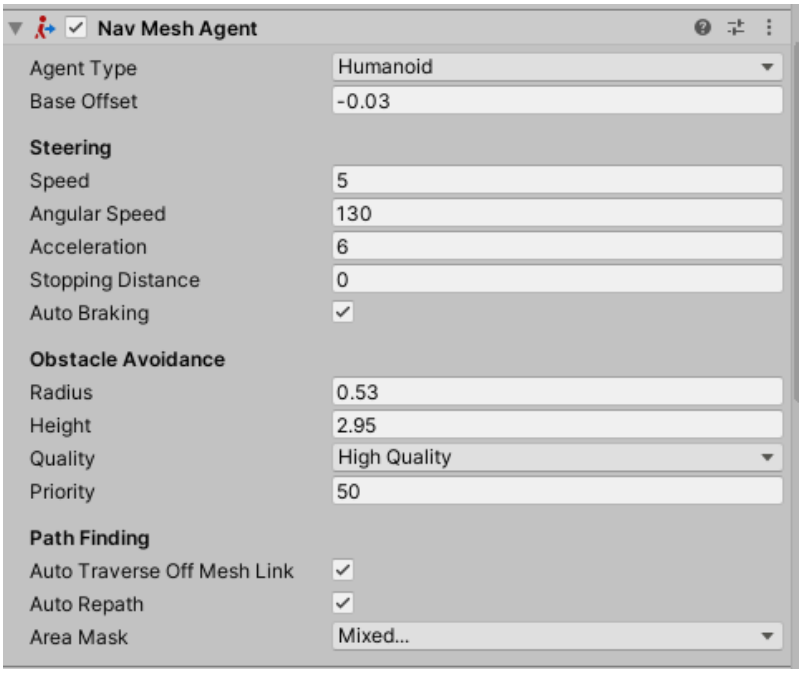


Figura 6.6.1 Component NavMeshAgent

Després, només s'ha de referenciar el 'NavMeshAgent' per indicar a quina posició ha d'anar (Figura 6.6.2). Li assignem la posició del jugador.

```
case States.Walking:
    if (Player != null)
        nav.SetDestination(Player.position);
    break;
```

Figura 6.6.2 Codi 'EnemyController'

Vida

El sistema de vida és semblant al programat en el jugador. A la funció 'OnTriggerEnter' (Figura 6.6.3) comproven si entra en contacte amb alguna trampa i infligim mal. A més, guarda la trampa per si s'ha d'interactuar amb ella en un futur.

```
private void OnTriggerEnter(Collider other)
{
    if(other.transform.tag == "Daño")
    {
        GetDamage(1);

        if (other.GetComponent<DamageZone>() != null)
        {
            trapHit = other.GetComponent<DamageZone>().GetOriginTrap();
        }
        else
            trapHit = null;
    }
}
```

Figura 6.6.5 Funció 'OnTriggerEnter'

La funció 'GetDamage' (Figura 6.6.4) disminueix la vida de l'enemic i si arriba a zero l'elimina. Si no, l'atordeix durant un temps.

```
protected virtual void GetDamage(int damage)
{
    life -= damage;
    if(life <= 0)
    {
        Death();
    }
    else
    {
        Stun(true);
    }
}
```

Figura 6.6.4 Funció 'GetDamage'

Quan l'enemic queda atordit es para el component 'NavMeshAgent' perquè deixi de seguir al jugador i canvia d'estat (Figura 6.6.5).

```
public void Stun(bool stunned) //Called
{
    if (stunned)
    {
        nav.isStopped = true;
        actualState = States.Stunned;
    }
    else
    {
        nav.isStopped = false;
        actualState = States.Walking;
    }
}
```

Figura 6.6.5 Funció 'Stun'

Es compra el temps que s'està atordit i es torna a estat normal quan arriba al temps límit entrat per paràmetre (Figura 6.6.6).

```
    break;
case States.Stunned:
    countStunned += Time.deltaTime;
    if (countStunned >= stunnedTime)
    {
        countStunned = 0;
        Stun(false);
    }
    break;
```

Figura 6.6.6 Codi 'EnemyController'

Que l'enemic és eliminat, avisa a la classe 'ManagerGame' per actualitzar la llista d'enemics actius. Després, s'assigna l'animació corresponent, es para el 'NavMeshAgent' i es canvia l'estat (Figura 6.6.7).

```
public virtual void Death() //Called by Manager(when DeathPowerU
{
    if(actualState != States.Dying)
    {
        ManagerGame.Instance.EnemyDeath();
        anim.SetTrigger("Dead");
        nav.isStopped = true;
        col.enabled = false;
        actualState = States.Dying;
    }
```

Figura 6.6.7 Funcio 'Death'

Després de passat cert temps, s'avisa de la mort per calcular el combo i es destrueix l'objecte (Figura 6.6.8).

```
case States.Dying:

    countDestroying += Time.deltaTime;
    if (countDestroying >= secondsBeforeDestroy)
    {
        AnounceDeath();
        DestroyEnemy();
    }

    break;
```

Figura 6.6.8 Codi 'EnemyController'

Tots els models han sigut descarregats per separat o per paquets a través de 'UnityAssets' (Figura 6.6.9) i després col·locats dintre del joc.

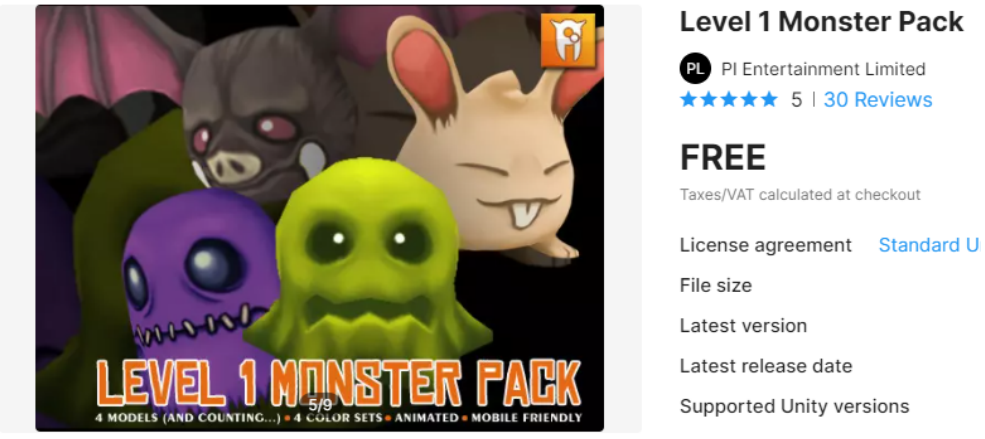


Figura 6.6.10 Pack UnityAssets

Mantinguin l'estructura bàsica, s'han creat un parell d'enemics amb diferents paràmetres però amb el mateix codi. Després per cada enemic especial s'ha creat un script que hereta de la base i s'adapta depenent de l'habilitat única d'aquests.

6.6.2 Esquelet

L'habilitat especial dels esquelets és que poden reviure quan són eliminats fins que el jugador els trepitgi. Per aconseguir això canviem la funció 'Death' (Figura 6.6.11), afegint una variable per saber que està revivint.

```
public override void Death()//Call
{
    anim.SetBool("Dead", true);
    nav.isStopped = true;
    reviving = true;
}
```

Figura 6.6.11 Funció 'Death'

A la funció 'Update' (Figura 6.6.12) augmenta un comptador que arribat a cert límit, reviu a l'enemic.

```
protected override void Update()
{
    base.Update();
    if(reviving)
    {
        countReviving += Time.deltaTime;
        if (countReviving >= timeToRevive)
        {
            countReviving = 0;
            reviving = false;
            Revive();

        }
    }
}
```

Figura 6.6.12 Funció 'Update'

Quan reviu, s'activa l'animació corresponent i es torna a activar el component 'NavMeshAgent' (Figura 6.6.13).

```
void Revive()
{
    anim.SetBool("Dead",false
    nav.isStopped = false;
    reviving = false;
}
```

Figura 6.6.13 Funció 'Revive'

Al controlador d'animació té afegit una animació de resurrecció. Aquesta és la animacio de mort invertida, simulant que es torna a aixecar i torna al atac (Figura 6.6.14).

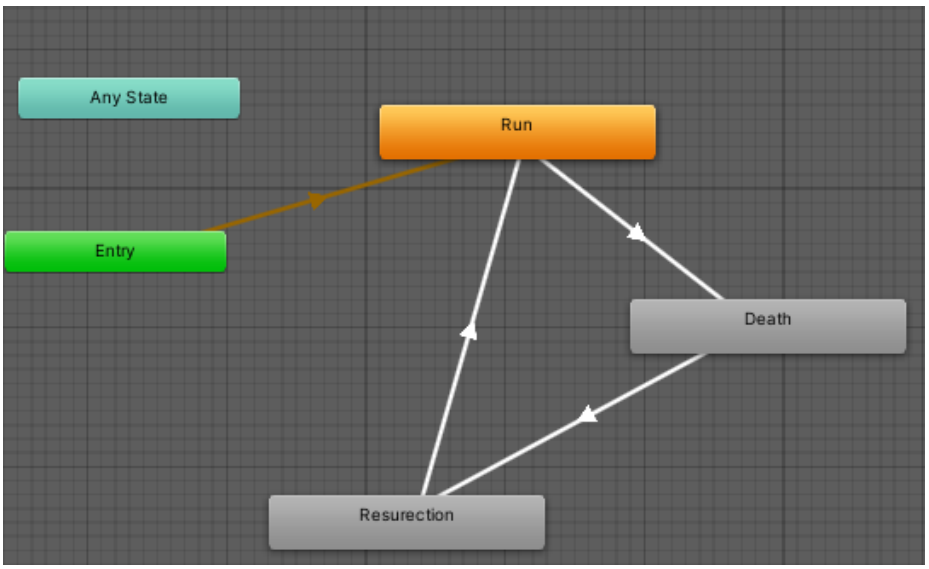


Figura 6.6.14 Controlador esquelet

Quan el jugador entra en contacte amb l'esquelet mentre aquest està revivint, el mata definitivament (Figura 6.6.15), executant el codi base de la funció 'Death().

```
public void PermaDeath()//
{
    reviving = false;
    base.Death();
}
```

Figura 6.6.15 Funció 'PermaDeath'

6.6.3 Fantasma

El fantasma es fa transparent cada cert temps, sent difícil de detectar i fent-se invulnerable. Per començar, s'ha afegit codi a la funció 'Update' (Figura 6.6.16) per comptar quan ha d'estar invisible i quan no. Això es calcula amb un comptador que quan arribi al límit, canviaran l'estat de l'enemic.

```
protected override void Update()
{
    base.Update();
    invCount += Time.deltaTime;
    if (!invencible)
    {
        if(invCount>timeToInvecible)
        {
            invCount = 0;
            invencible = true;
            getInvisible(invencible);
        }
    }
    else
    {
        if (invCount > invencibleDuration)
        {
            invCount = 0;
            invencible = false;
            getInvisible(invencible);
        }
    }
}
```

Figura 6.6.16 Funció 'Update'

A continuació a la funció 'GetDamage' (Figura 6.6.17) només es farà mal si no està en estat invencible.

```
protected override void GetDamage(int damage)
{
    if(!invencible)
        base.GetDamage(damage);
}
```

Figura 6.6.17 Funció 'GetDamage'

Per aconseguir mostrar al jugador que el fantasma és invisible, s'ha creat una animació que disminueix la transparència del material (Figura 6.6.18). Per fer-ho, s'han creat dues capes diferents. Per un costat s'executen les animacions de caminar i morir, i a la vegada les animacions de transparència (Figura 6.6.19)

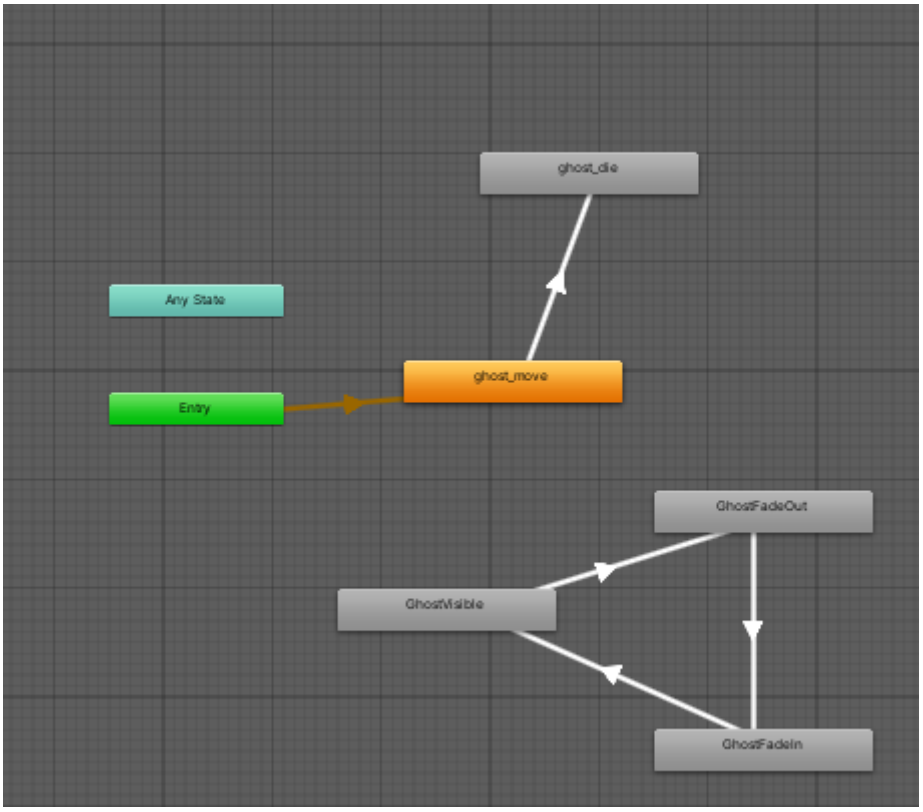


Figura 6.6.18 Controlador fantasma

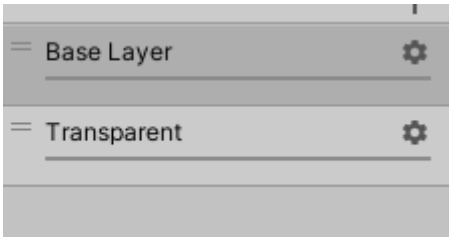


Figura 6.6.19 Capes de animacions fantasma

6.6.4 Slime

Els 'slimes' es divideixen en dos quan moren per primer cop. S'ha creat una còpia del 'slime' pare però de mida reduïda. Apareixeran dues d'aquestes còpies quan s'elimini el pare. Com comparteixen el mateix codi, per saber si s'ha de dividir es comprova una variable anomenada 'numOfDiv' quan es destrueix l'enemic (Figura 6.6.20). Si és 0, vol dir que és una de les còpies i s'elimina permanentment. Si és més gran, es creen les còpies.

```
protected override void DestroyEnemy()
{
    if(numOfDiv == 0)
    {
        base.DestroyEnemy();
    }
    else
    {
        SpawnChildren();
        Destroy(gameObject);
    }
}
```

Figura 6.6.20 Funció 'DestroyEnemy'

La funció 'SpawnChildren' (Figura 6.6.21) executa un procés per cada fill que es vol crear, per defecte 2. Es crea en una posició al voltant del pare i s'assigna la variable 'numOfDiv' com a la del pare menys 1. Així, en cas que es volgués dividir diversos cops, ho faria fins a arribar a 0. En el cas actual només es divideix una vegada. Per acabar, l'afegeix a la llista d'enemics de 'ManagerGame'.

```
void SpawnChildren()
{
    GameObject ob;
    SlimeController sl;
    EnemyController en;
    float rand1, rand2;
    for(int i=0;i<numOfChild;i++)
    {
        rand1 = Random.Range(xMin, xMax);
        rand2 = Random.Range(zMin, zMax);

        ob = Instantiate(slime, new Vector3(transform.position.x + rand1, transform.position.y,
            transform.position.z + rand2), transform.rotation);

        sl = ob.GetComponent<SlimeController>();
        en = sl as EnemyController;

        sl.Inicialize(numOfDiv - 1);
        ManagerGame.Instance.NewEnemy(en);
    }
}
```

Figura 6.6.21 Funció 'SpawnChildren'

6.7 EnemyLootController

Quan un enemic és eliminat, crida a la classe 'EnemyLootController' per crear el premi corresponent. Aquest conté un 'array' amb un premi per cada posició. En un altre 'array' es guarda la probabilitat de què aparegui el premi de la posició corresponent. És a dir, la probabilitat de la posició 1 correspon al premi de la posició 1. Quan mes a prop de la posició 0, més probabilitat de apareixer. A les posicions més altes hi ha els millors premis, tenint una baixa probabilitat d'aparèixer. A la posició 0 no hi ha cap premi per defecte. Després, depèn del nivell de combo, es canvia les posicions dels premis per incrementar la probabilitat dels premis més alts.

La funció 'Loot' (Figura 6.7.1) s'encarrega de tot aquest procés. Primer obté el combo actual i configura el nivell del premi. Després, genera un nombre aleatori i comprova a quin premi correspon. Seguidament el crea en el cos de l'enemic. Finalment, genera una planta o una clau depenent del personatge triat.

```
//Spawn coin depends on the probability of each type of coin.
public void Loot()
{
    GetCombo();
    ConfigLevel();

    spawnPos = new Vector3(transform.position.x, transform.position.y + 2, transform.position.z);
    int rand = Random.Range(1, 101);
    if (rand > 0 && rand <= coinProb[0])
            { Instantiate(coinsToSpawn[1], spawnPos, Quaternion.identity); }
    else if (rand > coinProb[0] && rand <= coinProb[0] + coinProb[1] + 1)
            { Instantiate(coinsToSpawn[2], spawnPos, Quaternion.identity); }
    else if (rand > coinProb[0] + coinProb[1] + 1 && rand <= coinProb[0] + coinProb[1] + coinProb[2] + 2)
            { Instantiate(coinsToSpawn[3], spawnPos, Quaternion.identity); }
    else
    {
        if (coinsToSpawn[0] != null)
        {
            Instantiate(coinsToSpawn[0], spawnPos, Quaternion.identity);
        }
    }

    //Extra: King
    if (classTried == GlobalVariables.CharactersTypes.King) { SpawnKey(); }
    if (classTried == GlobalVariables.CharactersTypes.Undertaker) { SpawnPlant(); }
}
```

Figura 6.7.1 Funció 'Loot'

Per configurar el nivell del premi s'obre el combo actual i es comprova amb un 'array' que conte el combo mínim per pujar de nivell a cada posició (Figura 6.7.2). Per cada posició que superi el combo, més puja el nivell. A continuació disminueix en 1 la posició dels premis per cada nivell obtingut. Per exemple, amb un nivell 3, el premi de la posició 4 es mou a la 1. Amb això s'aconsegueix que a major combo, més probabilitat que apareguin els millors premis, ubicats a les posicions altes. Els premis de les posicions baixes no poden aparèixer.

```
void ConfigLevel() //Change lvl if combo it's bigger than the minimum
                           //and increase the reward probability depends on the lvl.
{
    for(int j=0;j<comboMinimum.Length;j++)
    {
        if(combo >= comboMinimum[j])
        {
            lvlOfLoot++;
        }
    }
    for(int i=0;i<coinsToSpawn.Length;i++)
    {
        int level = i + lvlOfLoot;
        if (level >= coinsToSpawn.Length) level = coinsToSpawn.Length - 1;
        coinsToSpawn[i] = coins[level];
    }
}
```

Figura 6.7.2 Funció 'ConfigLevel'

S'adjunta un exemple del canvi de nivell de premi (Figura 6.7.3).

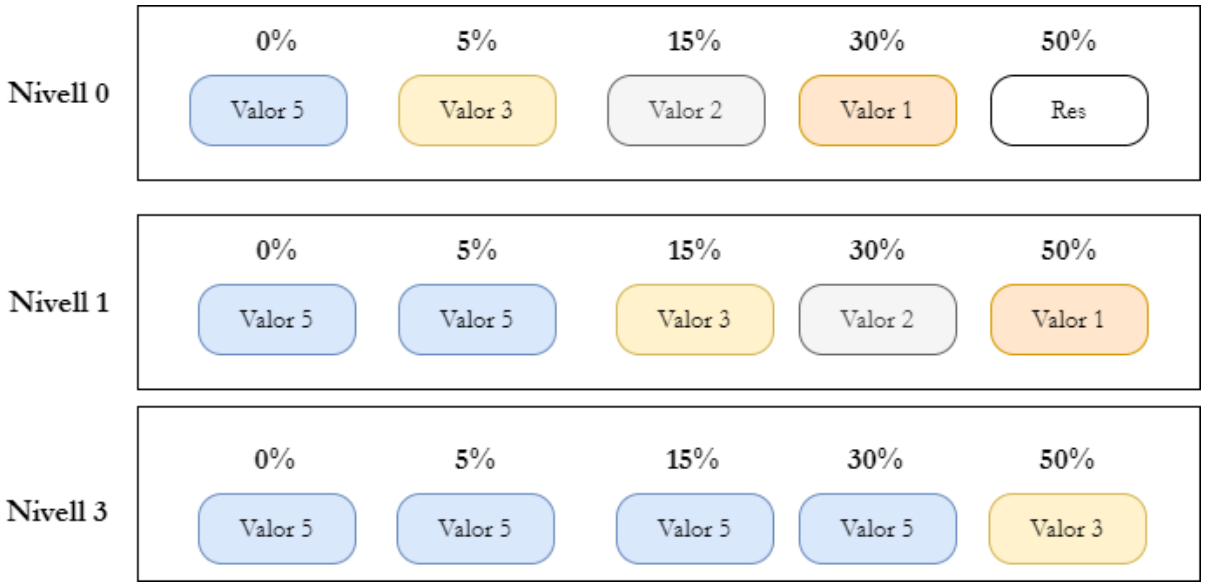


Figura 6.7.3 Exemple Loot

6.8 Monedes

Les monedes tenen una animació que les fa rotar constantment perquè no quedin molt estàtiques quan apareixen. A més, cada una té un script adjuntat (Figura 6.8.1) el qual els hi dona un valor i un temps de vida. Passat aquest temps desapareixen de la partida.

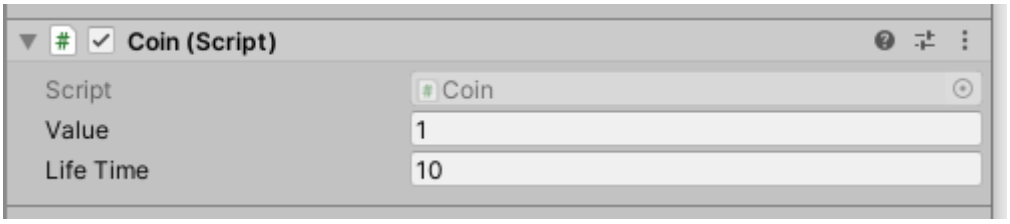


Figura 6.8.1 Script 'Coin'

Les monedes comparteixen el mateix model. S'ha descarregat a través de 'Unity Assets' i s'ha editat el color de la textura per poder tenir els diferents tipus (Figura 6.8.2).

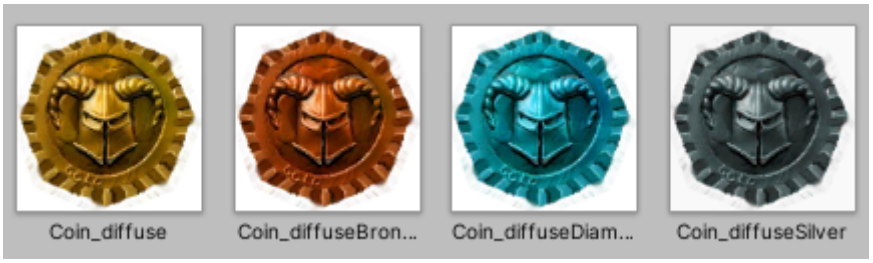


Figura 6.8.2 Textures monedes

Amb l'objectiu de facilitar la recollida de les monedes, s'ha creat un script anomenat 'CoinAtractor'. Aquest script s'adjunta a un 'Game Object' buit, el qual és fill de l'objecte moneda. Conté un collider que quan detecta a un jugador, mou la moneda cap a ell a una velocitat establida per paràmetre (Figura 6.8.3).

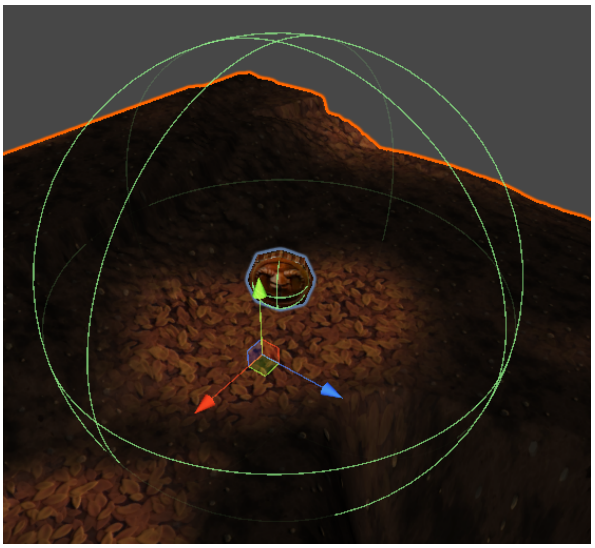


Figura 6.8.3 Àrea collider de 'CoinAtractor'

6.9 Trampes

6.9.1 Base

Les trampes s'han programat en un sistema d'estats. Aquests són: Esperant, activant i recarregant. Al començament es queda esperant que algu l'activi. En el moment que detecta al jugador es prepara per atacar i

després d'un cert temps, ataca. A continuació es queda inactiva durant un interval i torna a esperar. Tot el codi s'ha separat en funcions i s'han cridat en la funció 'Update' (Figura 6.9.1).

```
protected virtual void Update()
{
    switch (actualState)
    {
        case States.Waiting:
            Waiting(); //Waiting to Active
            break;
        case States.Activating:
            Activating(); //Preparing to attack
            break;
        case States.Reloading:
            Reloading(); //Waiting to restart
            break;
    }
}
```

Figura 6.9.1 Funció 'Update'

Quan un jugador és detectat per l'àrea de la trampa, aquesta canvia d'estat i activa l'animació corresponent (Figura 6.9.2).

```
protected virtual void Activation() //Playe
{
    if(actualState == States.Waiting)
    {
        actualState = States.Activating;
        ChangeAnim();
    }
}
```

Figura 6.9.2 Funció 'Activation'

Passat un temps entrat per paràmetre activa l'àrea que conté l'etiqueta 'Daño' per fer mal a tot personatge dins d'aquesta (Figura 6.9.3). Seguidament es passa a estat de recarregat.

```
protected virtual void Activating()
{
    activationCount += Time.deltaTime;
    if (activationCount >= timeToActive)
    {
        activationCount = 0;
        Activate();
    }
}
```

Figura 6.9.3 Funció 'Activating'

Si no està congelada, compta el temps i es recarrega quan arriba a cert límit, passant a estat d'espera (Figura 6.9.4).

```
protected virtual void Reloading()
{
    if(!frozen)
    {
        reloadingCount += Time.deltaTime;
        if (reloadingCount >= activationCD)
        {
            Reloaded();
            reloadingCount = 0;
        }
    }
}
```

Figura 6.9.4 Funció 'Reloading'

Les trampes tenen una àrea petita que detecta quan el jugador entra a la zona (Figura 6.9.5). L'àrea gran indica la zona que farà mal. S'activa un altre objecte indicat en vermell (Figura 6.9.6). Aquestes àrees estan ocultes quan es juga.

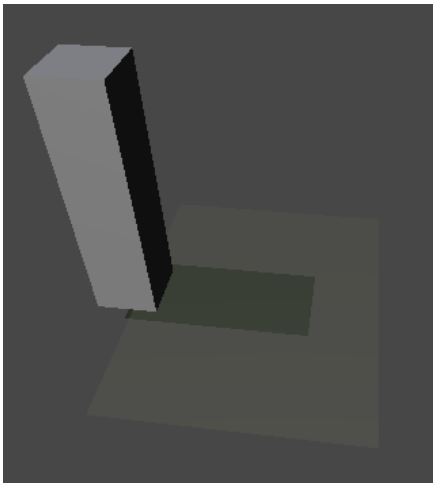


Figura 6.9.5 Trampa base

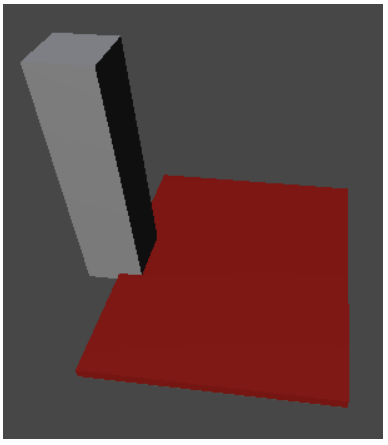


Figura 6.9.6 Trampa base atacant

A causa de les bonificacions, s'ha afegit una funció pública per canviar la velocitat de recuperació de les trampes (Figura 6.9.7). A més, activa una partícula per avisar al jugador mentre estan millorades.

```
//////PUBLIC FUNCTIONS//////
public void changeVel(float vel) //Called by: PowerUpManager(on
{
    if (powerUpPArticle != null)
    {
        if (powerUpPArticle.isPlaying) powerUpPArticle.Stop();
        else powerUpPArticle.Play();
    }

    activationCD = activationCD * vel;
}
```

Figura 6.9.7 Funció 'changeVel'

S'ha afegit una línia que envolta el model que canvia de color depenent de l'estat que es troba. Això es canvia a la variable 'ChangeAnim' (Figura 6.9.8). Es crida cada vegada que canvia d'estat.

```
protected virtual void ChangeAnim()
{
    switch (actualState)
    {
        case States.Waiting:
            outl.OutlineColor = Color.white;
            state = 0;
            break;
        case States.Activating:
            outl.OutlineColor =Color.red;
            state = 1;
            break;
        case States.Reloading:
            outl.OutlineColor = new Color(0, 0, 0, 0);
            state = 2;
            break;
    }

    if (anim != null) anim.SetInteger("State", state);
}
```

Figura 6.9.8 Funció 'ChangeAnim'

El controlador és pràcticament igual per totes les trampes, passant per tots els estats en ordre (Figura 6.9.9).

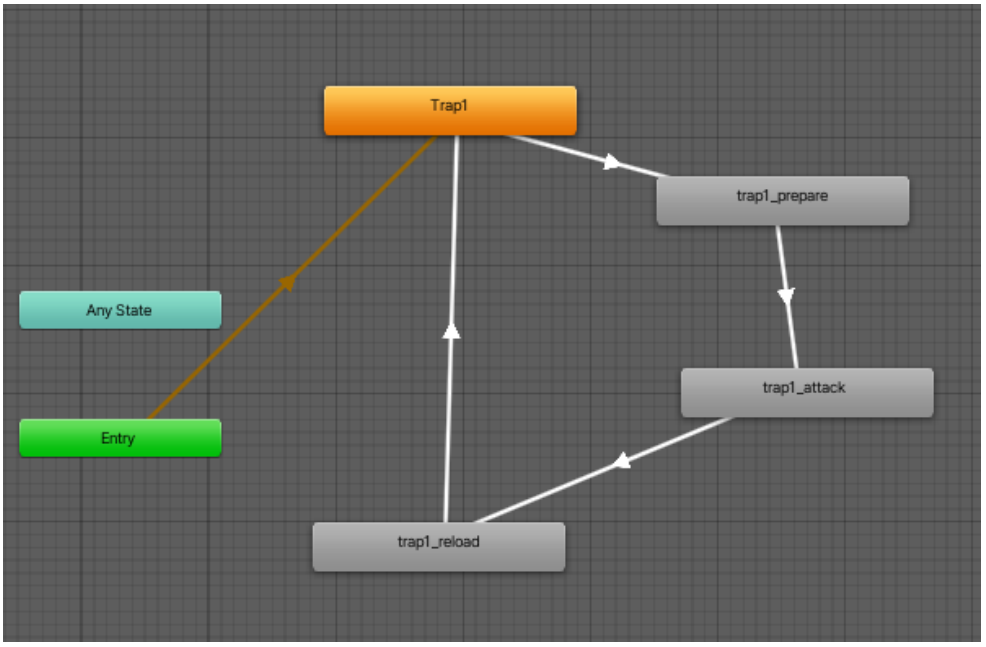


Figura 6.9.9 Controlador trampes

La majoria de trampes s'han extret de 'UnityAssets'. Algunes han sigut modelades amb Blender, però no totes s'han pogut incluir en el joc per falta de temps. La trampa bàsica anava a ser un esbarzer que forma part del bosc (Figura 6.9.10). La trampa que es mou anava a ser un espantaocells que et persegueix (Figura 6.9.11).

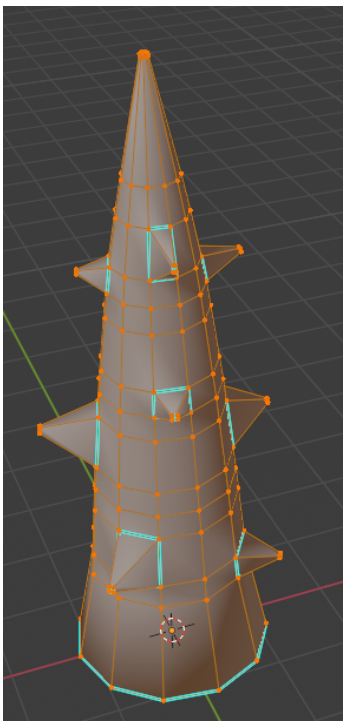


Figura 6.9.10 Model trampa base

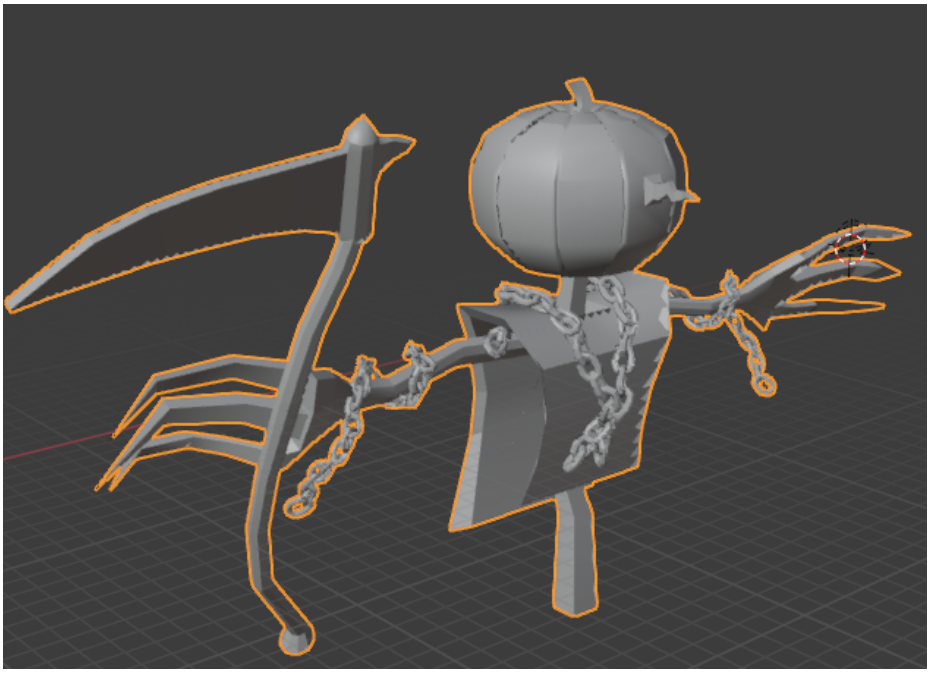


Figura 6.9.11 Model trampa mòbil.

6.9.2 Mòbil

La trampa mòbil et segueix mentre es prepara per atacar, acabant en una explosió en àrea. S'ha canviat la funció d'activació per fer que es mogui fins a la posició del jugador (Figura 6.9.12).

```
protected override void Activating()
{
    base.Activating();
    if(target != null)
    {
        float step = speed * Time.deltaTime; // calculate distance to move
        transform.position = Vector3.MoveTowards(transform.position, target.position, step);
    }
}
```

Figura 6.9.12 Funció 'Activating'

6.9.3 Rotatòria

La trampa rotatòria gira quan no està recarregant, fent mal a tot al seu voltant (Figura 6.9.13).

```
protected override void Update()
{
    base.Update();
    if(actualState != States.Reloading) transform.Rotate(0, rotSpeed * Time.deltaTime, 0);
}
```

Figura 6.9.13 Funció 'Update'

6.9.4 Doble

La trampa doble ataca dues vegades abans de començar a recarregar. S'ha canviat la funció 'Activate' (Figura 6.9.14), afegint un comptador d'atacs i només recarregant quan aquest comptador ha arribat al límit establert.

```
protected override void Activate() //Trap attacks
{
    anim.SetInteger("State", 2);

    StartCoroutine(ActivateZone());

    attackCount++;
    //if (anim != null) anim.SetInteger("State", 1);
    if (attackCount >= numOfAttacks)
    {
        anim.SetBool("Twice", false);
        attackCount = 0;
        StartReload();
    }
    else
    {
        anim.SetBool("Twice", true);
    }
}
```

Figura 6.9.14 Funció 'Activate'

Al controlador s'ha afegit una connexió per poder passar de recarregat a preparat per atacar de nou (Figura 6.9.15).

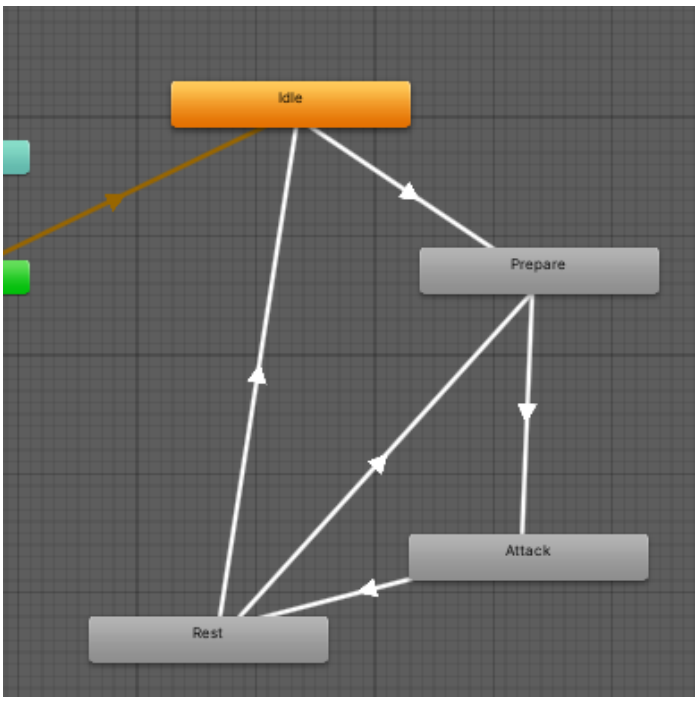


Figura 6.9.15 Controlador trampa doble

El model d'aquesta trampa ha sigut modelat amb Blender. És una planta carnívora que pertany al bosc (Figura 6.9.16).

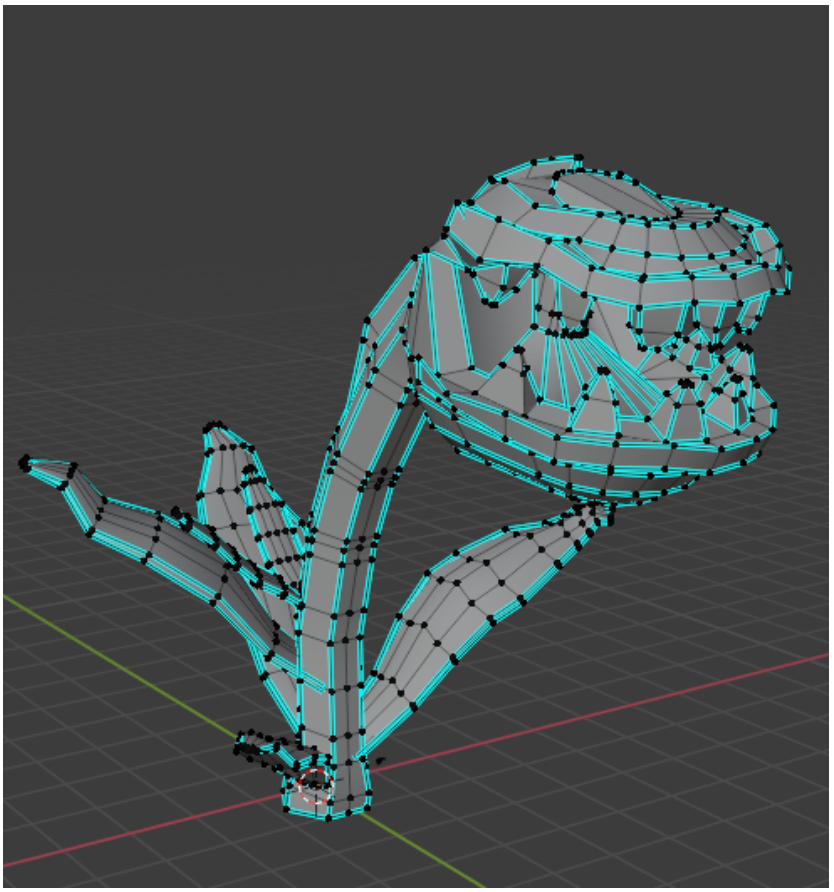


Figura 6.9.16 Model trampa doble

6.9.5 Disparar

Aquesta trampa no activa una zona, sinó que dispara un projectil amb l'etiqueta 'Daño' que pot eliminar un enemic. S'ha canviat el codi de la funció 'Activate' (Figura 6.9.17) per crear un projectil a la posició de la trampa.

```
protected override void Activate() //Trap attacks
{
    StartReload();
    bl = Instantiate(bullet,transform.position,Quaternion.identity);
    bl.GetComponent<Bullet>().Inicialize(transform.forward);
}
```

Figura 6.9.17 Funció 'Activate'

Aquesta trampa no arriba mai a l'estat d'espera, ataca constantment. Per això, una vegada recarregat torna a l'estat de preparat per atacar (Figura 6.9.18).

```
protected override void Reloaded() //Trap Reloaded
{
    actualState = States.Activating;
    ChangeAnim();
}
```

Figura 6.9.18 Funció 'Reloaded'

Pel projectil s'ha creat un cercle amb decoració simple per simular una bola amb pinxos (Figura 6.5.5). Aquesta te un temps de vida per si no toca cap personatge.

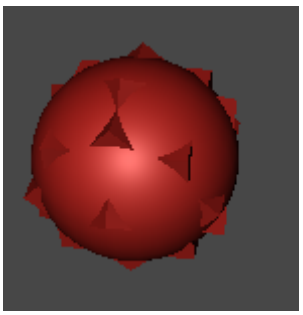


Figura 6.5.5 Model projectil

Per falta de temps aquesta trampa no s'ha implementat dintre de la partida, però és funcional.

6.10 Cofres

Els cofres apareixen al costat del jugador només començar una partida. Aquest tenen un comptador que cada cert número, pugen de nivell. Quan s'obren donen una recompensa depenent del nivell en el qual es troben. Per aconseguir això, creem un script 'Chest' per tenir el codi base i després cada tipus de cofre heretara d'aquest, fent els canvis necessaris per adaptar-se.

La funció publica 'IncrementPunt' (Figura 6.10.1) augmenta el comptador del cofre i en cas de superar el límit, establert per un 'array, puja de nivell.

```
public void IncrementPunt() //Called by: Manager(On Kill Enemy)
{
    if (!opened && (lvl < numOfPunt.Length))
    {
        punt++;
        puntTxt.text = punt.ToString();
        if (punt >= numOfPunt[lvl])
        {
            LevelUp();
        }
    }
}
```

Figura 6.10.1 Funcio 'IncrementPunt'

Per pujar de nivell simplement s'augmente la variable que el guarda i aplica el color corresponent passat per paràmetre (Figure 6.10.2).

```
void LevelUp()
{
    lvl++;
    outl.OutlineColor = lvlColors[lvl];
}
```

Figura 6.10.2 Funcio 'LevelUp'

Quan s'obre el cofre es reinicia el nivell posant la variable a 0 i aplicant el color que pertoca (Figure 6.10.3).

```
void ResetLvl()
{
    lvl = 0;
    outl.OutlineColor =  lvlColors[lvl];
    punt = 0;
    puntTxt.text = punt.ToString();
    if (punt >= numOfPunt[lvl])
    {
        LevelUp();
    }
}
```

Figura 6.10.3 Funcio 'ResetLvl'

La funció 'GiveReward' (Figura 6.10-4) serà la que canvii entre els diferents cofres, donant diferents recompenses. El codi en comú activa una partícula quan el nivell és 0 i no dona cap recompensa.

```
protected virtual void GiveReward() {
    if (lvl == 0) { part.Play(); }
}
```

Figura 6.10.4 Funcio 'GiveReward'

Perquè s'obri el jugador haurà de fer contacte amb un petit 'trigger' col·locat davant del cofre (Figure 6.10.5).

```
private void OnTriggerEnter(Collider other)
{
    if(other.tag == "Player")
    {
        Open();
    }
}
```

Figura 6.10.5 Funcio 'OnTriggerEnter'

Hi ha dos tipus diferents de cofres, el cofre de bonificacions i el de diners. El de bonificacions és el que apareix per defecte, augmentant el comptador quan s'elimina un enemic i atorgant diferents bonificacions depenent del nivell. El cofre de diners apareix només quan el personatge és el rei i augmenta el comptador agafant claus que apareixen pel mapa, donant una quantitat de diners segons nivell.

Els dos cofres comparteixen model i tenen un canvas col·locat en el front (Figura 6.10.6). Al cofre de diners se li afegeix una textura per simular els diners.

Figura 6.10.6 Models cofres

El cofre de bonificacions conté una llista amb 'arrays' d'objectes, cada un siguent els poders que poden aparèixer per aquell nivell. Es crida la funció 'SpawnPower' perquè crei un poder aleatori del nivell corresponent (Figura 6.10.7).

```
protected override void GiveReward()
{
    base.GiveReward();

    if (lvl > 0)
    {
        SpawnPower(lvlList[lvl - 1]);
    }
}
```

Figura 6.10.7 Funcio 'GiveReward' de 'KillChest'

Es tria un poder aleatori i es crea a la mateixa posició del cofre (Figure 6.10.8).

```
void SpawnPower(GameObject[] items)
{
    if (items.Length > 0)
    {
        int rand = Random.Range(0, items.Length);
        Instantiate(items[rand], transform.position, Quaternion.identity);
    }
}
```

Figura 6.10.8 Funcio 'SpawnPower'

En el cas del cofre de diners, simplement incrementa els diners depenent del nivell, agafant la dada d'un array emplenat per paràmetre (Figura 6.10.9).

```
protected override void GiveReward()
{
    base.GiveReward();
    ManagerGame.Instance.IncreaseMoney(coinValue[lvl]);
    coins.SetActive(lvl > 0);
}
```

Figura 6.10.9 Funcio 'GiveReward' de 'KeyChest'

6.11 Bonificacions

Totes les bonificacions tenen adjunt un script que simplement conté el tipus de poder que és (Figura 6.11.1), agafant el tipus de la classe 'GlobalVariables' explicada al principi de l'apartat. Quan entra en contacte amb el jugador, desapareix i deixa la feina a la classe PowerUpManager, que s'encarrega de fer els canvis corresponents.

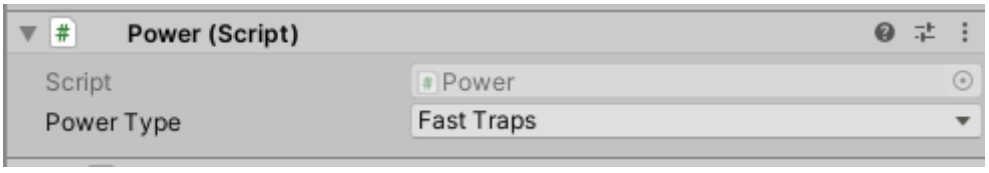


Figura 6.11.1 Script 'Power'

Per l'art, s'ha descarregat per 'Unity Assets' i s'han modificat les textures per poder tenir diferents colors (Figura 6.11.2) i fer cada bonificació diferenciable.

Figura 6.11.2 Textures poders

A més, també es conta amb diferents models (Figura 6.10.3) per evitar la repetició.

Figura 6.11.3 Diferents bonificacions

6.12 Efectes

6.12.1 Llum

Amb l'objectiu de millorar l'ambientació de l'escenari, s'han afegit dues llums amb colors blau i lila. La primera llum és tipus direccional i serà la principal (Figura 6.12.1). És la que més es veurà i té assignat el color més semblant a la nit.

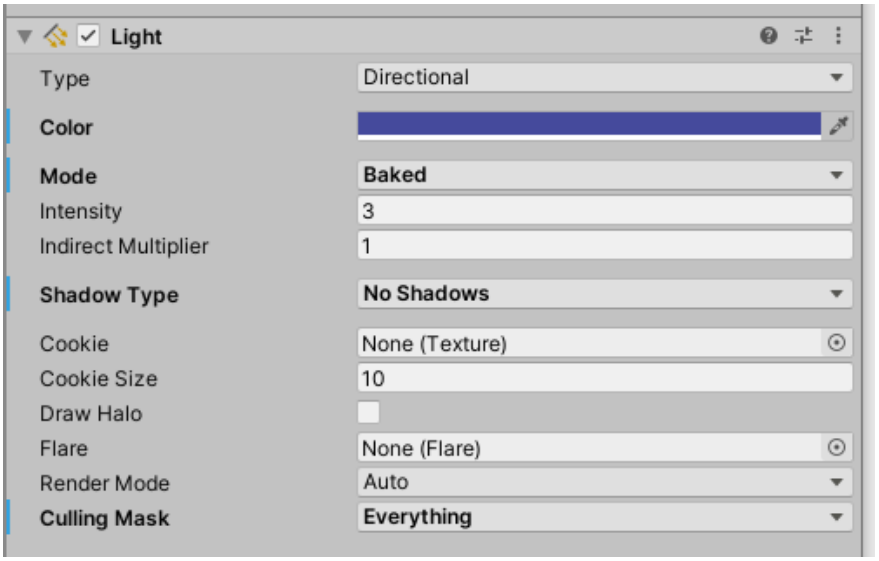


Figura 6.12.1 Component llum principal

La segona llum serà un blau més clar per generar un petit contraste a l'escena (Figura 6.12.2). Es casi imperceptible, però afegeix un detall interessant.

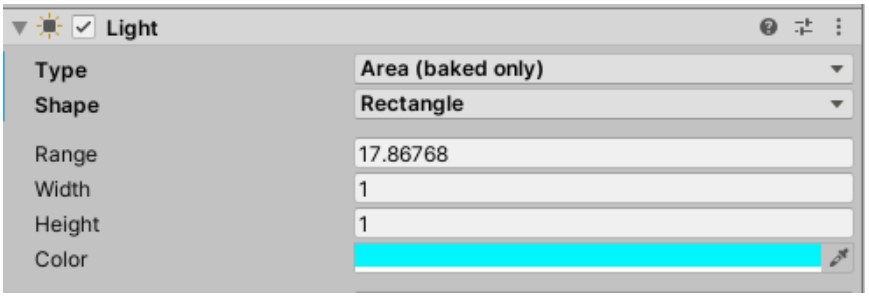


Figura 6.12.2 Component llum adjunta

Escenari sense l'efecte de les llums (Figura 6.12.3).

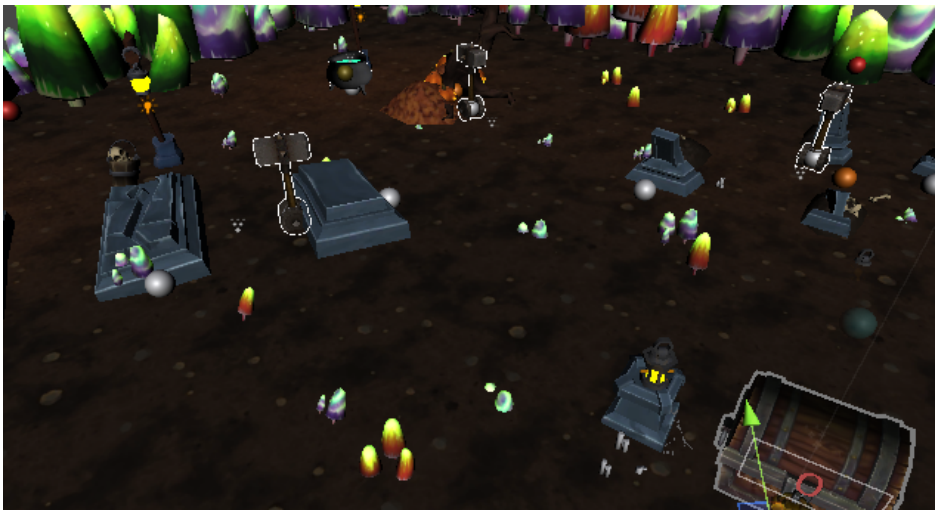


Figura 6.12.3 Escenari sense llums

Escenari amb l'efecte de les dues llums (Figura 6.12.4).

Figura 6.12.4 Escenari amb llums

6.12.2 Partícules

S'han afegit una sèrie de partícules en alguns elements per millorar l'aspecte visual i donar més informació al jugador. Totes s'han descarregat de 'UnityAssets'.

Quan un enemic és eliminat, apareix una petita explosió de color lila amb una calavera que va desapareixent (Figura 6.12.5).

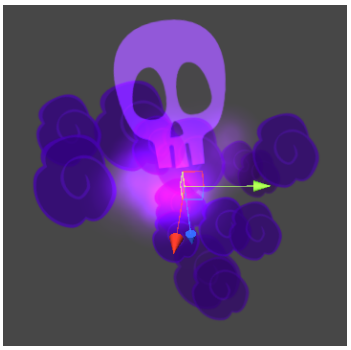


Figura 6.12.5 Partícula enemic eliminat

Quan s'agafa la bonificació que millora el temps de recàrrega de les trampes, apareix una petita il·luminació procedent del centre per indicar al jugador quan estan millorades i quan s'acaba (Figura 6.12.6).

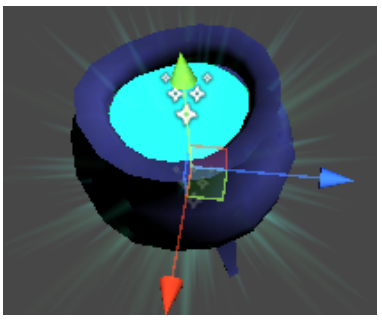


Figura 6.12.6 Partícula trampa millorada

Les trampes que afecten un àrea provoquen una explosió que indica l'àrea que provoca mal als personatges (Figura 6.12.7).

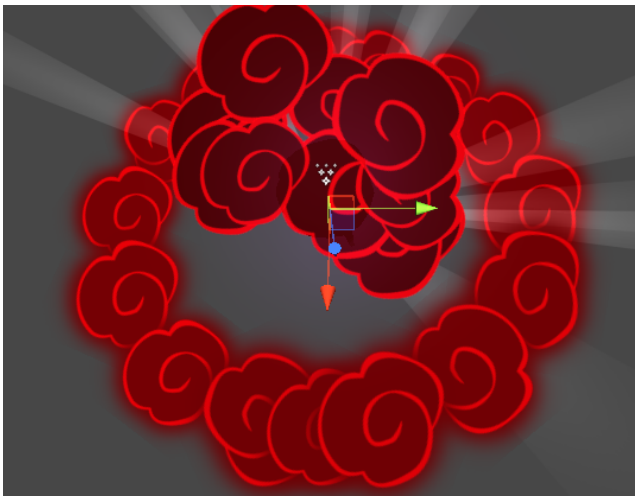


Figura 6.12.7 Partícula trampa activada

Quan el cofre és obert amb nivell 0, surten uns ratpenats en comptes de la recompensa per donar més ambientació (Figura 6.12.8).

Figura 6.12.8 Partícula cofre nivell 0 obert

Quan el golem està recarregant energia a la pedra màgica, apareix una llum al seu interior per indicar al jugador quan comença i acaba de carregar (Figura 6.12.9).

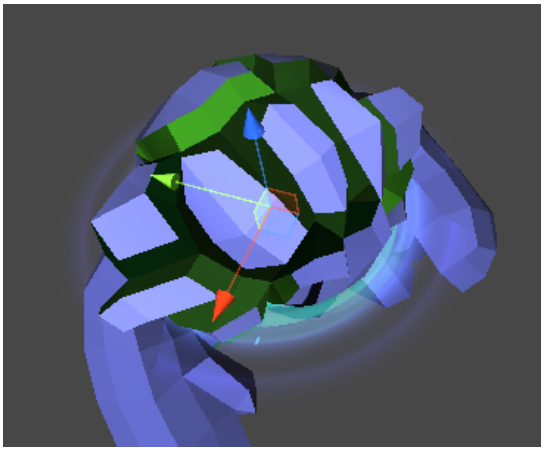


Figura 6.12.9 Partícula golem carregant

6.13 Proves realitzades

6.13.1 Errors

S'han trobat diferents errors mentre es desenvolupava el projecte. El joc s'ha anat creant per seccions, creant primer les trampes, després afegint els diferents enemics, acabant amb els personatges i bonificacions. Com els elements interactuen entre si, cada nou afegit inclou diferents problemes i dificultats. Per evitar-ho, s'ha dedicat un temps a fer proves de cada nou contingut que s'afegia, prohibint avançar fins a explorar totes les possibilitats. Amb aquest mètode s'han estalviat molts errors i ha ajudat a descartar a l'hora de intentar trobar l'origen.

Encara aix han sorgit erros de menys importància que s'han deixat per mes endavant. Cada certs canvis s'han fet proves de partides amb el dispositiu mòbil i s'han apuntat tot inconvenient que sorgia, per molt petit que sigues. A l'etapa final, s'han anat solucionant la llista de problemes i s'han fet les proves finals. Encara que hi ha errors que no ha donat temps a solucionar, són de menor importància i no afecten greument a la jugabilitat.

6.13.2 Balanceig

Per tal que el joc sigui entretingut i variat, s'ha hagut de dedicar un temps considerat en provar diferents combinacions de paràmetres. El més complicat de configurar ha sigut la classe 'SpawnManager' (Figura 6.13.1), encarregada de configurar les onades que apareixen al llarg de la partida. S'ha hagut de assignar valor a les variables per tal d'intentar fer del joc desafiant, però sense ser injust, evitant que sigui avorrit. S'ha configurat valors com el nombre d'enemics que apareixen, quan incrementa en cada onada, el temps entre aparicions o el número d'onades de cada grup.

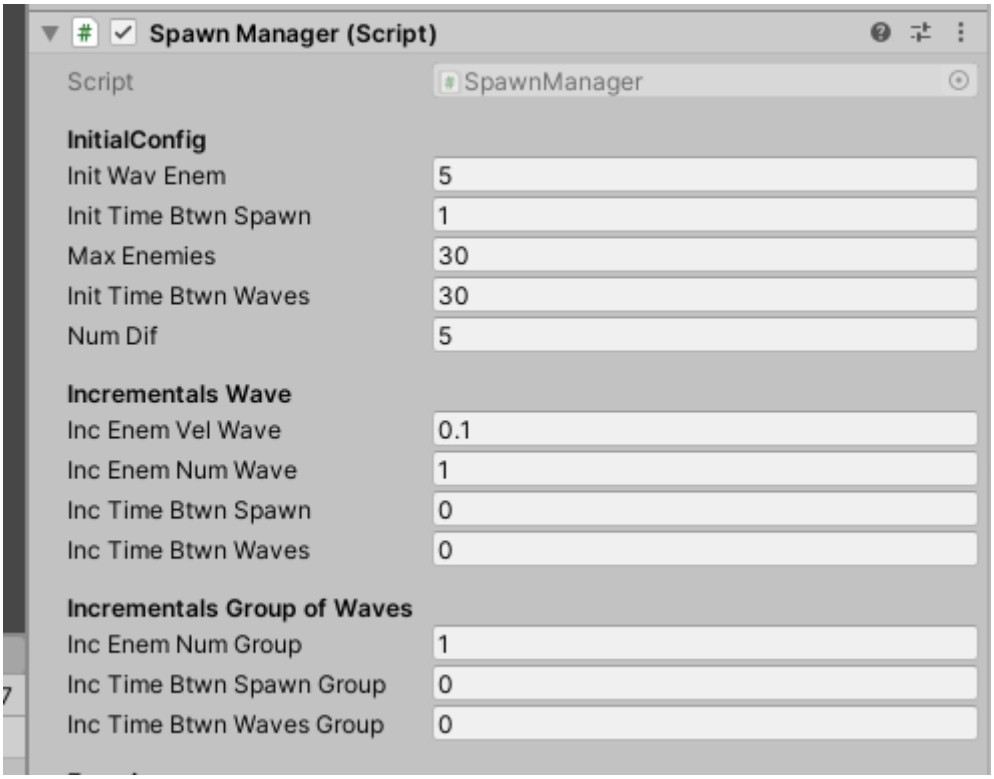


Figura 6.13.1 Configuració aparició enemics

Per cada trampa s'ha configurat el temps d'activació i quan s'ha d'esperar per tornar-la a utilitzar (Figura 6.13.2), així evitant crear una trampa millor que les altres.

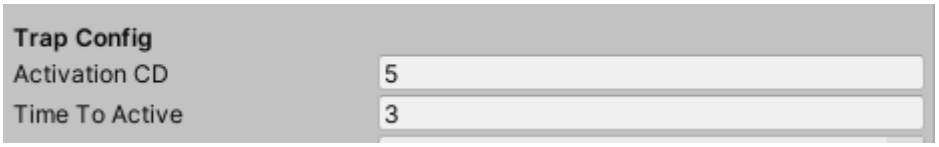


Figura 6.13.2 Configuració trampa

Per ultim, cada personatge conté una habilitat que afecta la jugabilitat de manera distinta. Perquè tots els personatges siguin igual de utils, s'ha configurat cada habilitat per ser justa respecte a les altres (Figura 6.13.3).

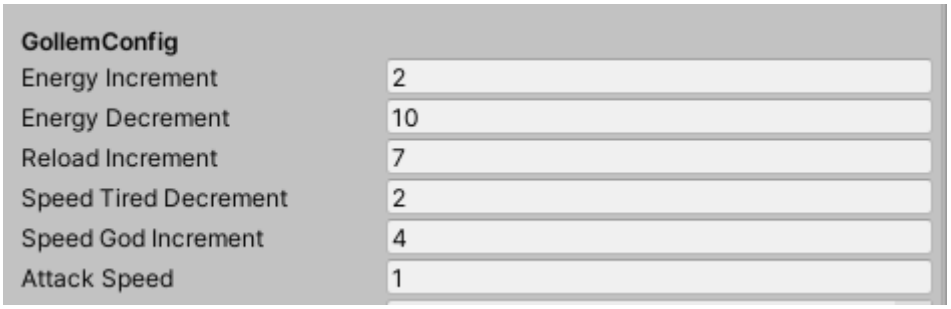


Figura 6.13.3 Configuració golem

6.13.3 Rendiment

Al acabar el prototip s'han trobat problemes de rendiment en alguns instant del joc. Per intentar millorar l'experiència de joc s'han canviat alguns paràmetres d'alguns components per reduir el cost de memòria.

S'ha canviat el tipus de les llums de 'Realtime' a 'Baked', disminuint la qualitat d'aquesta, però mantinguen l'essència de l'estil (Figura 6.13.4).

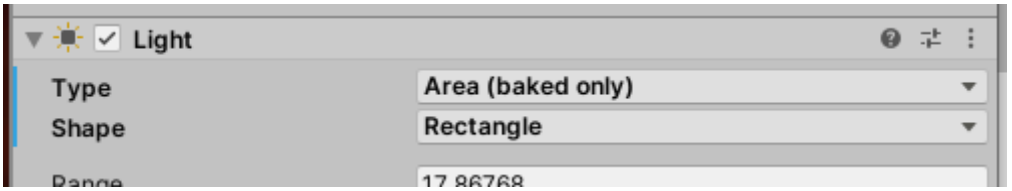


Figura 6.13.4 Tipus llum

Per reduir costos en les crides per dibuixar els objectes, s'han marcat com 'Static' tots aquells que formen part de l'escenari i no es mouran en cap moment (Figura 6.13.5).

Figura 6.13.5 Objectes a 'Static'

S'han canviat el 'Shader' dels materials a tipus 'Mobile' reduint la qualitat (Figura 6.13.6). Com es vol mantenir la qualitat dels elements d'escenari, s'han exclòs d'aquesta mesura.

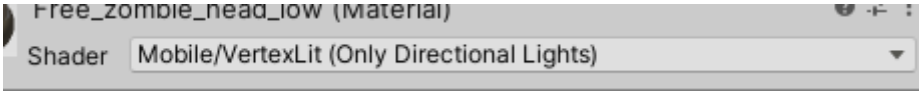


Figura 6.13.6 Materials 'Mobile'

Per aquestes últimes, s'ha activat l'opció 'Enable GPU Instancing' (Figura 6.13.7) de les textures per evitar crear un material per cada element repetit a l'escena. Es crea un material per cada objecte del mateix tipus, evitant ús de memòria.

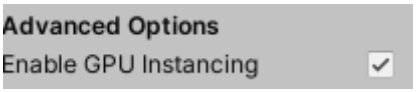


Figura 6.13.7 Material GPU Instancing

7 Resultats

7.1. Legislació i normativa vigent

El joc desenvolupat no presenta cap problema en aspectes legislatius. No 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 constitueix cap activitat econòmica.

Pel que fa als problemes de Copyright, a excepció dels models dels personatges, el joc ha estat desenvolupat completament per una persona o amb elements descarregats d'ús lliure. Per tant, una vegada canviat els models dels personatges, no hauria de suposar cap problema si es volgués comercialitzar. Tot i això, si la comercialització del joc fos un èxit i se superés un cert llindar d'ingressos definit pels propietaris de Unity, s'hauria de comprar llicències professionals del programa per a no tenir problemes legals.

7.2. 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 (Figura 7.2.1). En cap moment les classificacions són de compliment obligatori, les classificacions són merament informatives. És a dir, una persona menor d'edat mai li prohibiran comprar un joc que tingui l'edat recomanada de divuit anys.

Figura 7.2.1 Etiquetes PEGI

En el cas d'aquest projecte, en ser un joc centrat en l'eliminació d'enemics hauria de dur l'etiqueta de violència. Tot i ser la violència un tema principal, com aquesta es practica a enemics fantàstics i amb un estil gràfic gens realista, podem qualificar el joc com a PEGI 7, igual que Redungeon, un dels jocs en el qual s'ha basat per la idea.

7.3. Resultat final

En aquest apartat es mostraran captures de pantalla amb els resultats del projecte, per poder comparar amb els objectius establerts a l'inici.

Figura 7.3.1 Pantalla inicial

Figura 7.3.2 Selecció de personatge

Figura 7.3.3 Informació habilitats

Figura 7.3.4 Personatge bloquejat

Figura 7.3.5 Inici partida

Figura 7.3.6 Menú pausa

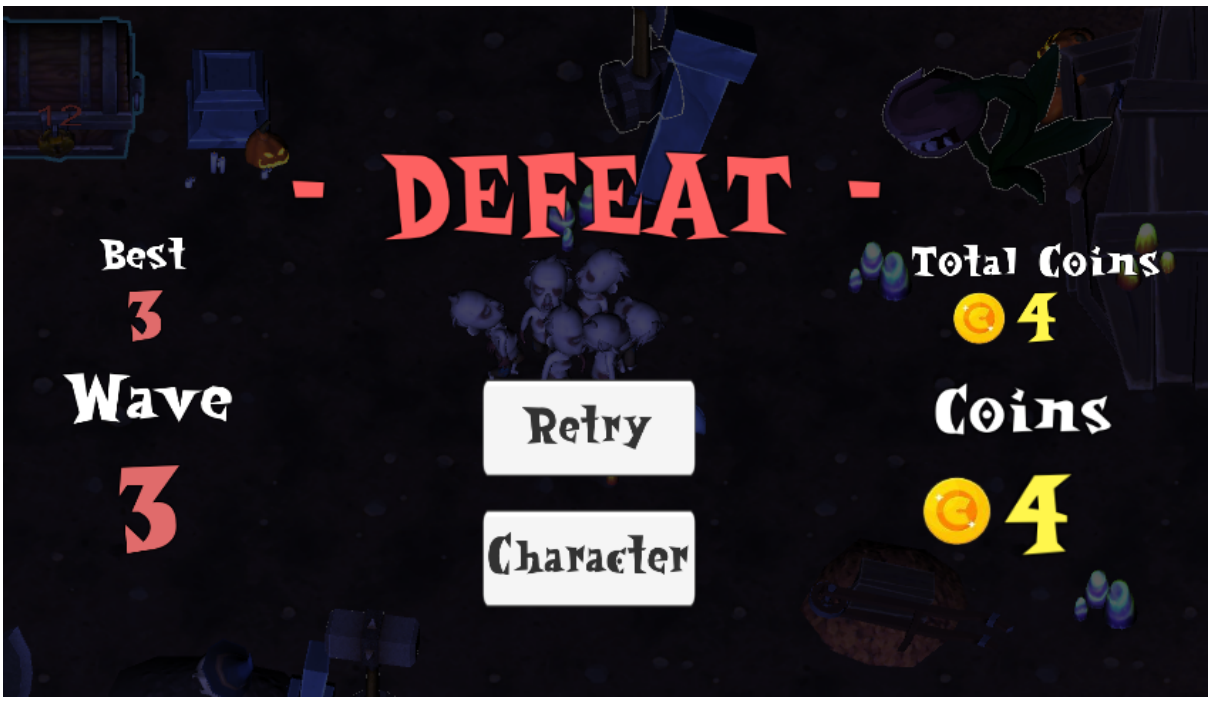


Figura 7.3.7 Pantalla final

Figura 7.3.8 Aparició enemics

Figura 7.3.9 Activació trampa

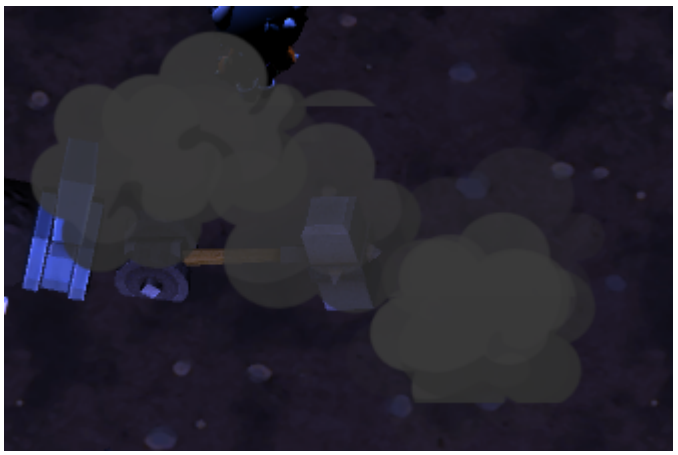


Figura 7.3.10 Atac trampa

Figura 7.3.11 Enemic mort

Figura 7.3.12 Partícules i recompensa

Figura 7.4.13 Combo enemic eliminats

Figura 7.4.14 Cofre nivell 2

Figura 7.4.15 Cofre obert i bonificació

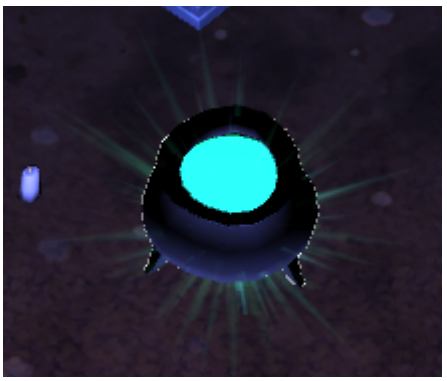


Figura 7.4.16 Bonificació trampa millorada

Figura 7.4.17 Bonificació escut

Figura 7.4.18 Enemic 2

Figura 7.4.19 Enemic 3

Figura 7.4.20 Enemic 3

Figura 7.4.21 Enemic 4

Figura 7.4.22 Guerrer escut

Figura 7.4.23 Guerrer sense escut

Figura 7.4.22 Habilitat Bufó

Figura 7.4.23 Claus Rei

Figura 7.4.24 Cofre rei

Figura 7.4.25 Energia Golem

Figura 7.4.26 Pedra Golem desactivada

Figura 7.4.27 Pedra Golem activada

Figura 7.4.27 Planta Enterrador

Figura 7.4.28 Ajudants Enterrador

8 Conclusions

En aquest treball s'ha pogut posar en pràctica tot l'aprenentatge rebut durant el Grau de Disseny i Desenvolupament de Videojocs. He tingut l'oportunitat de fer un projecte des de zero fins a completar-lo, passant per totes les fases per les quals ha de passar un joc seriós. Encara que s'hagin complert tots els objectius establerts a l'inici del projecte, he pogut notar que encara hi ha moltes coses per aprendre relacionat sobretot amb el disseny i la programació.

Aquesta falta de petits desconeixements, han sigut tant una dificultat com un repte per a mi. Desenvolupar un videojoc, comporta molta feina, sobretot obtenir la idea clara, ja que anava variant al llarg del temps, però enllestir-lo és molt plaent. Tot i que queden moltes coses que no han arribat al prototip per falta de temps o recursos, tinc el propòsit de què tinguin un lloc en el videojoc en un futur. De totes maneres, la finalització d'aquest projecte proporciona una recompensa molt satisfactòria.

Durant el transcurs de la implementació hi ha hagut numerables problemes en la programació, però que han set trobats en el moment de fer proves. A la vegada, cal comentar que no es tenia la idea 100% completada, fent que hi hagués alguns dubtes sobre possibles necessitats en un futur, o provocava més feina al moment d'adaptar el codi. Tot això s'ha pogut solucionar tenint en compte les possibilitats per prevenir canvis al moment d'afegir un element nou.

El disseny del joc conté la majoria d'objectius pensats al principi del projecte, mentre que el prototip té pràcticament totes les parts més importants i amb les quals es pot entendre la idea del joc. Totes les mecàniques principals que formen les bases han set completades amb èxit, igual que la part artística necessària per crear una ambientació adequada a la temàtica; aconseguint una finalització d'aquest treball interessant i enriquidora

Personalment, considero que és un videojoc el qual pot ser entretingut, amb un perfecte lloc en el mercat i amb res a envejar als jocs més famosos d'aquest gènere; tot després de dedicar-li unes petites millores i major temps en completar-lo. Els resultats han estat molt satisfactoris, he assolit els propòsits plantejats i a la vegada he après a treballar bé per obtenir uns bons resultats, el qual valoro molt positivament.

9 Treball futur

Encara sent un prototip, el projecte abasta un joc complet, d'inici a fi. Amb tot i això, queden alguns retocs i millores, per tal d'oferir l'experiència completa.

Un dels principals temes a treballar és afegir nou contingut dels elements principals. La idea del joc és oferir gran varietat d'opcions i situacions, actuant depenent de l'estat actual. Per aconseguir això, hi hauria d'haver un número més elevat de trampes, escenaris, bonificacions, enemics i personatges. A més, oferia més diversitat entre partida i partida, evitant la monotonia.

Enemics: Els enemics són dels elements més importants del joc, ja que són els que faran entretinguda les partides més llargues. Seria prioritari augmentar-los en gran quantitat. L'objectiu actual és afegir fins a 10 enemics.
Trampes i bonificacions: Ambdós són importants per oferir més opcions de sobreviure al jugador. S'hauria d'afegir més varietat, sent prioritari després dels enemics.
Personatges: Ja hi ha una bona varietat de personatges actualment en el prototip. Potser s'afegirà algun mes, però no és prioritari.
Escenaris: Ofereixen partides diverses, que eviten la repetició al llarg del joc. Ara mateix hi ha dos diferents. Seria un bon objectiu afegir entre 2 i 3 més.

Un altre tema important és l'art. Alguns dels recursos utilitzats (Personatges), són d'altres videojocs i no poden fer-se servir a la versió final. És necessari crear els propis, així com animar-los i texturitzar-los. S'hauria d'afegir música i sons per crear una bona ambientació. A més, es podria millorar l'estil artístic en general, afegint partícules o millorant les animacions actuals, així com la interfície.

Per últim, hi ha algunes idees de mecàniques extres que s'han acudit durant el transcurs del projecte, i podrien fer el joc més entretingut i divertit.

Les idees són:

Permetre gastar monedes en millores dels personatges, desbloquejant petites habilitats que poden combinar amb la principal.

Un sistema de 'Logros' amb mini reptes que pot completar el jugador mentre juga cada partida. No hi haurà una recompensa, simplement per incitar al jugador a completar-los tots.

Afegir onades especials, oferint un repte al jugador que haurà de superar d'imprevist, per aconseguir bonificacions o simplement sobreviure. Aquestes apareixeran de manera aleatòria, siguent diferent en cada partida. Per exemple, una onada on s'ha d'eliminar a tots els enemics en un període de temps, o un altre on s'ha de sobreviure X temps a una quantitat elevada d'enemics.

Implementar que els enemics puguin aparèixer amb millores, en onades més avançades. La idea és que apareguin els enemics que havien aparegut fins ara, però podrien tenir unes bonificacions

extres, com un escut o molta velocitat. Amb això s'aconseguiria sorprendre el jugador inclús quan ja ha vist tots els tipus d'enemics.

Afegir un mode de joc amb dificultat extrema, el qual es desbloqueja una vegada s'ha arribat a certa onada en el mode normal. Aquest mode començaria amb una dificultat equivalent a les onades avançades, així com incrementant l'aparició de millores als enemics, prèviament mencionades.

Aquests punts mencionats en aquest apartat són objectius que, una vegada assolits, permeten considerar videojoc com acabat.

10 Bibliografia

AuraProds. (2020, agost 24). *LA GUÍA DEFINITIVA DE BLENDER! (Tutorial completo en Español) | Desde cero! 2.91.* [Vídeo]. https://www.youtube.com/watch?v=h4hZzPCOMKs

Brackeys. (2012. desembre 23). [Canal]. https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA

Brackeys. (2018, desembre 2). *SAVE AND LOAD SYSTEM in Unity*. [Vídeo]. https://www.youtube.com/watch?v=XOjd_qU2Ido

Game Maker's Toolkit. (2006, juny 4). [Canal]. https://www.youtube.com/c/MarkBrownGMT/videos

GitHub, Inc. 2021. *GitHub*. https://github.com/

JLMUSSI (2021, setembre 23). *Substance Painter for Beginners Tutorial*. [Vídeo]. https://www.youtube.com/watch?v=s2MOx1Iteik

Stack Exchange, Inc. 2021. *Stack Overflow*. https://stackoverflow.com

SketchFab. 2021. https://sketchfab.com/

Unity technologies. 2021. *Unity Answers.* https://answers.unity.com/

Unity technologies. 2021. *Unity Asset Store*. https://assetstore.unity.com/

Unity technologies. 2021. *Unity Forum.* https://forum.unity.com/

Unity technologies. 2021. *Unity Manual.* https://docs.unity3d.com/

vBulletin Solution. 2021. *Foro3D* - https://www.foro3d.com/f111/

11 Manual d'usuari i d'instal·lació

11.1 Instal·lació

A continuació, es farà una breu explicació de com instal·lar el videojoc en un dispositiu mòbil Android.

1. Primer de tot, s'ha de baixar l'arxiu 'ZIP', contenidor de l'instal·lador (Figura 11.1). Es pot descarregar directament en el mòbil o a un ordinador, després transferir-ho al dispositiu.

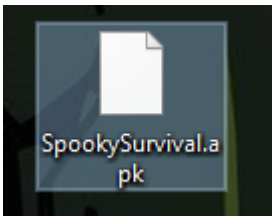


Figura 11.1 Instal·lador

2. Extraiem l'instal·lador (Figura 11.2) i una vegada passat al mòbil, l'executem.

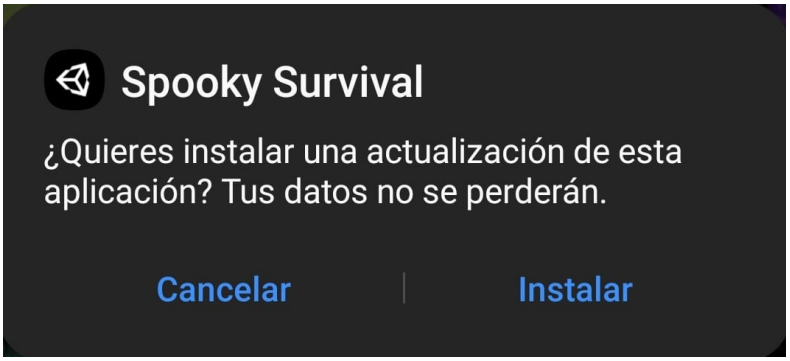


Figura 11.2 Missatge per instal·lar

Si teniu dificultats per transferir-ho al mòbil, conjuntament amb l'instal·lador, hi ha un fitxer de text amb un enllaç de descàrrega (Figura 11.3). Només cal enviar l'enllaç al mòbil (Whatsapp, Mail...) i descarregar-ho.

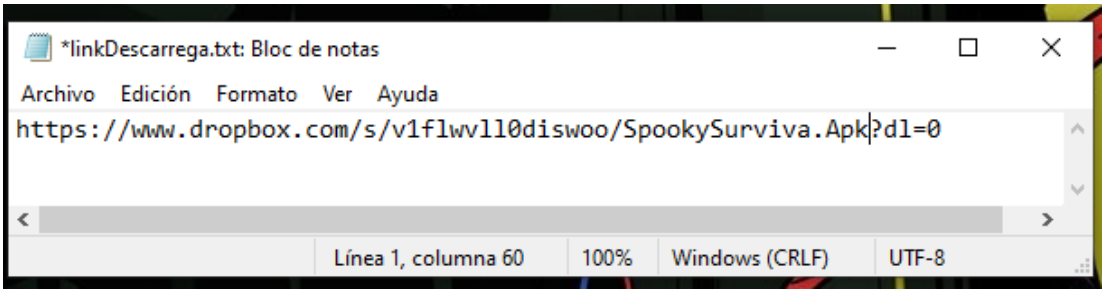


Figura 11.3 Enllaç descarrega

3. Pot sortir un missatge d'advertència, notificant que no està permès instal·lar aplicacions desconegudes. Per desactivar-ho, en el mateix missatge, s'oferirà una opció per desactivar aquesta limitació (Figura 11.4).

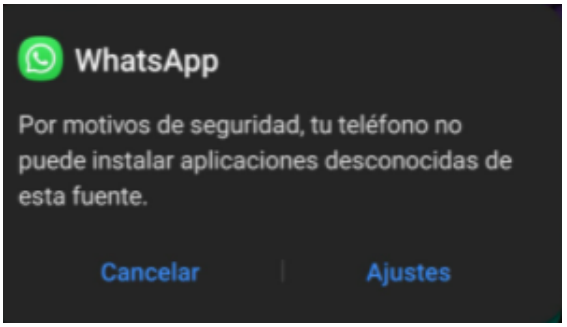


Figura 11.4 Avís permisos

Se'ns obrirà un menú i permetem executar APK (Figura 11.5)

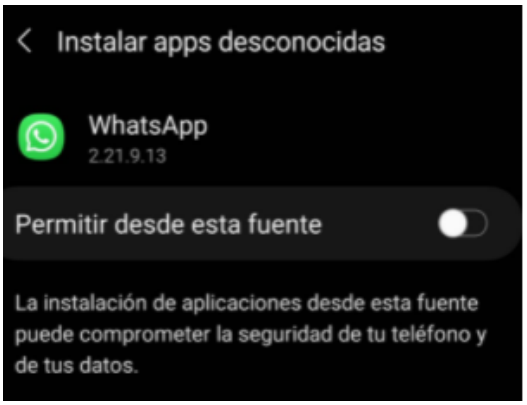


Figura 11.5 Desactivar limitació

4. Instal·lació finalitzada. Ara només caldrà trobar l'aplicació a la llista (Figura 11.6) i executar-ho.

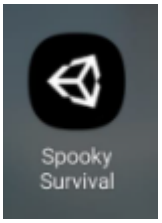


Figura 11.6 Aplicació instal·lada

11.2 Controls

Igual que molts videojocs per dispositius mòbils, a la part inferior esquerra de la pantalla, apareix un Joystick (Figura 11.7). Aquest, permet el moviment del personatge, simulant un Joystick d'un comandament físic. Arrossegant la bola central cap a la direcció desitjada, moure el personatge.

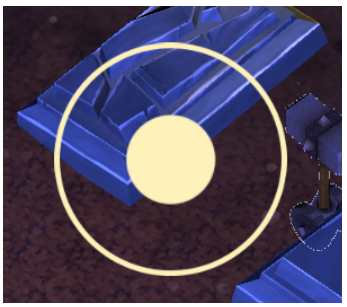


Figura 11.7 Joystick

A la part inferior dreta, apareix un botó, encarregat d'activar l'habilitat del personatge actual (Figura 11.8). Si s'utilitza, caldrà esperar un cert temps per tornar-la a utilitzar o donada certa circumstància, tot depèn del personatge. Quan no es pugui activar, apareixerà lleugerament fosc.

Figura 11.8 Botó habilitat

11.3 Començar una partida

Només obrir el joc, es mostrarà la pantalla inicial, mostrant el logotip del videojoc. Prement a qualsevol part de la pantalla, avancem al següent menú.

Es mostrarà el menú per triar personatge. Sempre apareixerà l'últim personatge amb el qual has jugat una partida, però sent el primer cop, apareixerà el primer personatge. A l'esquerra, apareix una breu descripció del personatge, així com les descripcions de les habilitats (Figura 11.9).

Figura 11.9 Selecció personatge.

Prement les fletxes laterals, es pot canviar de personatge. Quan un personatge està bloquejat, aquest apareixerà en negre. A més, apareixerà un botó per desbloquejar-lo, indicant les monedes necessàries. A la part superior apareixen les monedes actuals (Figura 11.10).

Figura 11.10 Personatge bloquejat.

Quan tinguem clar el personatge amb el qual jugarem, premem el botó de jugar.

11.4 Jugant una partida

11.4.1 Objectiu del joc

Només entrar a la partida, es genera el mapa i començarem en una posició predeterminada.

L'objectiu del joc és sobreviure el màxim d'onades possibles. A cada onada apareixeran un cert nombre d'enemics (Figura 11.11). Si un d'aquests toca el teu personatge, s'acaba la partida. En passar cert temps, o eliminar tots els enemics d'una onada, passarem a la següent.

Figura 11.11 Enemics

Al llarg del mapa hi ha una sèrie de trampes que s'activen quan el personatge està a prop. Quan s'activés i passat un petit temps de càrrega, aquestes eliminaran tot personatge que estigui dins de la seva àrea d'efecte. Això inclou tant el personatge com els enemics. Una vegada activada, necessitarà un cert temps per poder activar-se de nou.

Les trampes tenen un contorn de color indicant l'estat el qual es troben.

Blanc: La trampa està preparada per activar-se (Figura 11.12). Si detecta al personatge principal, es prepararà per atacar.

Figura 11.12 Trampa en espera

Vermell: La trampa es prepara per llançar un atac (Figura 11.13). Passat poc temps, atacarà.

Figura 11.13 Trampa atacant

Sense contorn: La trampa està recarregant-se(Figura 11.14). No es pot utilitzar.

Figura 11.14 Trampa recarregant

Cal centrar-se a sobreviure, esquivant les trampes i enemics, però alhora, cal utilitzar les trampes per eliminar els enemics, evitant l'acumulació d'aquests.

11.4.2 Mecàniques

El joc incorpora diverses mecàniques per facilitat la supervivència. Són totalment opcionals d'utilitzar.

Només començar la partida, el jugador apareix davant d'un cofre amb un comptador. Aquest cofre, dona una sèrie de bonificacions quan és obert, depenen del nivell en el qual es troba. Com més nivell, millor serà la recompensa obtinguda.

El nivell és representat pel color que l'envolta, sent el gris el color inicial i groc el més gran. Per pujar de nivell, cal eliminar enemics mentre el cofre estigui tancat (Figura 11.15). Quan arribi a cert número d'eliminats, el nivell augmentarà. El primer nivell no dona cap bonificació.

Figura 11.15 Cofre nivell 2

Els enemics deixen anar monedes quan moren. Depenen del color, la moneda té un valor diferent (Figura 11.16), passant de bronze a diamant, sent menys probable que aparegui. Es pot incrementar la probabilitat de millors premis eliminant més enemics alhora. És a dir, si en comptes d'eliminar enemics un a un, s'espera a acumular i eliminar-los a la vegada, es bonificarà amb monedes de més valors.

Figura 11.16 Monedes plata i or