Universitat de Girona
Escola Politècnica Superior

Grau en Enginyeria Informàtica

PROJECTE FINAL DE GRAU

Disseny i implementació d'un joc de tipus Hack & Slash en 2 dimensions

Autor:
Arnau Rísquez Guiteras

Tutors:
Gustavo Patow

MEMÒRIA

Convocatòria:
Setembre 2023

Departament:
Informàtica, Matemàtica Aplicada i Estadístic

Índex de contingut

Index de figures

1. Introducció

La indústria dels videojocs és un sector dinàmic i en constant creixement que engloba la creació, desenvolupament i distribució de videojocs electrònics. Des de les seves humils arrels als anys 70 fins a l'actualitat, s'ha convertit en una de les indústries de l'entreteniment més lucratives i influents del món.

Els videojocs han evolucionat des de senzills jocs d'arcada fins a experiències virtuals complexes, impulsats pel ràpid desenvolupament de la tecnologia i la connectivitat. Actualment, aquesta indústria atrau milions de jugadors de totes les edats i orígens, des de plataformes de videojocs tradicionals com a PC i consoles, fins a dispositius mòbils i realitat virtual.

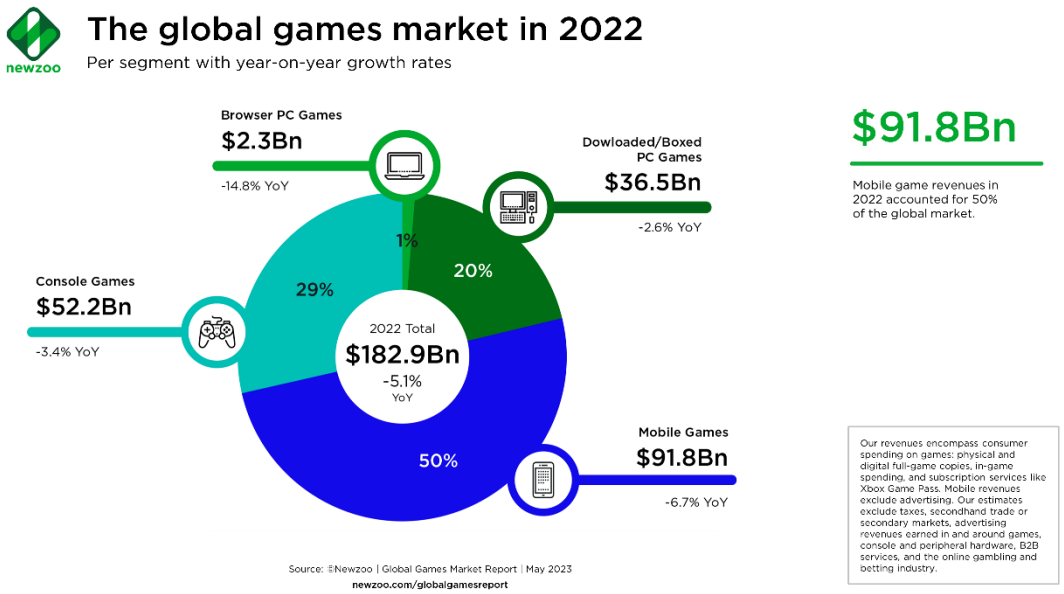


Figura 1: Gràfic dels guanys de cada plataforma l'any 2022

1.1 Motivació

La meva motivació darrera aquest projecte se centren a aplicar els meus coneixements apresos durant la carrera al sector del qual m'agradaria dedicar-me en un futur, alhora aprofitant per aprendre les diferents eines que s'utilitzen per a la creació de videojocs com ara és un motor de videojocs com Unity o habilitats no tan lligades amb la programació com és animació o disseny.

Un cop decidit que el meu projecte seria un videojoc, la meva decisió que fos un Hack & Slash en 2D ha estat pel fet que, tot i haver crescut en popularitat aquests últims anys, ha quedat en un segon pla a causa del creixement dels jocs de consola, normalment jocs triple A o doble A i sobretot a l'enorme creixement en popularitat dels jocs mòbils com es pot veure a la Figura 1.

1.2 Propòsit

El propòsit d’aquest projecte és el disseny i desenvolupament d’un videojoc hack & Slash en 2 dimensions amb pixelArt on el jugador es troba dintre un calabós ple d’enemics i haurà de trobar la sortida.

1.3 Objectius

Durant el transcurs d’aquest projecte l’objectiu és aprendre tot el següent:

- Aprendre i dominar el llenguatge C#
- Aprendre el funcionament del motor gràfic Unity3D
- Aprendre a crear animacions
- Aprendre a dissenyar un videojoc
- Crear un joc que sigui re-jugable sense tornar-se repetitiu.
- Crear masmorres que es generin aleatòriament
- Crear enemics amb diferents comportaments
- Crear diferents tipus d’atacs per als personatges jugables
- Crear un sistema de menús
- Crear diferents estils del personatges
- Crear un sistema de regulació de so
- Documentar el projecte

2. Estudi de viabilitat

El cost d'infraestructura necessitat per a desenvolupar aquest projecte ha estat pràcticament inexistent.En el cas del software el programari utilitzat ha estat el següent:

- Windows 11
- Motor de videojocs multiplataforma Unity.
- Visual Studio Code
- Editor d'imatges GIMP

En el cas del hardware s'ha utilitzat ha estat el següent.

	Ordinador sobretaula	Ordinador portàtil
Processador	11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz 3.60 GHz	
Ram	16GB	16GB
Gràfica	Nvidia Geforce RTX 3070	

A l'inici del projecte ja es comptava amb tot el material necessari i el programari de software, o ja es tenia prèviament, és de codi lliure o s'ha utilitzat la versió gratuïta, i per això no ha suposat cap cost addicional.

2.1 Viabilitat legal

En el cas dels drets legals i de propietat intel·lectual, tots els recursos externs que no són s’han obtingut de productes amb una llicència CC-0 com en el cas de gran part dels sprites o la música.

2.2 Inversió inicial

En tenir totes les necessitats de software i hardware, l'única inversió inicial requerida serà pagar la taxa de publicació que demanen tant Steam Greenlight com Epic Games Store, que en aquest cas, es tracta de 100 € tant per una part com per l'altra.

2.3 Recursos humans

Durant la creació d'un videojoc es poden diferenciar 5 perfils de treballadors que són completament necessaris per al desenvolupament d'aquest, aquests perfils de treball són els següents:

- Dissenyador: és el responsable de la creació de la jugabilitat i l'experiència del jugador.

- Programador: S'encarrega d'escriure i mantenir el codi de programació que fa funcionar el videojoc.

- Artista: es dediquen a crear els aspectes visuals del joc, com els personatges, els escenaris, els objectes i els efectes especials. El seu treball ajuda a donar vida i personalitat al món del joc.

- Guionista: és el responsable de crear la història i la narrativa del joc. Això pot incloure escriptura de guions, diàlegs i tots els elements narratius.

- Compositor: es dedica a crear els efectes de so, la música i les veus del jo

En el nostre cas considerarem la creació del joc des de la perspectiva d'una empresa indie, les quals solen tenir el mínim de treballadors possibles així que agafarem una persona de cada àmbit.

Perfil tècnic	Sou mensual
Dissenyador	1.720 €
Programador	1.740 €
Artista	1.101€
Compositor	1.570€
Guionista	1.583€
Total 6 mesos:	46.284€

Per saber el sou de cada perfil tècnic es comprovarà el sou mensual net mitjà de cada sector, les fonts de la informació mostrada a continuació, es troben a la bibliografia:

2.4 Cost tecnològics

En el cas dels recursos tecnològics, a l'utilitzar la major part del software de codi lliure, l'únic cost que hem de tenir en compte és el desgast estimat de la maquinària junt amb les llicències de Windows pertinents.

	Cost
Llicència Windows 11	725€(145€ x 5)
Desgast maquinària	250€(50€ x 5)
Total	975€

Ajuntant el cost tecnològic, els recursos humans i la taxa de publicació del joc quedaria un total de 47.359 € aproximats d'inversió inicial.

3. Metodologia

La metodologia utilitzada per al desenvolupament del nostre projecte és una metodologia personalitzada proposada pel tutor que consisteix en els següents passos:

1. Escollir el projecte a realitzar
2. Escollir les eines i llenguatges a utilitzar per dur a terme el treball escollit al punt anterior
3. Dividir el treball en diferents parts o mòduls per facilitar el desenvolupament del projecte
4. Escollir una de les parts dividides al punt 3
5. Comprovar si la part realitzada funciona correctament
    a. En cas de no funcionar es tornarà al punt 4 per revisar i arreglar els errors
    b. En cas de funcionar es comprovarà si era l’última part, en cas de ser-ho passarem al punt 6, d’altra banda tornarem al punt 4
6. Unim totes les parts del projecte
    a. Si la unió no funciona del tot correctament tornarem al punt 4
    b. En cas que tot vagi com s’espera avançarem al punt 7
7. Generar diferents models de proves per comprovar que el seu funcionament sigui el correcte
    a. Si el resultat mostra algun error de funcionament, tornem al punt 4
    b. Si els models de proves no mostren cap error, avancem al punt 8
8. Redactem el document del projecte

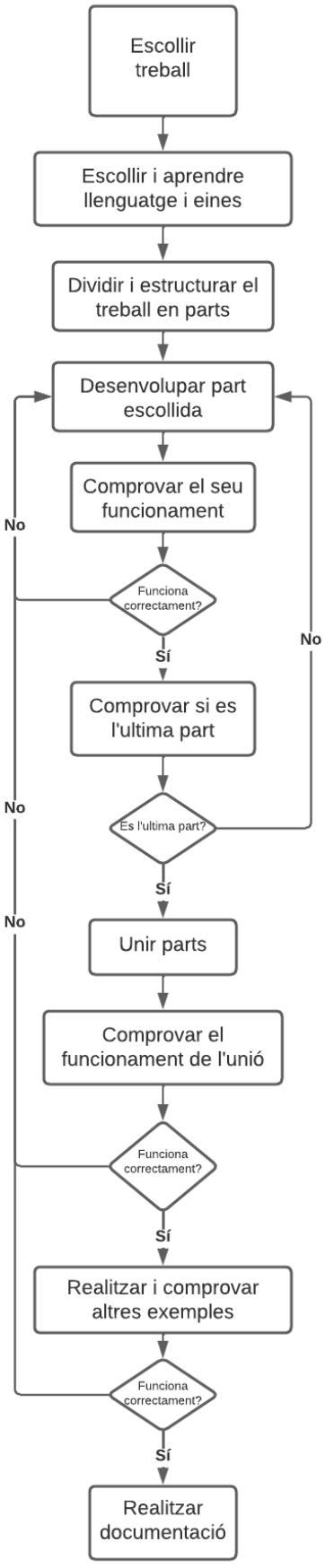


Figura 2 Metodologia utilitzada

4. Planificació

La planificació del projecte va començar al 13 de gener, quan es va produir la primera reunió amb el tutor per decidir com desenvolupar el projecte de final de carrera. El desenvolupament real no va començar fins al 20 de febrer, finalitzant així el 1r trimestre, durant aquest temps vam dur a terme les 3 primeres tasques.

4.1 Tasques planificades

4.1.1 Planificació del joc

En aquesta tasca vam decidir l’objectiu del joc, les característiques principals, els elements amb els quals podrà interactuar el jugador al llarg del joc i les regles principals.

4.1.2 Estudi dels motors gràfics i llenguatge

En aquesta fase vam aprendre el funcionament del motor de videojocs Unity junt amb el llenguatge de programació que utilitza, C#.

4.1.3 Cerca dels elements gràfics

En aquesta tasca buscarem tots els elements gràfics que necessitarem per al videojoc. Sprites dels herois, enemics, armer, projectils i decoracions, i en cas de no trobar-ne algun el crearíem.

4.1.4 Crear un prototip de sala

Es va crear una sala de les quals es formarà el mapa després amb les parets funcionals però sense poder obrir les portes per tenir un lloc de proves on poder fer les següents tasques sense necessitar fer tot el mapa procedural.

4.1.5 Disseny i implementació del moviment jugador

Aquí vam crear l’objecte jugador, quines estadístiques necessitaria. També es va decidir el tipus de moviment que tindria i les habilitats que faríem servir.

4.1.6 Disseny i implementació de la IA i el seu moviment

Aquí vam crear l'algorisme del comportament dels enemics, junt amb els diferents enemics que existiran i les diferències entre estadístiques entre tots ells. També es decidirà i implementarà el moviment dels jugadors i quin algorisme de moviment utilitzaran.

4.1.7 Disseny del sistema de combat

Aquí vam decidir i fer el sistema de combat. Si el sistema de combat seria cos a cos, a distància, si hi haurien habilitats i en cas d'existir habilitats, quantes i quines habilitats.

4.1.8 Disseny i implementació de l’algorisme d’un mapa procedural

Aquí vam dissenyar i fer l’algorisme que genera els mapes aleatoris de laberints, mida del laberint, tipus de sales, com apareixerien els enemics i com seria el funcionament de cada sala.

4.1.9 Disseny i implementació del sistema del control de càmera

Al tenir els mapes procedurals creats, vam decidir com funcionaria el moviment de la càmera i l’abast de visió que tindria.

4.1.10 Implementació d’aparició aleatòria d’enemics

En aquesta tasca vam implementar l’aparició aleatòria dels enemics en una sala, quants enemics apareixeran, el rang d’aparició que tindran i quins enemics poden aparèixer.

4.1.11 Implementació d’animacions

En aquesta tasca vam implementar animacions als personatges herois, enemics i entorn. A l'entorn vam fer les animacions de les portes en tancar i obrir i les animacions dels projectils enemics i herois. L'animació dels enemics són les animacions de córrer, i dels personatges herois, l'animació de córrer i animació d'atac.

4.1.12 Disseny i implementació d’enemics finals

En aquesta secció vam implementar l’enemic final que hi haurà en cadascun dels 2 mons, les mecàniques, animacions i comportament.

4.1.13 Disseny i implementació de menús

En aquesta fase vam decidir tots els menús necessaris que hi haurà durant tot el joc per després dissenyar e implementar tots els menús decidits al joc, tant els menús d'abans del menú principal com els menús durant la partida.

4.1.14 Implementació d’efectes de so i banda sonora

En aquesta etapa vam cercar tots els efectes necessaris del joc, igual que les bandes sonores i les vam implementar al joc.

4.1.15 Verificació i proves de funcionament

En aquest apartat vam realitzar diferents proves per assegurar que tot funcionés correctament.

4.1.16 Creació de la demo del videojoc

Vam crear la demo del joc amb tots els elements realitzats del joc.

4.1.17 Documentació

Aquesta tasca comença al principi del projecte i acaba al final del projecte i funciona amb paral·lel amb la resta de les tasques del projecte.

4.2 Temps estimat

El projecte està pensat perquè comencés el 13 de gener i acabés a finals d'agost, la planificació esperada es pot comprovar a la figura següent.

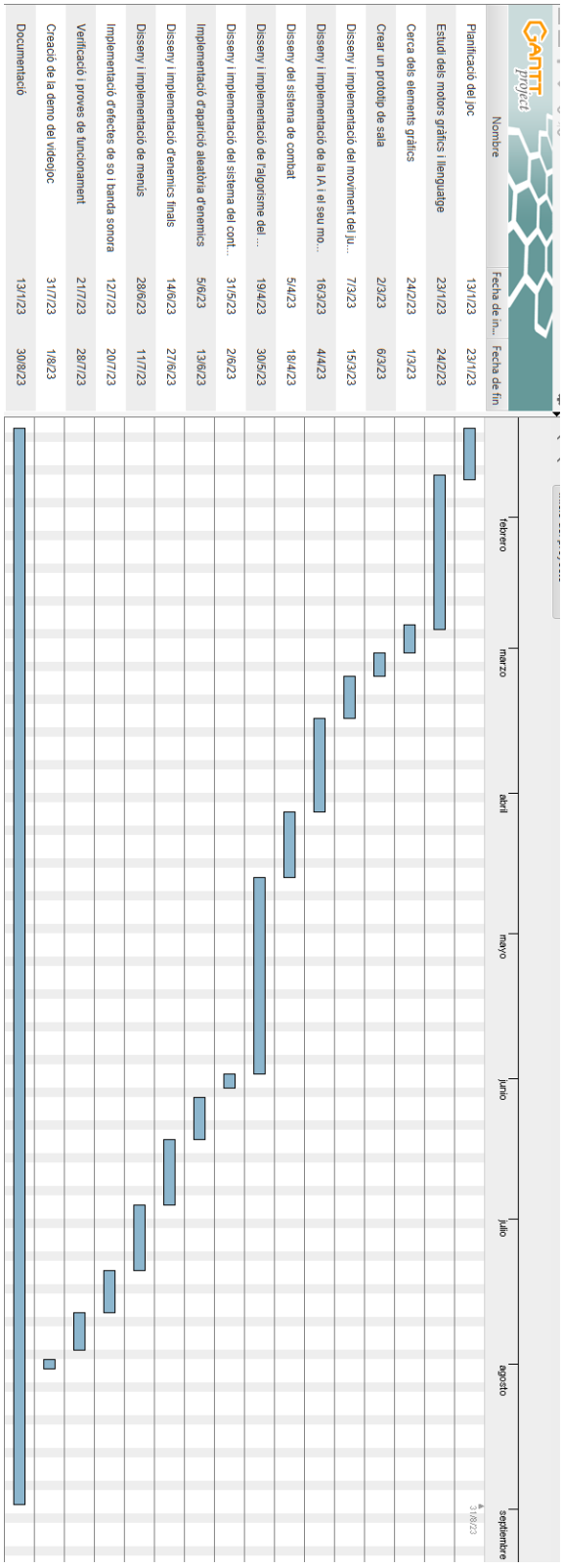


Figura 3 Diagrama de Gantt

4.3 Resultats esperats

L'objectiu d'aquesta planificació és crear un prototip d'un joc completament funcional i entretingut, però sobretot plantejat de tal manera que l'expansió d'aquest sigui molt fàcil.

5. Marc de treball i conceptes previs

La creació d'un videojoc des de 0 creant la infraestructura és molt complicat i porta moltíssim temps, En l'actualitat la gran majoria dels desenvolupadors utilitzen motors de videojocs per ajudar a agilitzar el desenvolupament.

Els motors de videojocs són conjunts de programari utilitzats per desenvolupar i crear videojocs de manera eficient. Aquests motors proporcionen eines i funcionalitats predefinides que permeten als desenvolupadors crear jocs amb més facilitat i rapidesa. Ofereixen diverses característiques com a gràfics 2D i 3D, física, gestió d'assets (com imatges, sons i models), intel·ligència artificial, sistemes d'animació, efectes especials que permeten als desenvolupadors centrar-se en la creativitat i la jugabilitat, ja que molts aspectes tècnics i de rendiment es tracten de manera automàtica pel motor.

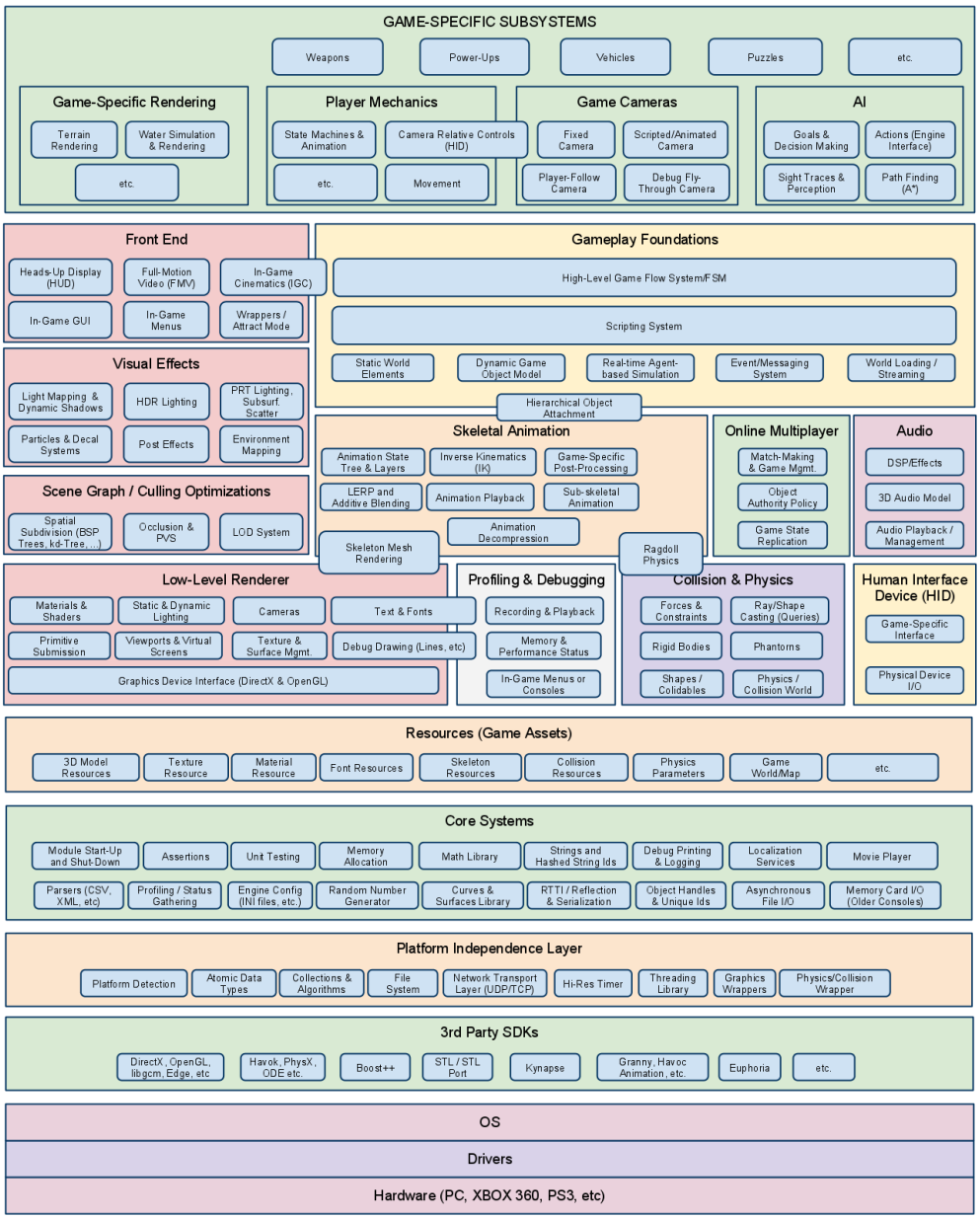


Figura 4 Esquema d'un motor de videojocs

5.1 Diferents motors de videojocs

5.1.1 Godot

Figura 5 Logo de Godot

Godot és un motor de videojocs 2D i 3D que destaca per la seva flexibilitat, eficiència i facilitat d’ús, és programari lliure i codi obert i és compatible amb una gran varietat de plataformes(Windows, macOS, Linux, iOS, Android i consoles).

Godot suporta els llenguatges de programació C#, VisualScript i GDScript amb tecnologia GDNative que permet al motor interactuar amb biblioteques compartides natives en temps d’execució. Compte amb una comunitat global molt activa de desenvolupadors i usuaris que comparteixen recursos, tutorials i solucions a problemes comuns.

5.1.2 CryEngine

Figura 6 Logo de Cryengine

CryEngine és un motor de videojocs desenvolupat per l’empresa Crytek. Inicialment, es va construir com un motor de demostració per Nvidia, però en veure el seu potencial, es va decidir implementar-lo per al videojoc Far Cry.

És conegut per la seva impressionant capacitat per renderitzar gràfics d'alta qualitat, amb il·luminació avançada, ombres realistes i efectes visuals impressionants.

5.1.3 UnrealEngine

Figura 7 Logo de Unreal Engine

Unreal Engine és un motor de videojocs desenvolupat per Epic Games que ofereix eines avançades per a la creació de videojocs i experiències interactives en 2D i 3D.

Aquest motor és un dels més importants del món dels videojocs que ofereix eines avançades per a la creació de projectes visuals, amb una gran flexibilitat per a desenvolupadors novells i experimentats. La seva potència gràfica, la capacitat de personalització i l’extensa comunitat fan que sigui una opció popular per a la indústria dels videojocs i altres aplicacions interactives.

5.1.4 Unity

Figura 8 Logo de Unity

Unity és un motor de videojocs àmpliament utilitzat que permet als desenvolupadors crear videojocs i altres aplicacions interactives en 2D i 3D. Unity compta amb una versió gratuïta, però és necessari canviar a una llicència per desenvolupadors quan s'arribi a un benefici superior a 10.000 €. Un dels grans atractius d'aquest motor és el gran suport per a diverses plataformes i una gran comunitat de desenvolupadors. Unity suporta el llenguatge C#. En aquest projecte la nostra elecció ha estat fer ús d'aquest motor.

6. Requisits del sistema

En aquest capítol estipularem els requisits de l’aplicació els quals agrupen en gran manera els objectius i les funcionalitats amb les quals ha de complir. Dividirem aquests requisits en dos grans apartats: els requisits funcionals, que fan referència a les funcionalitats que el sistema ha de dur a terme; i els no funcionals, que fan referència a les restriccions de disponibilitat dels recursos, seguretat, interfícies externes, la forma en la qual es desenvolupa el projecte, etc.

6.1 Requisits funcionals

- El jugador haurà de ser capaç de desplaçar-se utilitzant les tecles W,A,S,D del teclat, fer una envestida utilitzant l'espai i atacar amb el clic esquerre del ratolí.
- El jugador haurà de comptar amb una interfície gràfica que li digui la vida actual.
- Tant a l'iniciar com durant el joc el jugador serà capaç de modificar les opcions de so per reduir el volum de la música i efectes de so.
- El jugador podrà derrotar els enemics del mapa i rebre or i experiència d'ells.
- Els enemics podran atacar el jugador i, en cas de derrotar-lo, el jugador perdrà la partida
- Els diferents elements del joc podran col·lidir entre ells.
- El generador de masmorra sempre generarà laberints completament aleatoris que sigui possible de completar.
- Cada món de la masmorra comptarà amb un enemic final per aconseguir passar a la següent etapa.
- El jugador ha de poder tornar el menú principal en tot moment.

6.2 Requisits no funcionals

Els requisits no funcionals fan referència a la qualitat i característiques que hem de tenir en compte a l'hora de dissenyar el sistema, com ara els recursos necessaris per a un bon funcionament del joc, la seguretat necessària al sistema, etc.

En el nostre projecte, en ser un programa que s'executa únicament en mode usuari, només haurem de tenir en compte les restriccions de plataforma i els requisits tècnics per a un bon funcionament del joc.

- Les restriccions de plataforma seran les restriccions del motor de videojoc, en el nostre cas, Unity

- En el cas dels requisits tècnics l'ordinador amb les prestacions més limitades que s'ha utilitzat per testejar el joc són les següents:
    - Processador: Intel® Core™ i7-1065G7
    - Memòria RAM: SDRAM DDR4-2666 de 16 GB
    - Sistema Operatiu: Windows 10 Home 64
    - Targeta Gràfica: NVIDIA® GeForce® MX250
    - Memòria SSD: 256 GB

7. Estudi i presa de decisions

7.1 Sistema Operatiu

El Sistema Operatiu utilitzat per aquest projecte ha estat Windows 11 Pro.

7.2 C#

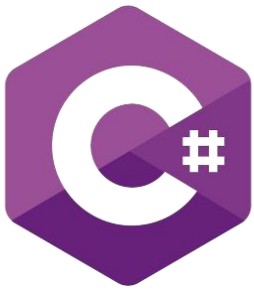


Figura 9 Logo de C#

És el llenguatge que suporta el motor de videojocs Unity que utilitzarem per programar tots els scripts que formen part del videojoc. Aquest és un llenguatge de programació orientat a objectes desenvolupat per Microsoft basant-se en els llenguatges C i C++ amb similituds amb el llenguatge Java.

Hem decidit escollir aquest llenguatge al ser un dels llenguatges més utilitzat dins l’industria dels videojocs i al ser el llenguatge de programació estàndard per al desenvolupament de videojocs amb Unity.

7.3 Visual Studio Code

Figura 10 Logo de Visual Studio Code

Visual Studio Code és un editor de text lleuger però potent desenvolupat per Microsoft de codi obert, que s'executa a l'escriptori i està disponible per a Windows, macOS i Linux. Visual Studio Code és altament extensible i permet instal·lar extensions que afegeixen funcionalitat addicional per diferents llenguatges de programació, eines i tecnologies. Hem elegit Visual Studio Code, ja que és una eina molt versàtil, potent i de fàcil ús que amb l'àmplia gamma d'extensions permet adaptar-se a les necessitats del projecte.

7.4 GIMP

Figura 11 Logo de GIMP

GIMP és un programa d'edició d'imatges de codi obert i gratuït, és desenvolupat pel projecte GNU i està disponible per a múltiples plataformes, incloent-hi Windows, macOS i diverses distribucions de Linux. Hem decidit utilitzar el GIMP per crear els sprites que ens podrien faltar per desenvolupar el videojoc ja sigui editant altres sprites o creant-ne de nous.

7.5 Microsoft Word

Figura 12 Logo de Microsoft Word

Microsoft Word és un software d'edició de textos comercialitzar per Microsoft i ens vam decantar per utilitzar aquest editor de text en tenir prèviament una llicència i haver utilitzat durant molts anys aquest programa.

7.6 Gantt Project

Figura 13 Logo de Gantt project

Gantt Software és una aplicació gratuïta per realitzar diagrames de Gantt, i amb la que hem realitzat els diagrames de la Figura 3. Un diagrama de Gantt, és una gràfica que conté la cronologia de totes les tasques previstes en un projecte, juntament amb la seva duració i l'ordre en el qual es realitzaran.

7.7 Unity

7.7.1 Entorn visual de Unity

L'entorn de Unity permet de forma intuïtiva ordenar, escalar i obrir/tancar totes les finestres per fer la interfície d'usuari el més còmode possible. A continuació explicarem les finestres més importants.

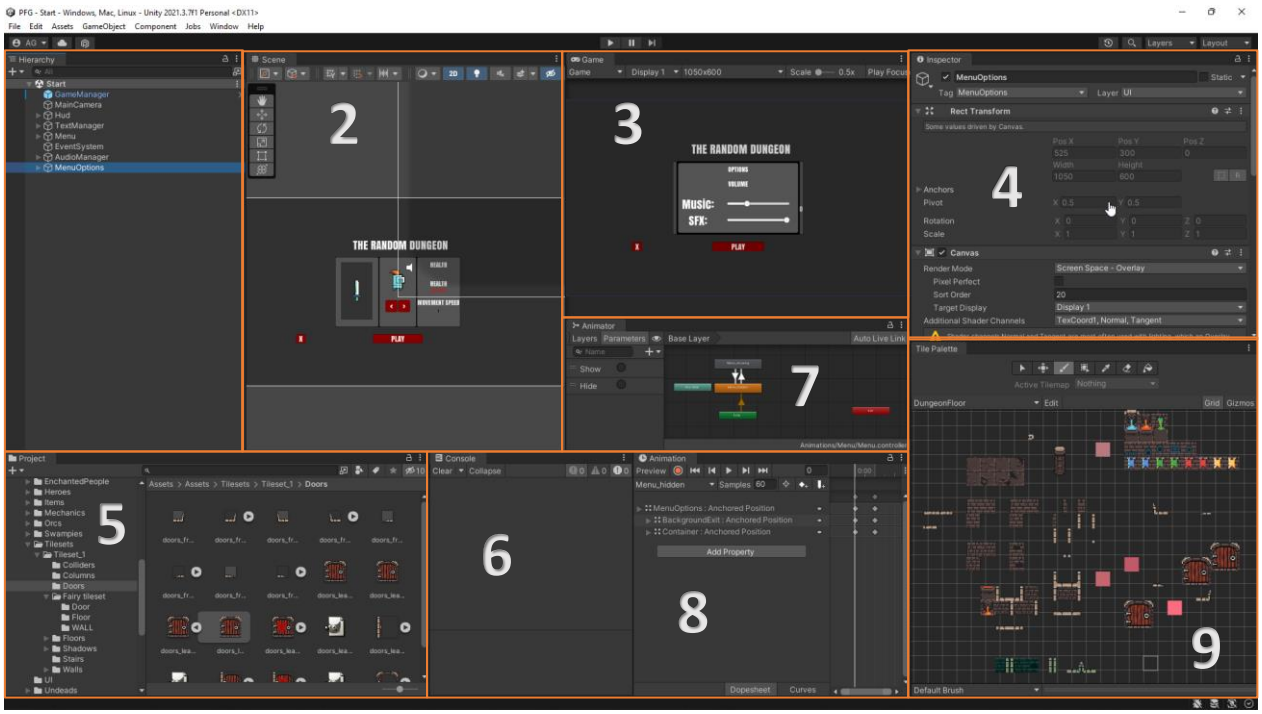


Figura 14 Entorn vidual de Unity

1. Jerarquia d’escena

Aquesta finestra mostra tots els GameObject de l’escena actual, permet crear, moure , modificar i esborrar tots els GameObject.

2. Visió d’escena

Aquesta finestra permet veure l’escenari d’aquesta escena amb els GameObjects de la Jerarquia d’escena i modificar la vista que es veuria a la visió de joc.

3. Visió de joc

La visió d’aquesta finestra és la vista prèvia de com el joc es veuria a cada moment.

4. Inspector

La finestra d’inspector permet visualitzar tots els components del GameObject seleccionat i afegir, eliminar o modificar els mateixos.

5. Explorador de fitxers

Aquesta finestra permet accedir a tots els elements del projecte, es troben ordenats a través d’un arbre de directoris.

6. Consola

En aquesta finestra podem visualitzar tots els missatges de consola, error i avis que doni el motor durant l’execució i els errors que puguin tenir els scripts.

7. Visió de control d’animació

Aquesta finestra permet organitzar com s’executaran les animacions del GameObject que tingui col·locat aquell Animator Controller.

8. Visio d’animació

En aquesta finestra podrem veure i modificar les animacions del GameObject seleccionat.

9. Paleta de textures

Aquesta finestra contindrà tots els elements de textures que ens permetrà decorar el mapa de forma senzilla.

7.7.2 Conceptes importants

Per tal d’entendre correctament els capítols següents de la documentació i el funcionament de Unity, és important conèixer els conceptes generals que s’esmenten a continuació:

7.7.2.1 Textura

La textura és una imatge aplicada a un model o material per donar-li un canvi visual.

7.7.2.2 Collider

Component de Unity que ajudarà a saber quan dos objectes se superposaran o contactaran, un Collider pot només avisar d'una superposició, però també evitar-la. Necessita tenir un Rigidbody associat.

7.7.2.3 Rigidbody

Rigidbody permet que un GameObject sigui afectat per la física del joc. Permet donar al GameObject una massa i fricció modificable i restringir quins eixos del món es podrà desplaçar i rotar l’objecte.

7.7.2.4 GameObject

Els GameObjects es tracten de tots els elements que hi ha a una escena i que podem desplaçar, escalar i eliminar. Cada GameObject està format per un conjunt de Components que li donen diferents característiques especials. Un GameObject amb Components específics pot ser guardat per a un ús posterior. Aquests elements guardats s’anomenen Prefabs.

7.7.2.5 Component

Els components agrupen tots els atributs que un GameObject pot arribar a tenir. L'únic Component que és obligatori tenir per a tots els GameObjects és el Component Transform, el qual dona la posició, rotació i escala del GameObject.

7.7.2.6 Camera

Dispositiu que actuarà com una càmera de vídeo i donarà la visió del joc al jugador.

7.7.2.7Canvas

Component de Unity que permet la creació d’interfícies d'usuari i de menús en el videojoc.

7.7.2.8 Event System

GameObject que permet la detecció d’events, és necessari per exemple per saber quan un element ha estat clicat.

7.7.2.9 Audio Source

Aquest component permetrà reproduir un clip d’àudio des d’un punt del joc. Es pot reproduir el clip d’àudio de manera que, quan la càmera s’allunyi de la font d’àudio el volum decreixi o que mantingui el mateix volum sense importar la distància.

7.7.2.10 Transform

Component d'un GameObject que informa sobre la posició, rotació i escala.

7.7.2.11 Tag

El tag ens permet diferenciar els nostres GameObjects dintre del videojoc.

7.7.2.12 Layer

El layer, com també en el tag, permet diferenciar els GameObjects dins el videojoc, però en aquest cas divideix els GameObjects en diferents capes per poder evitar renderitzar tots els GameObjects de l'escena o evitar que alguns GameObjects col·lideixin entre si.

7.7.2.13 Animator Controller

Un Animator Controller ens permet posar quines animacions tindrà l'objecte al qual estigui adherit i quins requisits hi haurà per passar d'una animació a un altre.

7.7.2.14 Animator Component

Aquest component permetrà modificar un GameObject dins un cert període de temps. Aquest component estarà controlat per una Animator Controller.

7.7.2.15 SceneManager

Aquesta classe permet gestionar totes les escenes del videojoc carregant-ne, destruint-ne o creant-ne de forma síncrona o asíncrona.

7.7.2.16 Monobehaviour

És la classe precedent de totes les classes que implementen scripts de Unity. Aquesta classe conté mètodes que controlen el flux d’execució del programa i la resposta davant d’interaccions externes per part dels Colliders.

7.7.2.17 Script

Fitxer de programació que podem afegir a un GameObject que permet donar un comportament específic a l’objecte.

7.7.2.18 Astarpath

Astarpath és un sistema de cerca de camins que utilitzant dijkstra et dona el camí més curt per arribar d’un punt a un altre.

8. Anàlisi i disseny del sistema

En aquest apartat explicarem com serà el joc que desenvoluparem, la temàtica i els elements del que contindrà.

8.1 Descripció General

The Random Dungeon és un joc que creat pensant principalment en l'estil de joc "Hack&Slash". Aquest gènere es caracteritza per posar la major part del pes mecànic al combat acabant amb grans nombres d'enemics, generalment utilitzant armes cos a cos i utilitzant un combat en tercera persona. És un gènere molt combinat amb els gèneres metroidvania i roguelike.

En el nostre cas el projecte es basa en un videojoc en 2D amb una visió top-down que emprarà un sistema de combat al més pur estil "hack and slash" amb la utilització de generació de mapa d'un "Roguelike". El jugador haurà de passar per diferents sales matant a cada enemic fins a arribar a matar l'enemic final que es troba a cada món fins a trobar la sortida. Alguns jocs semblants es podria considerar "Hades" i "The Binding of Isaac".

Figura 15 The Binding of Isaac

Figura 16 Hades

8.2 Disseny del funcionament

8.2.1 Personatges principals

El jugador podrà triar, abans de començar la partida, un personatge d'entre 4 diferents amb el que recórrer la masmorra per intentar superar-la. Cada personatge compta amb estadístiques i armes diferents, les quals podrà millorar dues vegades i en pujar de nivell millorarà la mida màxima a part de recuperar tota la vida perduda.

8.2.1.1 Personatges cos a cos

Els personatges elegibles cos a cos són personatges que comptaran amb més vida que els personatges a distància i amb els atacs podran parar els projectils enemics per així ajudar que es puguin acostar als enemics amb atacs a distància.

8.2.1.1.1 Knight (Cavaller)

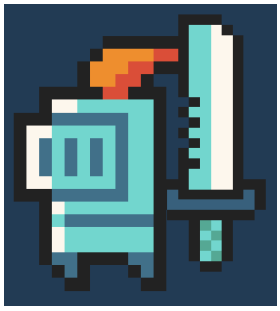


Figura 17 Knight (Cavaller)

Aquesta classe és la que compta amb més vida de tots els personatges, però també és la més lenta. El personatge va recobert amb una armadura i porta una espasa enorme amb la qual infligeix grans quantitats de mal amb un temps d'espera moderat entre cada atac.

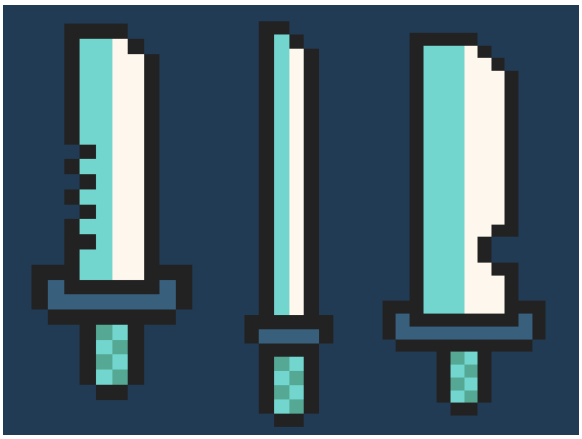


Figura 18 Armes del Cavaller

8.2.1.1.2 Swordsman (espadatxí)

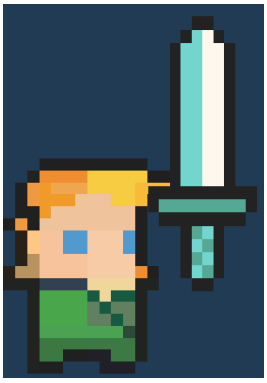


Figura 19 Swordsman(espadatxí)

Aquesta classe és una classe amb atacs cos a cos més ràpida i amb menys vida que el cavaller. Compta amb una espasa que es millora fins a una claymore durada. En aquest cas el personatge no porta armadura com l'anterior i els atacs, tot i que no fan el mateix mal que l'anterior, disposen de més velocitat d'atac.

Figura 20 Armes de l’Espadatxí

8.2.1.2 Personatges a distància

Els personatges amb atacs a distància comptaran amb menys vida, però més velocitat per incentivar a intentar mantenir la distància més gran possible amb els objectius. En aquest cas, per evitar que les classes a distància siguin molt millors que les cos a cos, els projectils enemics en entrar en contacte amb els projectils aliats els faran desaparèixer i els projectils enemics seguiran el seu camí.

8.2.1.3 Archer (Arquer)

La classe arquera és la classe amb menys vida de tot el joc i és la més ràpida també. Aquesta és una de les dues classes amb atacs a distància i en millorar no millora l'arc sinó, en canvi, les seves fletxes. Compta amb una velocitat d'atac molt alta però un dany per fletxa molt baix.

Figura 21 Archer(Arquer)

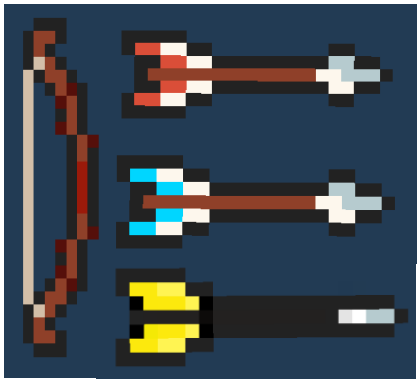


Figura 22 Armes de l'Arquer

8.2.1.4 Mage (Mag)

La classe Mag és una classe a distància que dispara projectils de foc als enemics. Els projectils són significativament més lents que les fletxes de l'arquer, però proporcionen una gran quantitat de mal i compten amb una hitbox molt més alta fent-los molt més probables de ser bloquejats per projectils enemics. En millorar l'arma milloren tant l'arma del personatge com els projectils, i el projectil millorat al màxim és l'atac amb més mal de totes les classes.

Figura 23 Mage (Mag)

Figura 24 Armes del Mag

8.2.2 Enemics

En aquest apartat explicarem la varietat d’enemics comuns que formen part del videojoc. Tots els enemics tenen un comportament específic donat per un algorisme que simula el comportament de les dades que estan especificades a cada GameObject enemic.

8.2.2.1 Slime

Aquest enemic és l'enemic més fàcil del joc i el comportament consisteix en seguir al jugador fins a xocar a una velocitat no molt alta. Junt amb el Slime de fang són els únics enemics que apareixen als dos mons i en morir dona al jugador 1 d'experiència i 1 d'or.

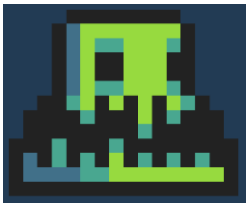
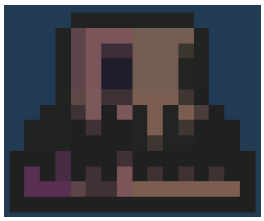


Figura 25 Slime

8.2.2.2 Slime de fang



Aquest enemic segueix el jugador fins a estar a una distància curta i cada 5 segons dispara un projectil cap a les 4 direccions. En morir dona 2 d'experiència i 2 d'or.

Figura 26 Slime de fang

8.2.2.3 Petit Diable

El Petit Diable és l'enemic amb menys vida de tot el joc. Aquest enemic morirà amb un sol cop de qualsevol personatge excepte l'arquer, a l'entrar a la sala perseguirà al jugador a gran velocitat per intentar causar mal. El Petit Diable en morir dona 1 d'experiència i 1 d'or.

Figura 27 Petit Diable

8.2.2.4 Diable

El Diable és una versió més gran amb més vida que també es pot trobar al primer món igual que el Petit Diable, però també té menys velocitat de l'anteriorment esmentada. En morir dona al jugador 1 d'experiència i 2 d'or.

Figura 28 Diable

8.2.2.5 Dimoni

El Dimoni és la versió d’enemic més fort del primer món que es dediqui a perseguir el personatge per causar mal. Compta amb el doble de vida del Diable i el triple de mal. En matar un Dimoni deixarà al jugador 2 d’experiència i 1 d’or.

Figura 29 Dimoni

8.2.2.6 Bruixot

El Bruixot és un enemic del primer món que intentarà sempre mantenir una certa distància amb el jugador i disparà cada 3 segons un projectil dirigit al jugador. En morir deixa al jugador 1 d’experiència i 3 d’or.

Figura 30 Bruixot

8.2.2.7 Centaure

El Centaure és un enemic del segon món que seguirà a l’enemic amb gran velocitat i disposarà de una gran quantitat de vida. Realitzarà més dany que qualsevol dels enemics que es troben al primer món i és l’enemic amb atacs cos a cos amb més mal del joc. En matar-lo el jugador aconsegueix 1 d’experiència i 1 d’or.

Figura 31 Centaure

8.2.2.8 Mag

El Mag és un enemic que tira boles de foc cap al jugador, del qual intentarà sempre mantenir distància del jugador. És l’enemic comú amb més mal del joc, però dels enemics que apareixen només al segon món és el que té menys vida entre ells. En morir el jugador aconsegueix 1 d’experiència i 1 d’or.

Figura 32 Mag

8.2.2.9 Ent

L’Ent és l’enemic comú amb més vida del joc que dispara discos a gran velocitat cap al jugador cada 2 segons. L’Ent és l’únic enemic sense moviment i en morir deixarà al jugador 3 d’experiència i 1 d’or.

Figura 33 Ent

8.2.3 Enemics únics

Els enemics únics són molt més forts que els enemics comuns i compten amb mecàniques pròpies. Cada món disposa d'una sala que contindrà un enemic únic que en morir desbloquejarà les escales al següent món.

8.2.3.1 Gran Dimoni

Figura 34 Gran Dimoni

El Gran Dimoni és l'enemic únic del primer món i és un enemic gran amb moviment molt lent i molta vida que intentarà fugir del jugador mentre invoca entre 1 i 3 Dimonis cada 4 segons que no pararan d'aparèixer fins que mori. El gran dimoni no compta amb un gran mal, però en tenir una gran vida, si el jugador no té prou mal per matar ràpidament als Dimonis o el mateix Gran Dimoni, els Dimonis es van acumulant fins a derrotar el jugador. Els Dimonis invocats pel Gran Dimoni no donen ni experiència ni or, però en morir, el Gran Dimoni deixa anar 5 d'experiència i 10 d'or. En matar tant al Gran Dimoni com a tots els Dimonis invocats apareixeran les escales.

8.2.3.2 Reialesa feèrica

Figura 36 Rei feeric

La Reialesa feèrica és un enemic únic dual que compta amb el Rei feeric i la Reina feèrica, on cada un té un set d'atacs diferents i estadístiques diferents. El Rei perseguirà el jugador i quan detecti que està a una distància prou petita farà un atac amb l'espasa al jugador, i la Reina mantindrà distància amb el jugador, la qual disposa de 4 estils d'atacs a distància on n'elegirà un aleatòriament i disparará una ràfega de projectils que duraran 5 segons. En acabar esperarà 5 segons abans de disparar la ràfega següent. El Rei és l'enemic amb més vida del joc i en morir donarà al jugador 30 d'experiència i 1 d'or i la Reina tot i que compta amb molta menys vida que el Rei, és el segon enemic amb més vida del joc i en morir deixarà al jugador 1 d'experiència i 30 d'or. En morir els dos es considera l'enemic únic derrotat i apareixeran les escales.

Figura 35 Reina feèrica

8.2.4 Elements de l’entorn

8.2.4.1 Portes

Cada sala compta amb 4 portes col·locades a les 4 direccions que en cas d'haver-hi una sala a l'altra banda quan el jugador no estigui en combat s'obrirà. En entrar en una sala que no s'hagi entrat mai i no s'hagin netejat els enemics, les portes es tancaran i no es tornaran a obrir fins que la sala no hi quedi cap enemic. Les portes són totes iguales excepte per les portes de les sales amb els enemics únics, les quals tindran un distintiu que ajudi el jugador a distingir-les de les sales normals.

Figura 37 Portes d'enemics únics

Figura 38 Portes de sales normals

8.2.4.2 Escales

Les escales estan a les sales on estan els enemics únics i només apareixeran un cop l’enemic únic hagi estat derrotat. El jugador en entrar amb contacte amb les escales serà transportat o al següent món o haurà guanyat la partida.

Figura 39 Escales

8.2.5 Interfície d’usuari

En el projecte disposem de diferents interfícies d’usuari que van des del menú principal fins al menú de victòria.

8.2.5.1 Menú principal

Aquest menú que trobem a l’obrir el joc permet al jugador accedir al menú de so a través del botó de “options” o triar personatge amb el botó de “play”.

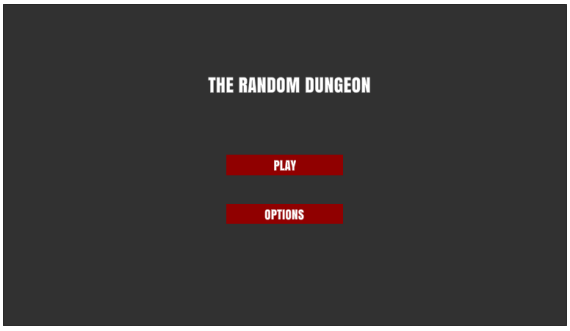


Figura 40 Menú principal

8.2.5.2 Menú de selecció de personatge

El menú de selecció de personatge, el qual s’accedeix a través del menú principal, permet al jugador veure els quatre personatges jugables amb les seves estadístiques. Amb el botó de “X” podrà tornar al menú principal i amb el botó play començarà el joc utilitzant el personatge que estigui actualment en pantalla.

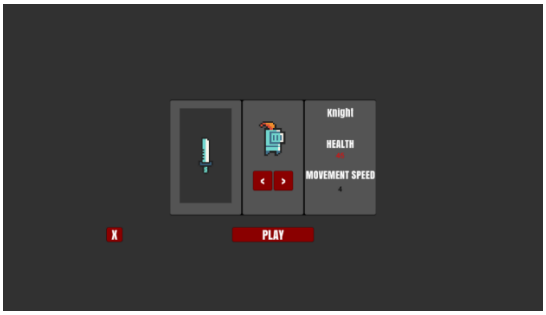


Figura 41 Menú de selecció de personatge

8.2.5.3Menú de partida

Aquest és la interfície gràfica que el jugador veurà durant el joc i podrà veure a la part superior esquerra la barra indicant la vida restant i a la part inferior esquerra un cofre amb el qual obrirà el menú de personatge. El jugador veurà tota la sala que està actualment el personatge que estigui controlant.

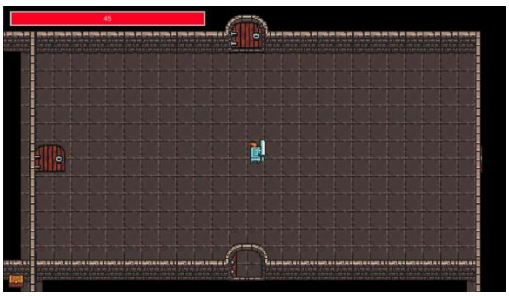


Figura 42 Menú de partida

8.2.5.4 Menú de so

El menú de volum permetrà al jugador modificar el so de la música de fons i dels efectes de so separat. Es pot accedir al menú des del menú principal i a dins el joc a través del botó “options” del menú de pausa.

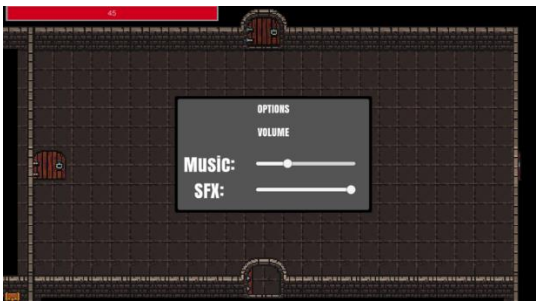


Figura 43 Menu de so

8.2.5.5 Menú de personatge

El menú de personatge és un menú de dins la partida que és pot accedir clicant el cofre de la part inferior esquerra. Aquest menú mostrarà al jugador l’or, l’experiència per pujar de nivell, nivell, vida i permetrà al jugador millorar l’arma en cas de tenir els recursos necessaris.

Figura 44 Menú de personatge

8.2.5.6 Menú de pausa

El menú de pausa s’accedeix a través de la tecla “Escape” del teclat dins una partida, on donarà tres opcions a triar:

- Resume: tornarà a la partida al mateix moment que el jugador ha obert el menú de pausa.
- Options: obre el menú de so
- Exit: Retorna al menú principal

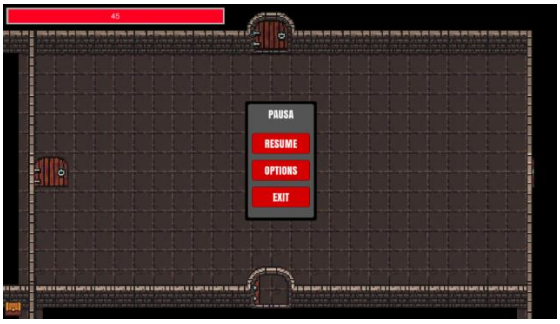


Figura 45 Menú de pausa

8.2.5.7 Pantalla de derrota

Al ser derrotat, al jugador li apareixerà la pantalla de derrota que en clicar en qualsevol lloc de la pantalla tornarà al menú principal.

Figura 46 Pantalla de derrota

8.2.5.8 Pantalla de victòria

Un cop el jugador completa el joc, al jugador li apareixerà la pantalla de victòria. En aquesta pantalla el jugador podrà tornar al menú principal.

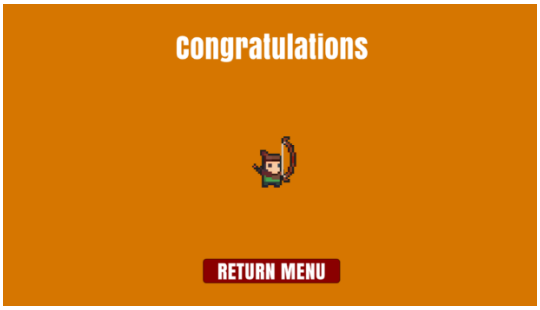


Figura 47 Pantalla de victòria

8.3 Casos d’ús

En aquest apartat veurem els diagrames de casos d’ús plantejats i diagrames d’activitats d’algunes de les funcionalitats que es mostren a l’apartat 8.2.5

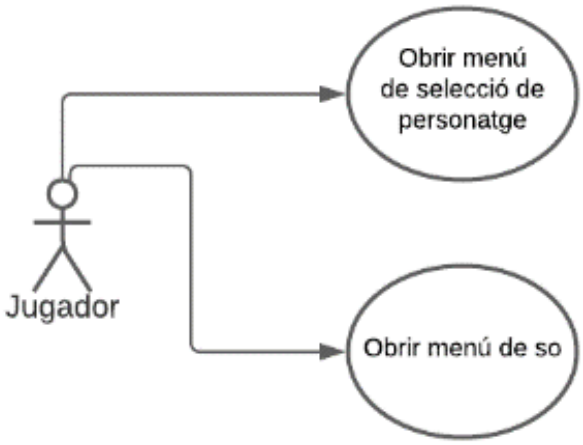


Figura 48 Diagrama de casos d'ús del menú principal

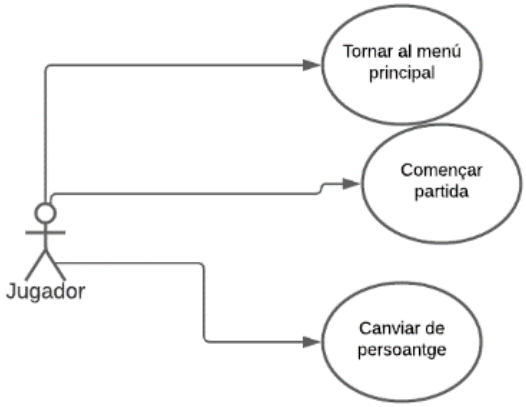


Figura 49 Diagrama de casos d'ús del menú de selecció de personatge

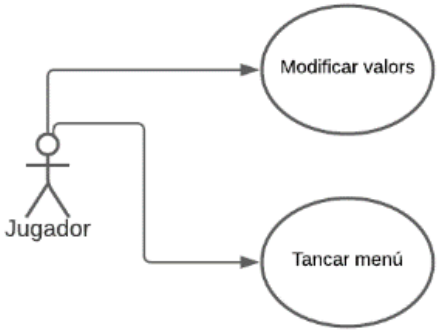


Figura 50 Diagrama de casos d'ús del menú de so

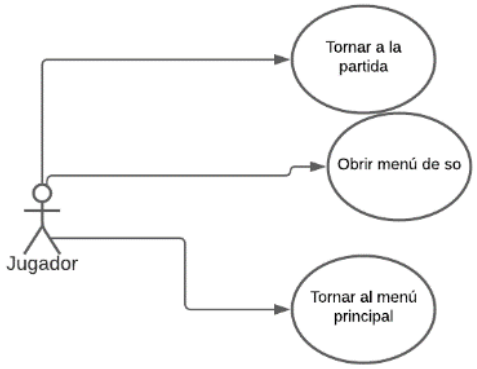


Figura 51 Diagrama de casos d'ús del menú de pausa

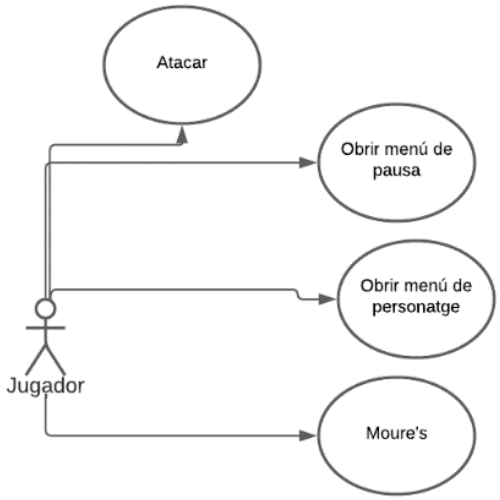


Figura 52 Diagrama de casos d'ús del menú de partida

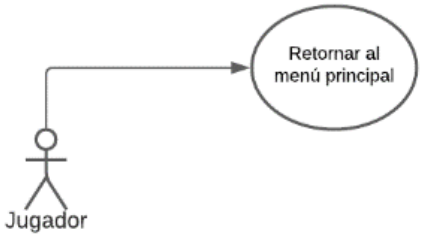


Figura 53 Diagrama de casos d'ús dels menús de derrota i victòria

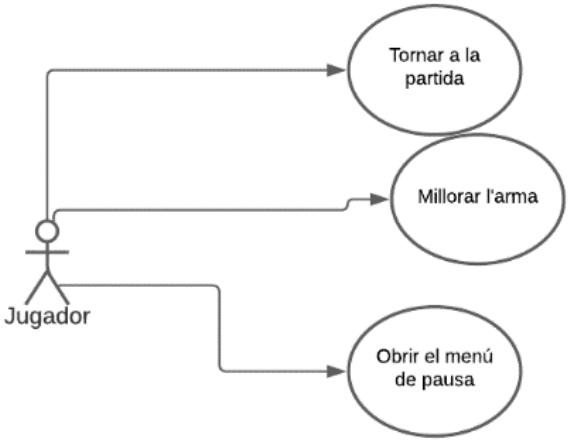


Figura 54 Diagrama de casos d'ús del menú de personatge

8.3.1 Fitxes de casos d’ús

Fitxa de cas d’ús: Obrir menú de selecció de personatge	
Descripció	El jugador selecciona l’opció de jugar del menú principal
Actor	Jugador
Precondició	S’ha iniciat el programa
Flux	1- Escollir l’opció PLAY del menú principal
Postcondició	Obre el menú de selecció de personatge

Fitxa de cas d’ús: Obrir menú de so	
Descripció	El jugador selecciona l’opció d’obrir el menú de so del menú principal o de pausa
Actor	Jugador
Precondició	El jugador es troba al menú principal o menú de pausa
Flux	1- Seleccionar el botó OPTIONS
Postcondició	Obre el menú de so

Fitxa de cas d’ús: Modificar valors	
Descripció	El jugador modifica els valors del volum del videojoc
Actor	Jugador
Precondició	S’ha obert el menú de so
Flux	1- Desplaçar les barres amb els valors de so
Postcondició	Els valors de so s’han modificat

Fitxa de cas d’ús: tornar al menú principal	
Descripció	Retornem al menú principal
Actor	Jugador
Precondició	S’ha obert el menú de selecció de personatge o de pausa o s’ha acabat la partida
Flux	1- Escollir el botó de tornar al menú principal o en qualsevol lloc de la pantalla en cas del menú de derrota
Postcondició	Tornem al menú principal

Fitxa de cas d’ús: Canviar personatge	
Descripció	Canvia el personatge amb el qual es començarà la partida
Actor	Jugador
Precondició	S’ha obert el menú de selecció de personatge
Flux	1- Escollim un dels botons ><
Postcondició	El personatge amb el qual es començarà la partida canvia

Fitxa de cas d’ús: Començar partida	
Descripció	El jugador selecciona el botó de començar partida del menú de selecció de personatge
Actor	Jugador
Precondició	S’ha obert el menú de selecció de personatge
Flux	1- Escull el botó de PLAY del menú de selecció de personatges
Postcondició	Comença la partida

Fitxa de cas d’ús: Obrir menú de pausa	
Descripció	El jugador obre el menú de pausa
Actor	Jugador
Precondició	El jugador esta actualment dins una partida
Flux	1- Prémer la tecla Escape
Postcondició	S’obre el menú de pausa

Fitxa de cas d’ús: Tornar a la partida	
Descripció	El jugador selecciona el botó de tornar a la partida
Actor	Jugador
Precondició	El menú de pausa està obert
Flux	1- Prémer el botó RESUME
Postcondició	Es tenca el menú de pausa

Fitxa de cas d’ús: Obre el menú de personatge	
Descripció	El jugador selecciona el cofre a la part inferior esquerra de la pantalla
Actor	Jugador
Precondició	El jugador esta actualment dins una partida
Flux	1- Prémer el cofre
Postcondició	S’obre el menú de personatge

Fitxa de cas d’ús: Tancar el menú de personatge	
Descripció	El jugador selecciona en qualsevol lloc de la pantalla fora del menú de personatge
Actor	Jugador
Precondició	El menú de personatge està obert
Flux	1- Clicar a qualsevol lloc que no hi hagi el menú de personatge
Postcondició	Es tanca el menú de personatge

Fitxa de cas d’ús: Millorar l’arma	
Descripció	El jugador selecciona l’opció de millorar l’arma
Actor	Jugador
Precondició	El jugador està dins el menú de personatge
Flux	1- Selecciona el botó de millorar l’arma
Postcondició	Si el jugador disposa del suficient or i l’arma no està al nivell màxim es millora l’arma

Fitxa de cas d’ús: Tancar el menú de so	
Descripció	El jugador tanca el menú de so i torna a la pantalla anterior
Actor	Jugador
Precondició	El menú de so està obert
Flux	1- Guarda els valors de so al sistema 2- Tornem al menú anterior
Postcondició	Els so del videojoc es reprodueix amb els nous valors

Fitxa de cas d’ús: Moure’s	
Descripció	El jugador es desplaça pel mapa
Actor	Jugador
Precondició	El jugador està actualment dins d’una partida
Flux	1- Clicar els botons de moviment
Postcondició	El personatge es desplaça cap a la direcció clicada

Fitxa de cas d’ús: Atacar	
Descripció	El jugador realitza un atac
Actor	Jugador
Precondició	El jugador està actualment dins d’una partida
Flux	1- Clicar el botó d’atacar 2- El personatge realitza l’acció d’atacar i activa l’efecte d’atac 3- Si hi ha un enemic que entra en contacte amb l’atac del jugador rep mal.
Postcondició	El jugador ha realitzat un atac

8.3.2 Diagrama d’activitats

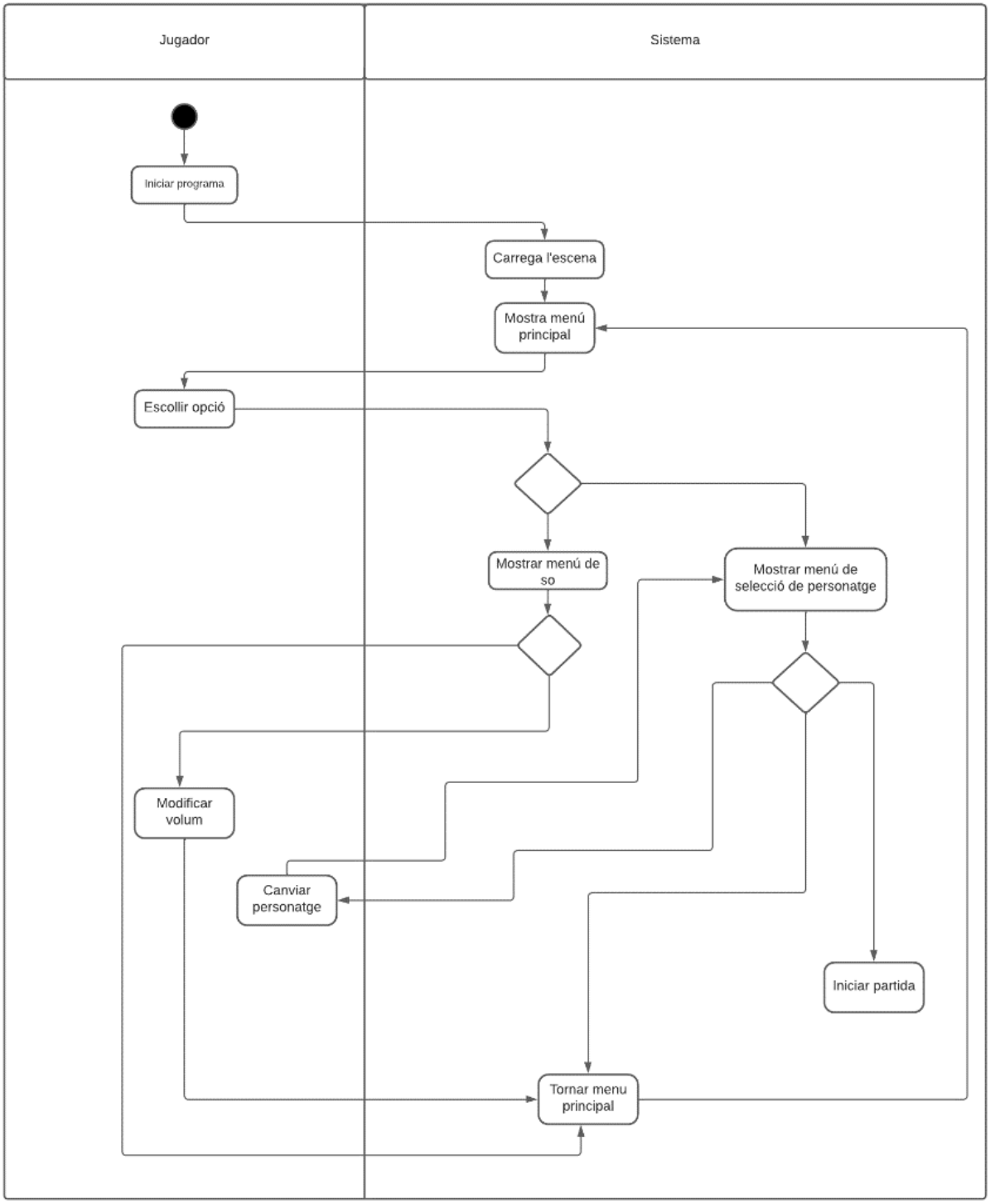


Figura 55 Diagrama d'activitats menú

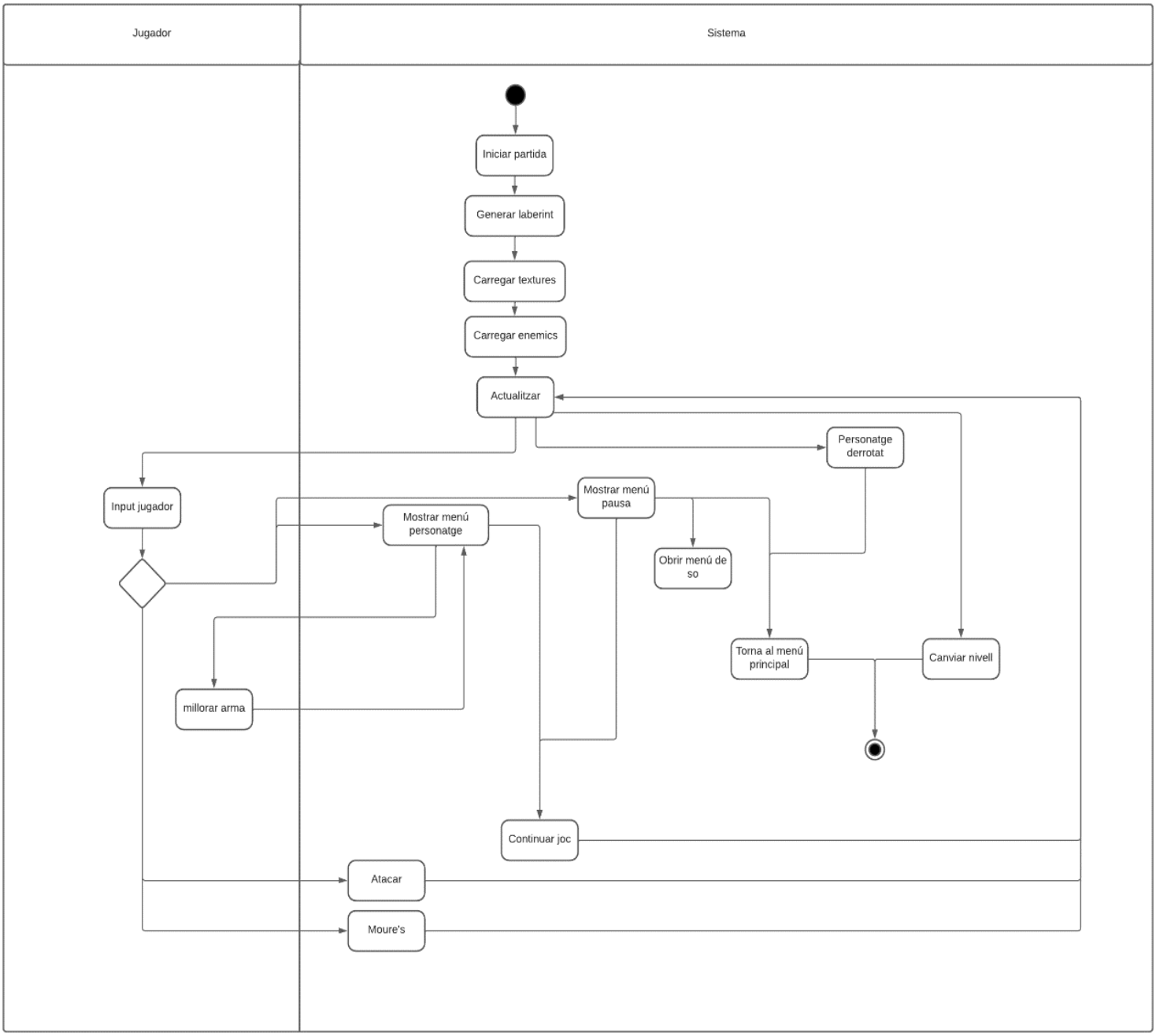


Figura 56 Diagrama d'activitats menú del menú de partida

8.4 Classes i mètodes

8.4.1 AudioManager

Classe responsable dels sons i sorolls del videojoc.

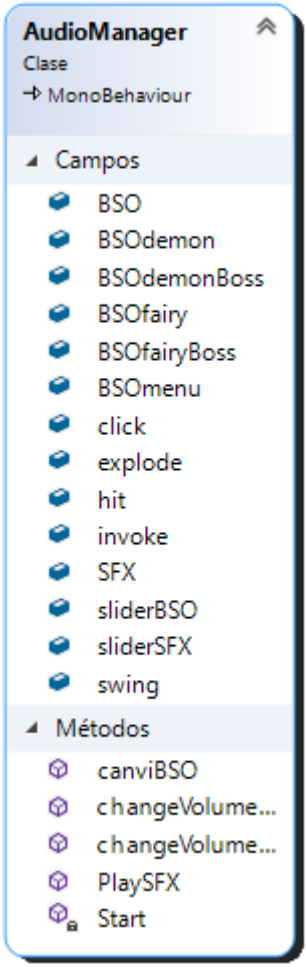


Figura 57 Classe AudioManager

- **Atributs**:

AudioSource BSO: AudioSource que reproduirà el soroll de la banda sonora.

AudioClip BSOmenu: Audioclip amb la banda sonora del menú.

AudioClip BSOdemon: Audioclip amb la banda sonora del 1r mapa.

AudioClip BSOdemonBoss: Audioclip amb la banda sonora del boss del 1r mapa.

AudioClip BSOfairy: Audioclip amb la banda sonora 2nd mapa.

AudioClip BSOfairyBoss: Audioclip amb la banda sonora del boss del 2nd mapa.

AudioClip Click: conté el soroll del botó al ser seleccionat.

AudioClip Explode: Clip amb l’àudio d’explosió de l’última millora de l’arma del mag.

AudioClip Hit: Clip que conté l’àudio de quan rep mal l’enemic o el jugador.

AudioClip Invoke: audio que sonarà quan el Gran dimoni invoqui súbdits.

AudioSource SFX: AudioSource que reproduirà tots els sorolls de les interaccions o efectes de so que passin a la partida.

Slider sliderBSO: Slider que regularà el volum de les bandes sonores.

Slider sliderSFX Slider que regularà el volum dels efectes de so.

AudioClip swing: so que efectuarà el jugador al atacar.

- **Mètodes:**

Start(): S’inicia la reproducció de so de la BSO amb el clip de BSOmenu.

canviBSO(AudioClip name): Canvia el clip de BSO amb la BSO passada per paràmetre i reinicia el funcionament.

playSFX(AudioClip clip): Executa el so del clip passat per paràmetre **.**

changeVolumeBso: Canvia el volum de la banda sonora.

changeVolumeSFX: Canvia el volum dels efectes especials.

8.4.2 Hud

Classe que controla la barra de vida del jugador dins de la partida.

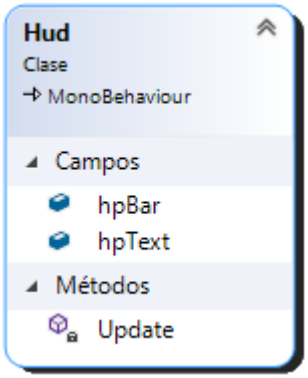


Figura 58 Classe Hud

- **Atributs**

Text hpText: Nombre que sortirà a la barra de vida .

RectTransform hpBar: Barra de la vida restant.

- **Mètodes**

Update(): Canvia els valors de la barra en perdre o guanyar vida **.**

8.4.3 ChraracterMenú

Classe que controla el menú de personatge dins de la partida.

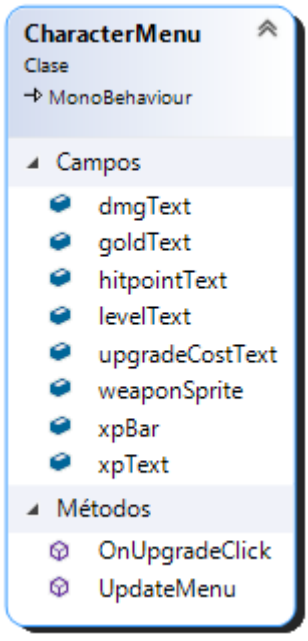


Figura 59 Classe CharacterMenu

- **Atributs:**

Imatge WeaponSprite: Imatge de l’arma que es veu al menú.

RectTransform XpBar: Barra que mostra el progrés completat del nivell.

Text LevelText: Mostra el nivell.

Text HitpointText: Mostra la vida restant.

Text UpgradeCostText: Mostra el cost de la millora d’arma.

Text XpText: Mostra l’experiencia actual i la necessària per pujar de nivell.

Text GoldText: Mostra l’or actual.

Text DmgText: Mostra l’or de l’arma (cos a cos).

- **Mètodes:**

OnUpgradeClick(): Funció que, en seleccionar millorar l’arma, si s’ha pogut millorar, cridarà UpdateMenu per canviar les dades del personatge per les noves.

UpdateMenu(): Actualitza les dades del menú de personatge.

```
UpdateMenu(){
        actualitzarVida()
        actualitzarNivell()
        Si (NivellArma == NivellMaxim)
                ActualitzarTextArma()
        Altrament
                actualitzarTextArma()
        fsi
        si (nivell Actual == nivellMaxim)
                actualitzarTextNivell()
        altrament
                calcularExperienciaActual()
                actualitzatTextNivell()
        fsi
}
```

8.4.4 CharacterSelection

Controla el menú de selecció de personatge abans de començar la partida.

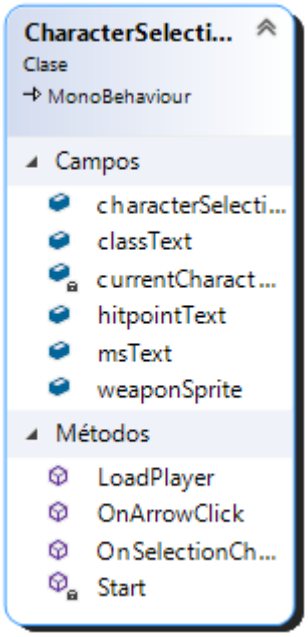


Figura 60 Classe CharacterSelection

- **Atributs:**

Text msText: Mostra la velocitat del personatge.

Text hitpointText: Mostra la vida del personatge.

TextMeshProUGUI classText: Mostra el nom del personatge.

Int currentCharacterSelection: Conté el nombre de la posició de la llista del GameManager del personatge actualment seleccionat.

Image characterSelectionSprite: Conté la foto del personatge seleccionat actualment.

Image weaponSprite: Conté la foto de l’arma del personatge seleccionat actualment.

- **Mètodes:**

Start(): Emplena tots els atributs amb els valors per defecte.

LoadPlayer(): Instància el GameObject del personatge seleccionat al seleccionar començar partida.

OnArrowClick(bool right**):** Canvia el número de currentCaracterSelection i crida OnSelectionChange.

OnSelectionChange(): Canvia les dades del menú per les del nou personatge seleccionat.

8.4.5 Pausa

Controla el menú de pausa de dintre la partida.

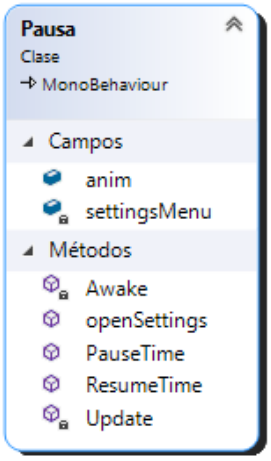


Figura 61 Classe Pausa

- **Atributs:**

Animator Anim: Té l’animació que mostra i amaga el menú de pausa.

Animator settingsMenu: Té l’animació que mostra i amaga el menú de so.

- **Mètodes:**

Awake(): Agafa l’animació que obre el menú de so.

openSettings(): Activa el trigger que obrirà el menú de so.

PauseTime(): Pausa la partida posant el timeScale a 0.

ResumeTime(): Torna a posar en marxa la partida en posar el timeScale a 1.

Update(): Si es polsa la tecla Escape obrirà el menú de pausa i pararà la partida.

8.4.6 ReturnMenú

Classe que torna al menú principal del joc i en cas d’estar enmig d’una partida o haver-la acabat esborra totes dades.

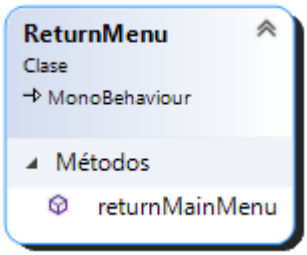


Figura 62 Classe ReturnMenu

- **Metodes:**

returnMainMenu(): Borra tots els GameObjects que s’han creat que tenien la classe Mantenir i crida la funció del GameManager per canviar d’escena a la del menú principal.

8.4.7 FloatingTextManager

Classe que crea i modifica els textos flotants que apareixen per pantalla.

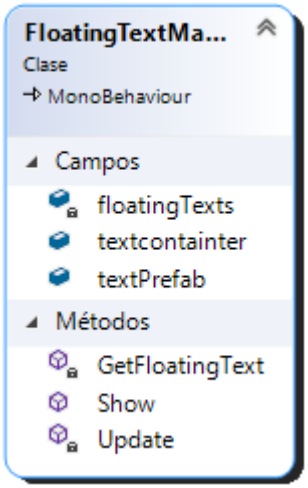


Figura 63 Classe FloatingTextManager

- **Atributs:**

List<FloatingText> floatingTexts: Conté una llista amb tots els FloatingText creats.

GameObject textcontainer: Conté el contenidor on es crearan els FloatingText **.**

GameObject textPrefab: Conté el prefab del text que s’instanciarà com a FloatingText.

- **Mètodes:**

FloatingText GetFloatingText(): Instància i retorna un nou FloatingText.

Show(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration): Agafa un FloatingText creat per GetFloatingText() i li posa les dades passades per paràmetre.

Update(): Actualitza l’estat de tots els FloatingText.

8.4.8 FloatingText

Classe amb els paràmetres i funcions de l’objecte FloatingText.

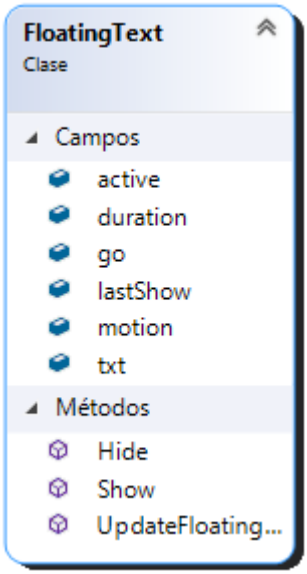


Figura 64 Classe FloatingText

- **Atributs:**

Bool Active: Marca si el text està actualment mostrant-se.

Float Duration: Duració que tindrà el FloatingText mostrant-se.

GameObject Go: GameObject Text instanciat al FloatingTextManager.

Float lastShow: Variable que guarda el temps de quan s’ha començat a mostrar el FlotatingText.

Vector3 motion: Guarda la direcció en què es mourà el text.

Text txt: Missatge a mostrar

- **Mètodes:**

Hide(): Amaga el GameObject.

Show(): Mostra el GameObject.

UpdateFloatingText(): Actualitza la posició de l’objecte que porta el text i en superar la duració establerta crida la funció Hide().

8.4.9 UniversalSounds

Classe que posa els SFX dels botons a l’escena actual.

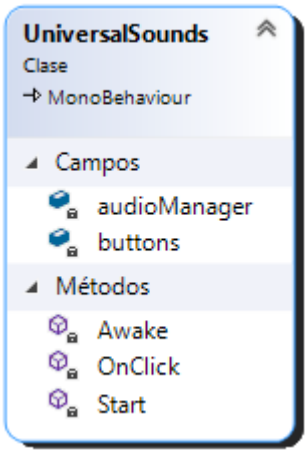


Figura 65 Classe UniversalSounds

- **Atributs:**

AudioManager audioManager: Variable on es buscarà i posarà l’AudioManager escena.

GameObject[] buttons: Array on es guardarà tots els botons de l’escena.

- **Mètodes:**

Awake(): Busca l’AudioManager d’aquesta escena i ho posa a la variable audioManager.

OnClick(): Crida la funció PlaySFX de l’audioManager amb l’audioClip click.

Start(): Busca tots els botons de l’escena i, per a cada un d’ells, crea un Event que espera a ser clicat per cridar la funció onClick.

8.4.10 GameManager

Classe que guarda i modifica l’estat de la partida i el personatge.

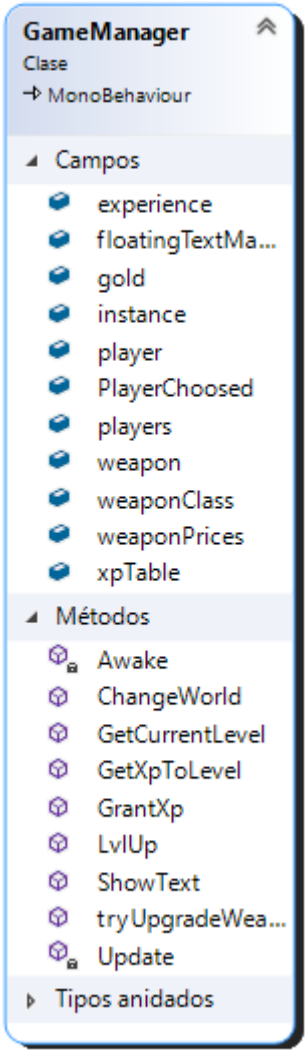


Figura 66 Classe GameManager

- **Atributs:**

Int Experience: Guarda l’experiència del personatge.

FloatingTextManager floatingTextManager: Guarda el GameObject que permetrà crear textos.

Int gold: Guarda l’or del jugador.

GameManager instance: Permet cridar les funcions del GameManager des de qualsevol classe.

Player Player: Guarda el personatge instanciat que s’hagi seleccionat per jugar la partida.

Int PlayerChosed: Guarda la posició del personatge seleccionat dins la llista Players per jugar la partida.

List<GameObjects> Players: Llista amb tots els personatges que pot controlar el jugador.

Weapon weapon: Arma del personatge que està jugant la partida.

List<weaponPath> weaponClass: Llista d’estructures on es guarden totes les millores de cada arma.

List<int> weaponPrices: Llista amb els preus que costen les millores de l’arma.

List<Int> xpTable: Llista amb l’experiència necessària per pujar de nivell.

- **Mètodes:**

Awake(): Instancia el GameManager.

Update(): Comprova si Player, Weapon i FloatingTextManager són null, en cas de ser-ho els buscaria.

ChangeWorld(string newScene): Canvia l’escena del joc per l’escena amb el nom passat per paràmetre.

Int GetCurrentLevel(): Retorna el nivell actual del jugador.

Int GetXpToLevel(int lvl): Retorna l’experiència necessària per pujar de nivell.

GrantXp(xp): Afegeix l’experiència passada per paràmetre al jugador, i en cas de superar l’experiència necessària per pujar de nivell, crida la funció LvlUp().

LvlUp(): Puja la vida del personatge a la vida del nivell pujat i recupera tota vida.

ShowText(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration): Crida la funció per crear un FloatingText enviant els paràmetres rebuts.

tryUpgradeWeapon(): Comprova si es pot millorar l’arma. En cas que es pugui resta l’or que costa la millora i crida la funció UpgradeWeapon() de l’arma del jugador.

8.4.11 SpawnerData

ScriptableObject que guardarà els enemics a invocar a cada sala, amb el màxim i mínim d’enemics a invocar.

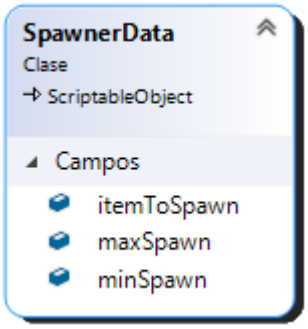


Figura 67 Classe SpawnerData

- **Atributs:**

List<GameObject> ItemToSpawn: Llista amb els enemics a invocar.

Int maxSpawn: Màxim nombre d’enemics a invocar a una sala.

Int minSpawn: Mínim nombre d’enemics a invocar a una sala.

8.4.12 DungeonGeneratorData

ScriptableObject que conté l’interval mínim i màxim de sales que crearà el laberint i la quantitat de rastrejadors que utilitzarà.

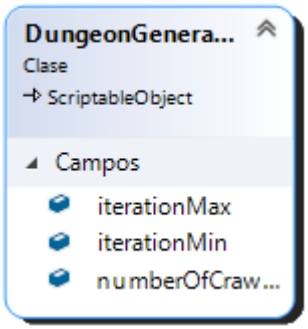


Figura 68 Classe DungeonGeneratorData

- **Atributs:**

Int IterationMax: Nombre màxim de sales a invocar.

Int IterationMin: Nombre mínim de sales a invocar.

Int numberOfCrawlers: Nombre de rastrejadors a utilitzar.

8.4.13 DungeonGenerator

Aquesta Classe agafa el vector de posicions creat pel DungeonCrawler.

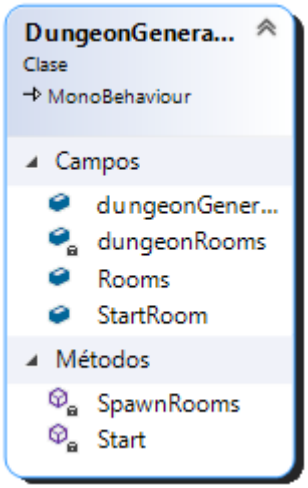


Figura 69 Classe DungeonGenerator

- **Atributs:**

DungeonGenerationData dungeonGeneratorData: Guarda les dades de l’ScriptableObject DungeonGenerationData per passar-les al DungeonCrawlesController.

List<Vector2Int> dungeonRooms: Guarda la llista de vectors retornada pel DungeonCrawlerController.

List<String> Rooms: Guarda la llista dels noms de les sales.

String StartRoom: Guarda el nom de la sala de començament.

- **Mètodes:**

Start(): Crida la funció GenerateDungeon de la classe DungeonCrawlerController i es guarda la llista de vectors amb la posició de cada sala. Després crida la funció SpawnRooms.

SpawnRooms(IEnumerable<Vector2Int> rooms): Crida la funció del RoomController per crear la sala de començament amb la posició 0,0 i, més tard, crida aquesta mateixa funció per a cada posició del vector2Int rooms passat per paràmetre amb una sala aleatòria de la llista Rooms.

8.4.14 DungeonCrawler

Aquesta classe rep un Dictionary amb diferents direccions i en retorna una aleatòriament.

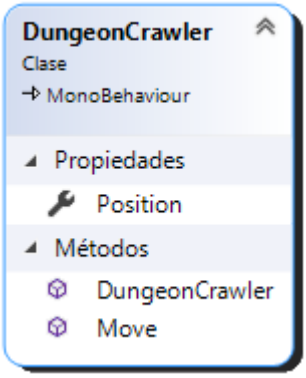


Figura 70 Classe DungeonCrawler

- **Atributs:**

Vector2Int Position: Vector(0,0) per poder en cridar la funció move() retornar un vector que es mogui una posició.

- **Mètodes:**

DungeonCrawler(Vector2Int startPos): Canvia la posició del dungeonCrawler a (0,0).

Vector2Int Move(Dictionary<Direction, Vector2Int> directionMovementMap): Rep un dictionary amb 4 valors que contenen 4 vector2Int cada un amb una direcció diferent, la funció n’agafa un aleatòriament i retorna el vector Position sumant el vector direcció del vector triat.

8.4.15 DungeonCrawlerController

Aquesta classe genera una llista de Vector2Int que on cada vector representa la posició d’una sala.

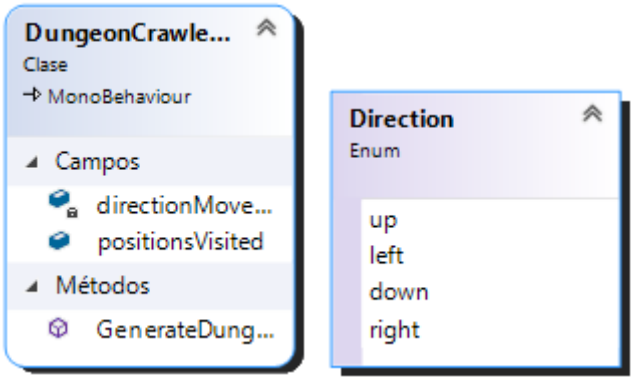


Figura 71 Classe DungeonCrawlerController

- **Atributs:**

Dictionary<Direction, Vector2Int> directionMovementMap: Dictionary que guarda les posicions on es pot moure el DungeonCrawler.

List<Vector2Int> positionsVisited: Vector amb les posicions de les sales que es crearan.

- **Mètodes:**

List<Vector2Int> GenerateDungeon(): Rep les dades de l’interval mínim i màxim de les sales que s’han de crear i el nombre de crawlers que haurà de fer servir i omple la llista positionsVisited amb un nombre entre el mínim i màxim de sales a crear que retornarà un cop completat.

8.4.16 Room

Un cop creada la sala a RoomController, aquesta classe s’encarregarà d’executar les accions necessàries a aquesta habitació que cridi el RoomController.

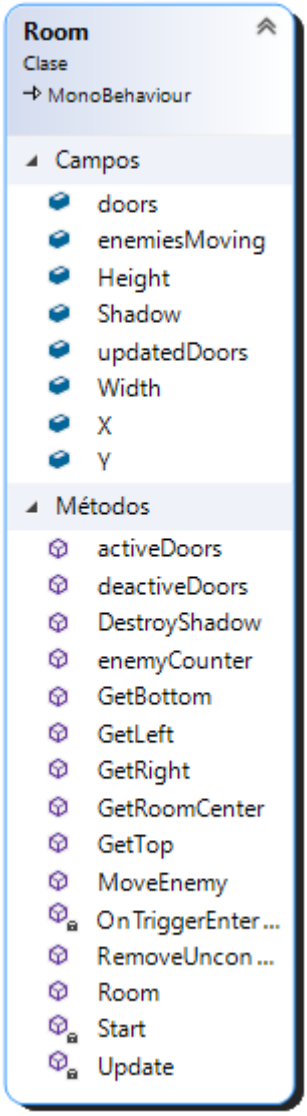


Figura 72 Classe Room

- **Atributs**

List<Door> doors: Guarda els GameObjects Door de l’habitació.

Bool enemiesMoving: Especifica si els enemics de la sala han començat a moure’s.

Float Height: Guarda l’altura de l’habitació.

Float Width: Guarda l’amplada de l’habitació.

GameObject Shadow: GameObject que tapa amb una capa de color negre l’habitació.

Bool UpdatedDoors: Indica si s’ha actualitzat les portes de la sala de boss.

Int X Posició horitzontal respecte la sala d’inici.

Int Y: Posició vertical respecte la sala d’inici.

- **Mètodes:**

activeDoors(): Tanca les portes.

deactiveDoors(): Obre les portes.

DestroyShadow(): Elimina l’objecte que impedia veure la sala.

Int enemyCounter(): Retorna els enemics que queden a la sala.

GetBottom(): Retorna si la sala d’avall. En cas de no haver-n’hi torna null.

GetLeft(): Retorna si la sala de l’esquerra. En cas de no haver-n’hi torna null.

GetRight(): Retorna si la sala de la dreta, en cas de no haver-n’hi torna null.

GetTop(): Retorna si la sala d’amunt, en cas de no haver-n’hi torna null.

Vector3 GetRoomCenter(): Retorna el centre de l’habitació.

MoveEnemy(): Activa el moviment dels enemics.

noEnemies(): Obre les portes si no queden enemics.

OnTriggerEnter2D (Collider2D other): En detectar que el jugador entra a l’habitació, crida la funció OnPlayerEnterRoom().

RemoveUnconnectedDoors(): Torna totes les portes de la sala que no portin a una altre habitació en no funcionals.

Room(int x, int y): Constructor de la classe que posa els valors de X i Y.

Start(): Comprova que la sala estigui correctament creada, afegeix les portes i crida la funció RegisterRoom().

Update(): Comprova si l’habitació és la d’un enemic únic i en ser el cas el i no tret les portes que no connecten enlloc crida la funció RemoveUnconnectedDoors().

8.4.17 RoomController

Crea les habitacions a les posicions necessàries i crida les funcions per cada acció de l’habitació.

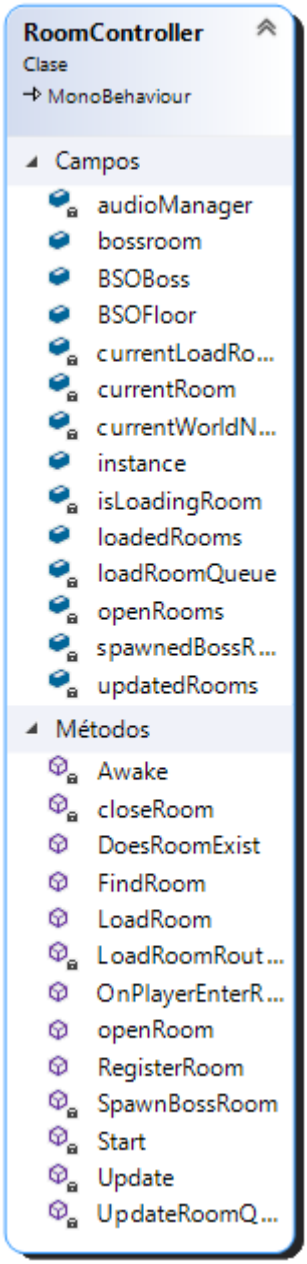


Figura 73 Classe RoomController

- **Atributs:**

AudioManager audioManager: Guarda el controlador de la música.

String bossroom: Nom de la sala que contindrà l’enemic únic d’aquest nivell del laberint.

AudioClip BSOBoss: Banda sonora de l’enemic únic d’aquest nivell del laberint.

AudioClip BSOFloor: Banda sonora d’aquest nivell del laberint.

RoomInfo currentLoadRoomData: Posició i nom de la sala que s’està carregant actualment.

Room currentRoom: Guardarà la sala en què es trobi el jugador.

String currentWorldName: Nom del món actual per carregar les habitacions d’aquest món.

RoomController instance: Permet cridar les funcions del RoomController des de qualsevol classe.

Bool isLoadingRoom: Indica si s’està carregant una habitació actualment.

Queue<RoomInfo> loadRoomQueue: Cua amb el nom i posició de totes les habitacions que s’han de crear.

Bool openRooms: Indica si l’habitació està oberta.

Bool spawnedBossRoom: Indicador que serveix per saber si l’habitació de l’enemic únic s’ha creat.

Bool updatedRooms: Indica si s’han eliminat les portes que no porten a cap sala.

- **Mètodes**

Awake(): Instància la classe i canvia la banda sonora a l’audiomanager per la banda sonora d’aquesta habitació.

closeRoom(): tanca les habitacions.

Bool DoesRoomExist(int x, int y): Retorna si existeix una habitació a la posició passada per paràmetre.

Room FindRoom(int x, int y): Busca i retorna l’habitació a la posició passada per paràmetre.

LoadRoom(string name, int x, int y): Rep una posició i un nom i crea un objecte RoomInfo amb aquesta informació que encua a loadRoomQueue.

LoadRoomRoutine(RoomInfo info): Rep un objecte RoomInfo i carrega l’escena de la sala amb el nom guardat a l’objecte.

OnPlayerEnterRoom(Room room): Canvia l’habitació actual del jugador, posa la càmera a la posició de la sala passada per paràmetre i activa tots els objectes dins d’aquesta.

openRoom(): Obre les habitacions.

RegisterRoom(Room room): Rep una habitació ja creada i si no existeix una habitació amb la mateixa posició, la guarda a la llista d’habitacions.

SpawnBossRoom(): Agafa la ubicació de l’última sala, la borra i hi crea una sala amb un enemic únic.

Update(): Crida UpdateRoomQueue() i en cas que totes les sales estiguin creades i la sala en què es trobi el jugador actual no quedin enemics, obrirà les portes i farà apareixer les escales al següent nivell si el jugador es troba a la sala de l’enemic únic.

UpdateRoomQueue(): Desencua l’última habitació de la cua loadRoomQueue() i crida la funció per carregar l’habitació amb la informació aconseguida. Si totes les sales estan creades, crida la funció SpawnBossRoom() i si ja ha cridat aquesta funció crida per a cada Habitació la funció RemoveUnconnectedDoors().

```
UpdateRoomQueue(){
Si (estaCarregantSala)
        Tornar
Fsi
Si (CuadeSales == 0)
        Si(!SalaEnemicÚnicCreada)
                CrearSalaEnemicÚnic()
        Altrament Si(SalaEnemicÚnicCreada i !HabitacionsActualitzades)
                actualitzatHabitacions()
        fsi
        tornar
fsi
carregarSala()
```

8.4.18 Door

Classe que obre o tanca la porta.

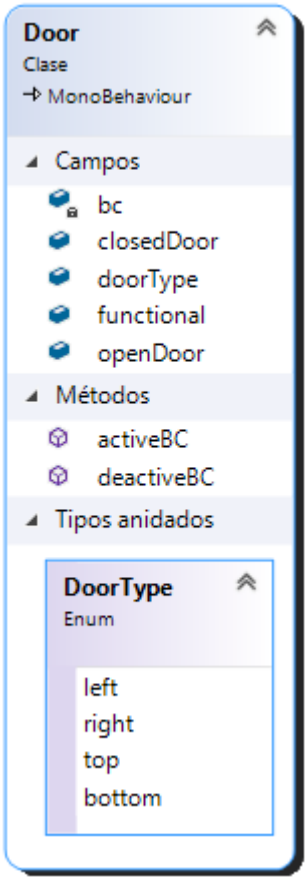


Figura 74 Classe Door

- **Atributs:**

BoxCollider2D BC: Física de l’objecte.

Sprite closedDoor: Imatge de la porta tancada.

Sprite openDoor: Imatge de la porta oberta.

Bool funcional: Indica si la portà s’obrirà o no té cap sala a l’altra banda.

Enum DoorType: Indica quina de les 4 portes és(dreta, esquerra, amunt, avall).

- **Mètodes:**

activeBC(): Tanca la porta.

deactiveBC(): Obre la porta.

8.4.19 RoomInfo

Guarda el nom i posició de l’habitació.

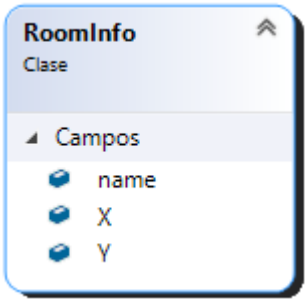


Figura 75 Classe RoomInfo

- **Atributs:**

String Name: Nom de la sala.

Float X: Posició de la sala horitzontalment.

Float Y: Posició de la sala verticalment.

8.4.20 GridController

Genera la zona on podran aparèixer enemics.

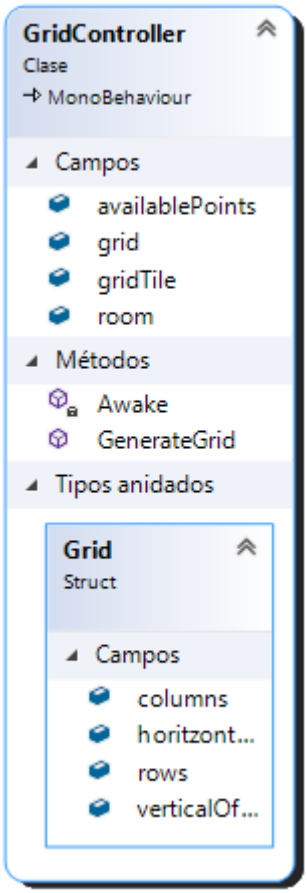


Figura 76 Classe GridController

- **Atributs:**

List<Vector2> availablePoints: Llista de punts on podran aparèixer enemics.

Grid grid: Zona on podran aparèixer enemics.

GameObject gridTile: GameObject que representarà una posició on podran apareixer enemics.

Room room: Habitació a la qual està el grid.

- **Mètodes:**

Awake(): Agafa l’objecte Room on està el grid, agafa la mida que ha de tenir la zona d’invocació d’enemics i crida GenerateGrid().

GenerateGrid(): Genera tots els punts d’aparició d’enemics dins el rang de grid i els guarda a la llista availablePoints.

8.4.21 ObjectRoomSpawner

Fa aparèixer els enemics a la zona marcada per la classe GridController.

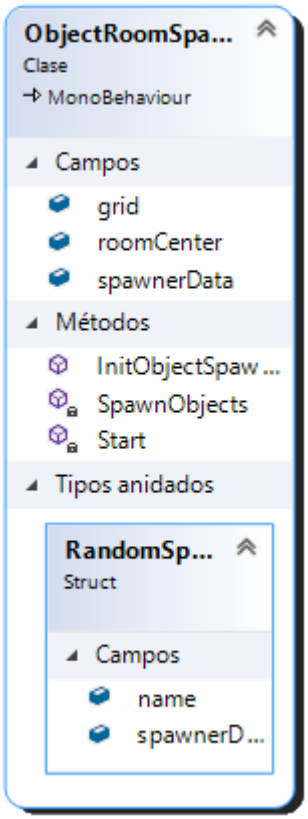


Figura 77 Classe ObjectRoomSpawner

- **Atributs**

GridController Grid: Objecte amb totes les posicions on pot aparèixer un enemic.

Vector2 roomCenter: Vector amb el centre de l’habitació.

RandomSpawner[] spawnerData: Llista d’enemics amb structs que tenen els enemics disponibles per a invocar i la quantitat mínima i màxima que es podran invocar.

- **Mètodes:**

initObjectSpawner(): Agafa tots els objectes de spawnerData i crida la funció SpawnObjects.

SpawnObjects(RandomSpawner sata): Tria aleatòriament un nombre entre el mínim i màxim que està al struct passat per paràmetre i invoca tants enemics com el nombre triat.

8.4.22 BossSpawn

Invoca els dimonis del Gran Dimoni.

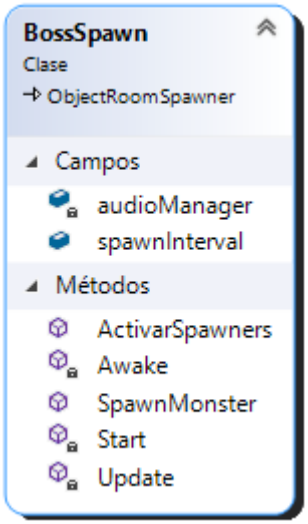


Figura 78 Classe BossSpawn

- **Atributs**

AudioManager audioManager: Guarda el controlador de la música.

Float spawnInterval: Segons que passaran entre cada invocació.

- **Mètodes:**

ActivarSpawners(): Activa la funció que invocarà enemics.

Awake(): Guarda el controlador de so al audioManager.

SpawnMonster(): Si el Boss continua viu, invocarà enemics.

Start(): Guarda el centre de l’habitació.

8.4.23 CameraController

Canvia la posició de la càmera.

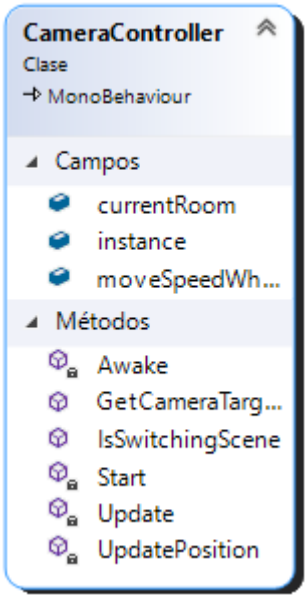


Figura 79 Classe CameraController

- **Atributs:**

Room CorrentRoom: currentRoom: Guarda l’habitació en què es troba el jugador actualment.

CameraController Instance: Instància de la classe.

Float moveSpeedWhenChange: Velocitat de la càmera.

- **Mètodes**

Awake(): guarda la instància.

Upadate(): Crida la funció UpdatePosition().

UpdatePosition(): Canvia la posició de la càmera si és necessari.

Vector3 GetCameraTargetPosition(): Retorna la posició del centre de l’habitació on es troba el jugador.

Bool IsSwitchingScene(): retorna si la posició de la càmera és la mateixa que la que retorna GetCameraTargetPosition().

8.4.24 ChangeLocationPath

Crea i canvia de posició la zona on A* buscarà camins fins al jugador.

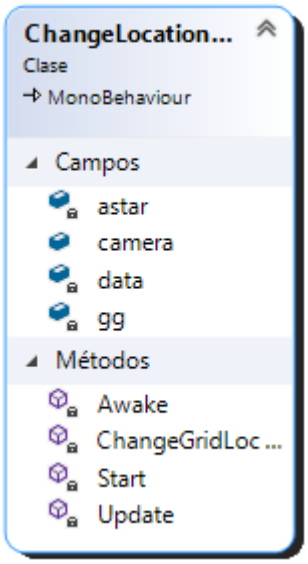


Figura 80 Classe ChangeLocationPath

- **Atributs**:

AstarPath astar: Gurarda el GameObject que contindrà l’objecte astarpath.

CameraController camera: Objecte controlador de la càmera.

Pathfinding.AstarData data: Guarda les dades de l’objecte astarpath.

Pathfinding.GridGraph gg: Guarda els paràmetres del graph.

- **Mètodes**:

Start(): Busca el GameObject AstarPath, crea un graph nou i modifica les seves dades.

Update(): Comprova si la càmera ha canviat d’habitació, de ser el cas crida ChangeGridLocation().

ChangeGridLocation(): Canvia el graph de posició.

8.4.25 Fighter

Classe que controla la vida i combat de qualsevol objecte que pugui combatre.

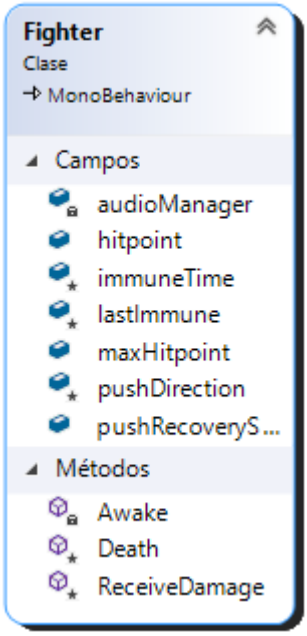


Figura 81 Classe Fighter

- **Atributs**

AudioManager audioManager: Guarda el controlador de so.

Int hitpoint: Guarda la vida actual.

Int maxHitpoint: Guarda la vida màxima.

Float pushRecoverySpeed: Guarda la velocitat a la qual es recuperarà del retrocés d’un atac.

Float immuneTime: Temps després de rebre un atac en el qual no podrà rebre mal.

Float lastImmune: Temps on ha començat el temps d’immunitat.

Vector2 pushDirection: direcció a la qual envia el retrocés de l’atac.

- **Mètodes**

Awake(): Guarda el controlador de so a l’audioManager.

ReceiveDamage(): Si no està dins el temps d’immunitat, rebrà dany, i en cas de baixar la vida més avall de 0, cridarà la funció Death().

Death(): Funció sobreescrita per les classes fill.

8.4.26 EnemyAI

Guarda totes les estadístiques de l’enemic que no convergeixen amb les del jugador i s’encarrega de controlar el moviment.

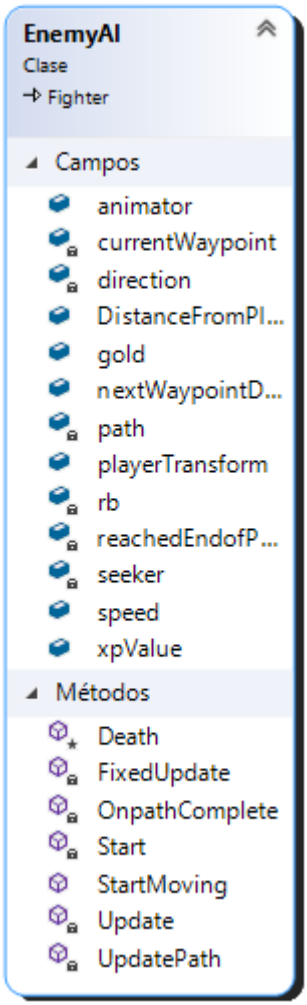


Figura 82 Classe EnemyAI

- **Atributs**:

Animator animator: Controlador d’animació.

Float currentWaypoint: Dins els punts del path calculat, aquí es guarda el punt en què es troba l’enemic.

Vector2 Direction: Direcció a la qual es vol moure l’enemic.

Float DistanceFromPlayer: Distància entre l’enemic i el jugador.

Int gold: Or que deixarà el morir.

Float nextWaypointDistance: Dins els punts del path calculat, aquí es guarda el pròxim punt al que es dirigirà l’enemic.

Path path: Guarda el camí format per diferents punts per arribar al punt desitjat.

Transform PlayerTransform: Posició del jugador.

Rigidbody2D rb: Component Rigidbody de l’enemic.

Bool reachedEndofPath: Indica si ha arribat al final del camí.

Seeker seeker: Calcula el camí més ràpid per arribar al jugador o la posició desitjada.

Float speed: Velocitat de l’enemic.

Int xpValue: Experiència que deixarà en morir.

- **Mètodes**:

Start(): Agafa i guarda els components del Rigidbody, Seeker i posició del jugador a les variables.

Update(): Canvia el sprite de l’enemic perquè miri al jugador.

UpdatePath(): Calcula un camí fins al jugador.

FixedUpdate(): Mou l’enemic seguint el camí de la variable path.

OnPathComplete(): Posa el camí calculat a la variable path.

StartMoving(): Activa el moviment de l’enemic.

Death(): Elimina el GameObject i dona al jugador l’or i l’esperiència corresponent.

```
FixedUpdate(){
SI (CAMI==NULL) llavors
        Tornar
fsi
Si (DistànciaJugadorEnemic < DistanciadelJugador)
        Direcció = -(direcciópróximaPosició)
Altrament
        Direcció = direcciópróximaPosició
fsi
Si (direcciódeRetrocés>0.3)
        Force += direcciódeRetrocés-Direcció
aplicarForce()
Si (distància Punt < mínimaDistanciaPerProximPunt)
        punt Actual++
fsi
}
```

8.4.27 Player

Guarda totes les estadístiques del jugador que no convergeixen amb les de l’enemic i s’encarrega de controlar el moviment.

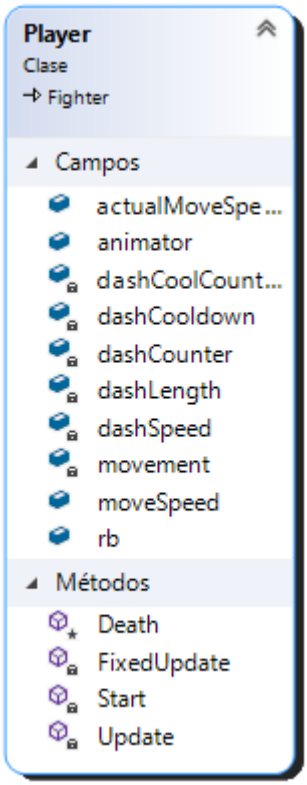


Figura 83 Classe Player

- **Atributs**:

Float actualMoveSpeed: Velocitat actual del jugador.

Animator animator: controlador d’animació.

Float dashCoolCounter: Contador del temps que falta fins a tenir el dash disponible.

Float dashCounter: Contador del temps de duració del dash.

Float dashCooldown: Temps d’espera per poder tornar a fer servir el dash.

Float dashLength: Duració del dash.

Float dashSpeed: Velocitat del dash.

Vector2 movement: Vector amb la direcció del moviment del personatge.

Float moveSpeed: Velocitat de moviment del personatge.

Rigidbody2D rb: Component rigidbody del personatge.

- **Mètodes:**

Start(): Posa la velocitat del personatge a la velocitat actual del personatge.

Update(): Canvia la imatge del personatge perquè miri cap on s’està movent i s’encarrega de canviar la velocitat quan es fa un dash.

FixedUpdate(): Mou el personatge.

Death(): Mostra la pantalla de derrota.

8.4.28 MeleeTrigger

Controla els patrons d’atac de l’enemic únic rei feèric.

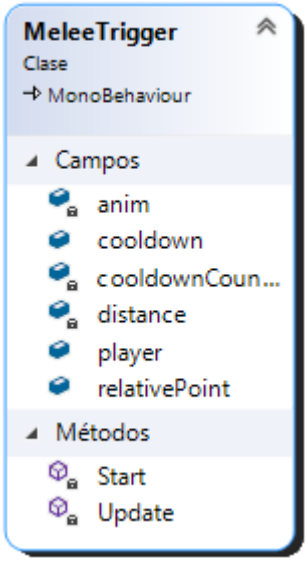


Figura 84 Classe MeleeTrigger

- **Atribut:**

Animator anim: Controlador de l’animador.

Float cooldown: Interval mínim entre cada atac.

Float cooldownCounter: Contador fins a poder tornar a fer l’atac.

Float distance: Distància a la qual s’activa l’animació de l’atac.

Vector2 relativePoint: Direcció de l’atac.

- **Mètode:**

Start(): agafa el component d’animació.

Uptade(): si el personatge s’acosta a certa distància i l’atac no està en cooldown, atacarà el jugador.

8.4.29 Damage

Struct que guarda el mal i retrocés de l’atac.

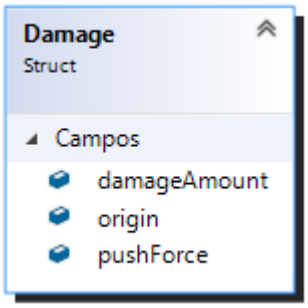


Figura 85 Struct Damage

Vector3 origin: Posició de l’atacant.

Int damageAmount: mal a treure.

Float pushForce: Força de retrocés de l’atac.

8.4.30 Collidable

Controla totes les col·lisions del joc.

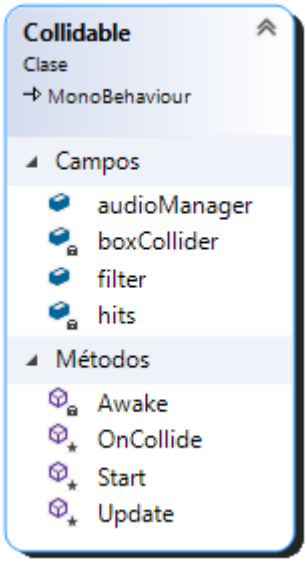


Figura 86 Classe Collidable

- **Atributs**

AudioManager audioManager: Controlador de so.

BoxCollider2D boxCollider: Guarda el collider de l’objecte.

Collider2D[] hits: Guardarà el nombre de colliders que es superposen.

ContactFilter2D filter: Filtre dels colliders que interessin.

- **Mètode**

Awake(): Guarda el controlador de l’AudioManager a la funció.

Start(): Guarda el BoxCollider2D a la variable.

Update(): Vigila si 2 o més objectes col·lideixen entre si.

OnCollide(): Funció que sobreescriurà als fills que farà l’acció en col·lidir.

8.4.31 Weapon

Controla totes les Estadístiques i accions de l’arma.

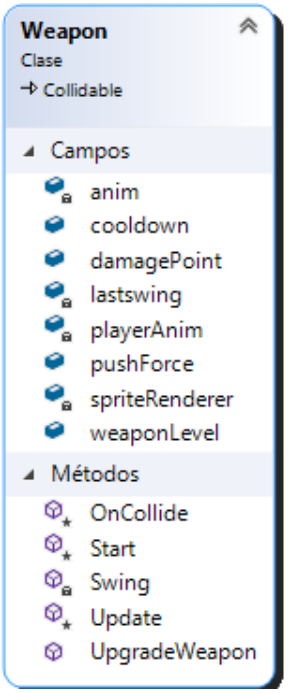


Figura 87 Classe Weapon

- **Atributs**:

int weaponLevel: Nivell de l’arma.

SpriteRenderer spriteRenderer: Imatge de l’arma portada actualment.

Animator playerAnim: Controlador d’animació del personatge.

float cooldown: Temps entre un atac i l’altre.

Animator anim; Controlador d’animació.

float lastswing: Últim atac fet.

Int[] damagePoint: Mal de l’arma a tots els nivells.

Float[] pushForce: Retrocés de l’arma a tots els nivells.

- **Mètodes**:

Start(): Agafa el sprite, controlador d’animació i controlador d’animació del personatge i els posa a les variables.

Update(): Canvia el sentit de l’arma depenent d’on es mogui el jugador i si el jugador selecciona l’opció d’atacar i no està en cooldown, cridarà la funció Swing().

OnCollide(): Si toca un enemic li infringeix mal i el retrocés de l’arma.

Swing(): Activa l’animació d’atac.

UpgradeWeapon(): Puja el weapon level i canvia el sprite pel del nou nivell.

8.4.32 EnemyHitbox

Controla el mal de l’enemic i les seves col·lisions.

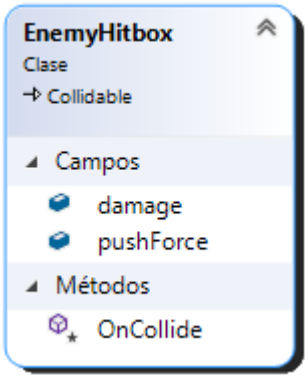


Figura 88 Classe EnemyHitbox

- **Atribut:**

Int damage: mal que farà l’enemic en col·lidir.

Float pushForce: Retrocés aplicat en col·lidir.

- **Mètode:**

OnCollide(): Si la col·lisió és amb el jugador, li aplica el mal i el retrocés.

8.4.33 EnemyProjectile

Controla el mal i col·lisions dels projectils disparats pels enemics.

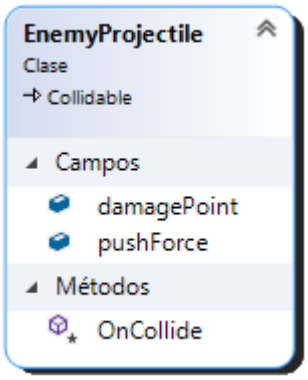


Figura 89 Classe EnemyProjectile

- **Atribut:**

Int damage: mal que farà l’enemic al col·lidir.

Float pushForce: Retrocés aplicat al col·lidir.

- **Mètode:**

OnCollide(): Si la col·lisió és amb el jugador, li aplica el mal i el retrocés.

8.4.34 PlayerProjectile

Controla el mal i col·lisions dels projectils disparats pel jugador.

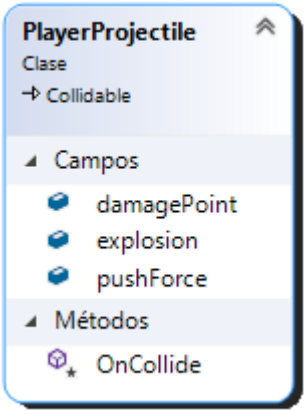


Figura 90 Classe PlayerProjectile

- **Atribut:**

Int damage: mal que farà l’enemic al col·lidir.

Float pushForce: Retrocés aplicat al col·lidir.

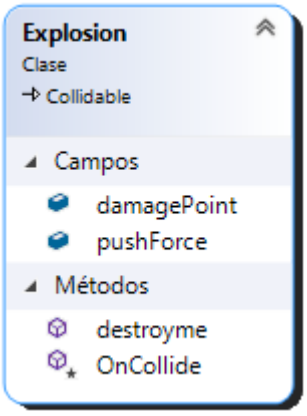
GameObject explosion: Guarda l’objecte explosió si en té.

- **Mètode:**

OnCollide(): Si la col·lisió és amb un enemic, li aplica el mal i el retrocés. En cas de tenir un GameObject explosion l’instanciarà.

8.4.35 Explosion

Controla el mal i col·lisions de les explosions creades per projectils.



- **Atribut:**

Int damage: mal que farà l’enemic al col·lidir.

Float pushForce: Retrocés aplicat al col·lidir.

- **Mètode:**

OnCollide(): Si la col·lisió és amb un enemic, li aplica el mal i el retrocés.

Destroyme(): Destrueix aquest GameObject.

Figura 91 Classe Explosion

8.4.36 ChangeWorld

Crida la funció per canviar d’escena.

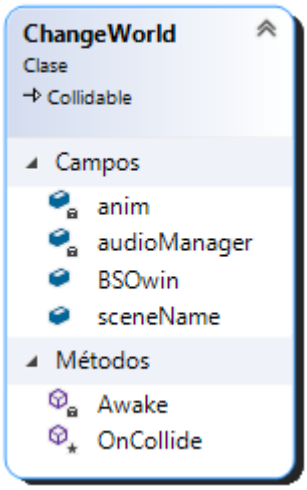


Figura 92 Classe ChangeWorld

- **Atribut:**

Animator anim: Controlador de l’animador.

AudioManager audioManager: Controlador de so.

AudioClip BSOwin: Banda sonora de la pantalla de victòria.

String sceneName: Nom del mapa a carregar.

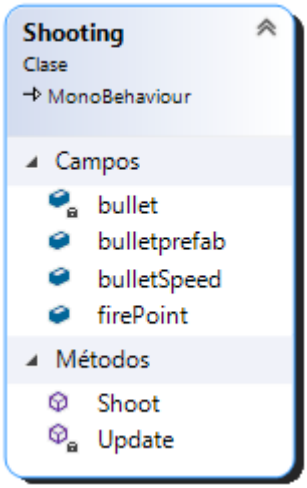
- **Mètode:**

Awake(): Carrega l’animator i l’audiomanager.

OnCollide(): Crida la funció del GameManager que canvia d’escena.

8.4.37 Shooting

Aquesta classe disparará els projectils.



- **Atributs**:

Transform firePoint: Punt d’on es dispara.

List<GameObject> bulletprefab: GameObject a disparar.

float bulletSpeed: Velocitat del projectil.

GameObject bullet: Projectil instanciat.

- **Mètodes**:

Shoot(): Agafa la posició del ratolí i dispara un projectil començant pel firepoint en direcció al ratolí.

Figura 93 Classe Shooting

8.4.38 PlayerShooting

Calcula la direcció on està el ratolí i crida la funció per disparar quan el jugador clica el botó dret del ratolí.

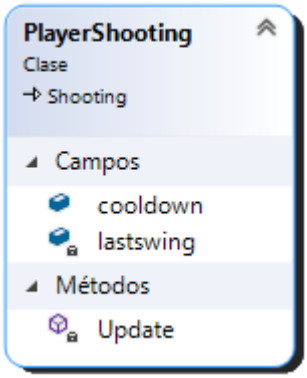


Figura 94 Classe PlayerShooting

- **Atributs**:

Float Cooldown: Interval mínim entre un trets.

Float lastswing: Comptador fins a poder tornar a disparar.

- **Mètodes**:

Update(): Agafa la posició de l'arma i la posa a la variable firePoint de la classe Shooting, i quan detecta que el jugador clica el botó esquerre del mouse, si ha passat més temps des de l'últim dispar del que posa a Cooldown, crida la funció Shoot().

8.4.39 EnemyShooting

Guarda les diferents direccions d’atacs que té l’enemic i crida la classe per disparar amb els que estan activats.

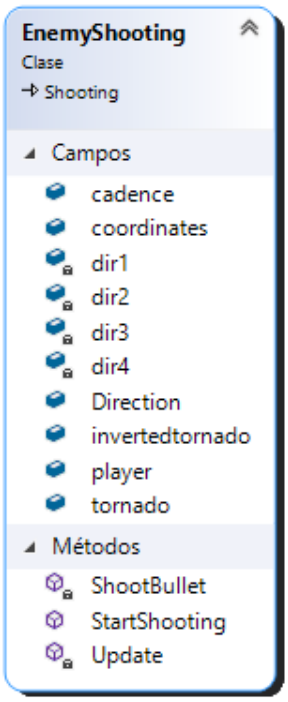


Figura 95 Classe EnemyShooting

- **Atributs:**

Transform Direction: Guarda la posició del jugador.

float cadence: Guarda el temps que hi haurà entre un tret i l’altre.

bool coordinates: Bool que indica si l’enemic dispara a les quatre direccions.

bool invertedtornado: Bool que indica si dispara a les quatre direccions, però va girant seguint en sentit horari.

bool tornado: Bool que indica si dispara a les quatre direccions, però va girant seguint en sentit horari.

bool Player: Bool que indica si l’enemic dispara en direcció al jugador.

Vector2 dir1 , dir2, dir3, dir4: Cada una guarda una de les 4 direccions a les quals dispararà si està seleccionat el bool coordinates, invertedtornado o tornado.

- **Mètodes:**

ShootBullet(): Crida la funció Shoot().

StartShooting(): Activa l’enemic perquè comenci a disparar.

Update(): Si Direction no té el jugador, el busca i li posa.

8.4.40 FairyBossShooting

Guarda les direccions d’atacs que té l’enemic únic i crida la classe per disparar, els dispara aleatòriament.

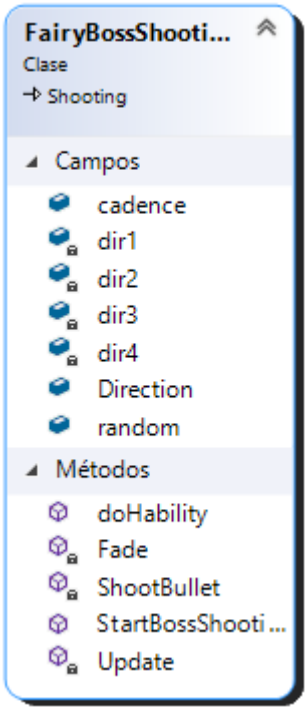


Figura 96 Classe FairyBossShooting

- **Atributs:**

Int random: Contindrà el pròxim atac seleccionat.

- **Mètodes:**

doHability(): Tria el pròxim atac aleatòriament.

StartBossShooting(): Activa l’enemic perquè comenci a disparar.

Fade (): Si Direction no té el jugador, el busca i li posa.

8.4.41 Mantenir

Aquesta classe ens provoca que els GameObject que la porten no s’esborrin al canviar d’escena.

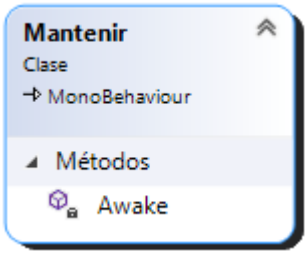


Figura 97 Classe Mantenir

- **Mètodes:**

Awake(): Activa que no es destrueixi el GameObject al canviar d’escena.

9. Implementació i proves

La següent secció se centrarà a explicar amb detall com s'han implementat cada una de les parts del projecte per tal de poder arribar a l’objectiu final.

9.1 Menús

9.1.1 Menú principal i menú de selecció de personatge

Aquests dos menús són els menús que ens torbarem abans de començar una partida. El menú principal consta de dues opcions, que es mostren a la Figura 98.

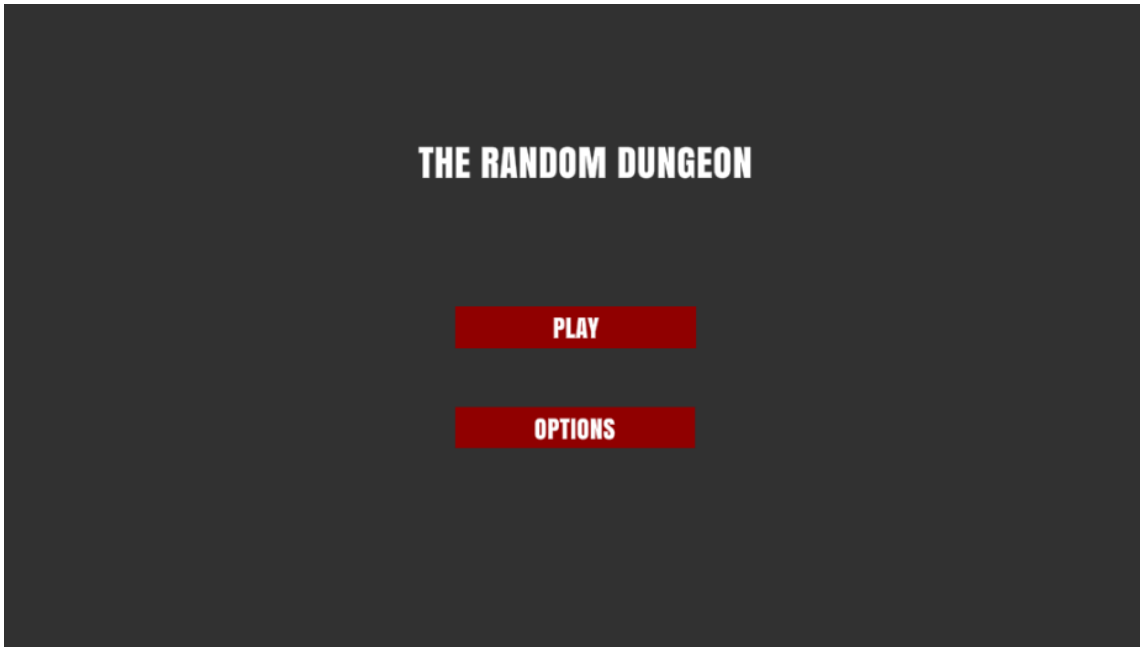


Figura 98 Menú principal

La primera opció en seleccionar-la, activarà una animació que traurà de pantalla aquest menú i obrirà el menú de selecció de personatge. La segona opció ensenyarà el menú de so. Els dos botons funcionen amb un la funció OnClick() que tenim al component button. Veure Figura 99.

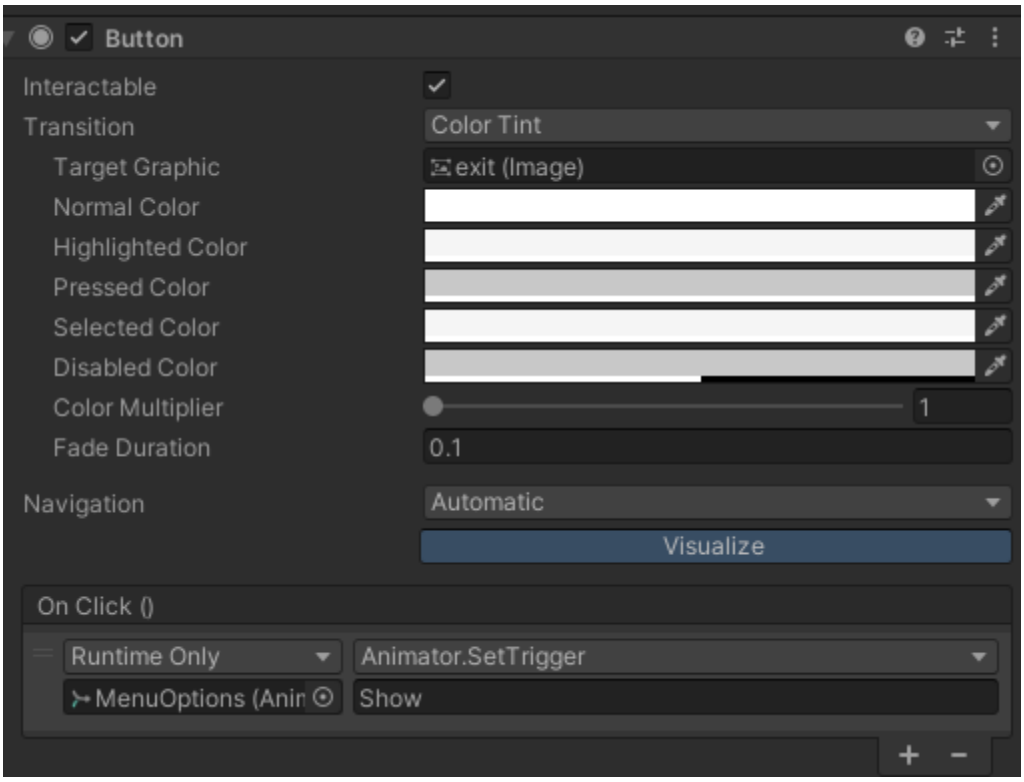


Figura 99 Component Button amb el trigger que mostra el menú d'opcions

El menú de selecció de personatge compta amb un script que canvia el personatge que controlaria el jugador, modificant també les dades que veu per pantalla amb el jugador actualment seleccionat. Tota la informació dels personatges i armes l’agafen del GameManager. Veure Figura 100.

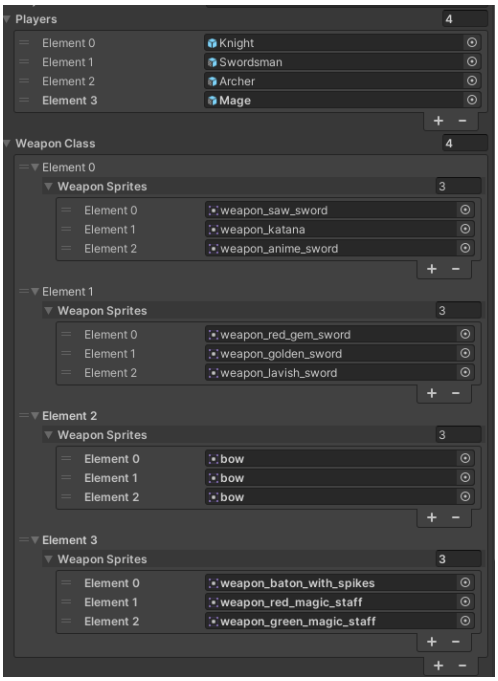


Figura 100 Armes i personatges que es guarden al GameManager

```
public void OnArrowClick(bool right)
{
   if (right)
   {
      currentCharacterSelection++;
      if(currentCharacterSelection == GameManager.instance.players.Count)
      {
         currentCharacterSelection = 0;
      }
      OnSelectionChange();
   }
   else
   {
      currentCharacterSelection--;
      if(currentCharacterSelection < 0)
      {
         currentCharacterSelection = GameManager.instance.players.Count-1;
      }
      OnSelectionChange();
   }
}
public void OnSelectionChange()
{
   characterSelectionSprite.sprite = Manager.instance.players[currentCharacterSelection].GetComponent<Spriteerer>().sprite;    weaponSprite.sprite
= GameManager.instance.weaponClass[currentCharacterSelection].weaponSprites[0];
   hitpointText.text = GameManager.instance.players[currentCharacterSelection].GetComponent<Player>().hitpoint.ToString();
   classText.text = GameManager.instance.players[currentCharacterSelection].GetComponent<Player>().name;
   msText.text = GameManager.instance.players[currentCharacterSelection].GetComponent<Player>().moveSpeed.ToString();
}
```

El botó de play instanciarà el personatge seleccionat i canviarà a l’escena del joc amb la que es començarà la partida. Veure Figura 101.

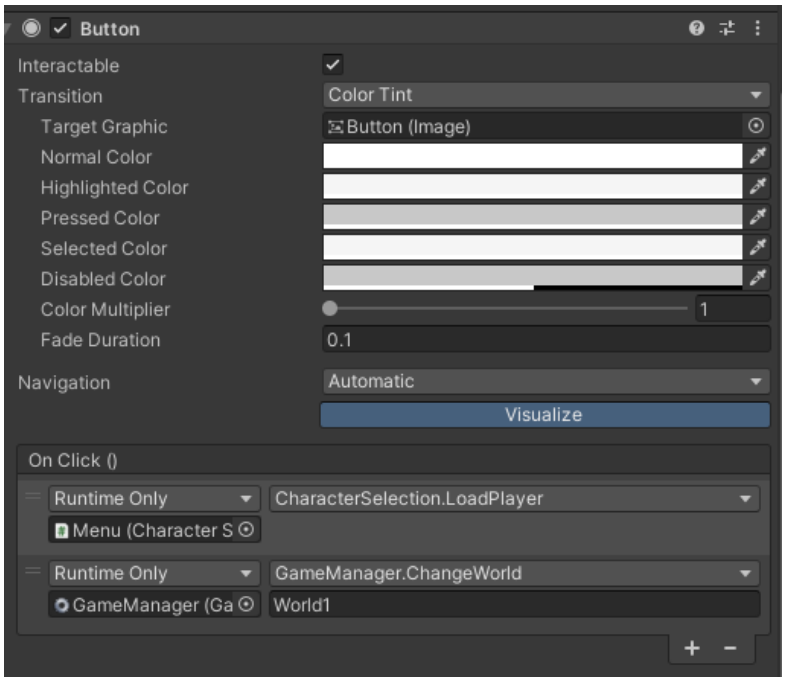


Figura 101 Botó amb funcions OnClick()

```
public void LoadPlayer()
  {
    GameManager.instance.PlayerChoosed = currentCharacterSelection;
    GameObject ply = Instantiate(GameManager.instance.players[currentCharacterSelection], transform.position,
Quaternion.identity);
    ply.name = "Player";
  }
```

Un cop instanciat el personatge, carregarà l’escena del primer món.

```
  public void ChangeWorld(string newScene)
  {
    Scene currentScene = SceneManager.GetActiveScene();
    if(player==null)
    {
      player = GameObject.Find("Player").GetComponent<Player>();
    }
    if(weapon==null)
    {
      weapon = GameObject.Find("Weapon").GetComponent<Weapon>();
    }
    player.transform.position = new Vector3(0,0,0);
    SceneManager.LoadScene(newScene);
  }
```

Aquest menú, durant la partida, estarà escoltant si el jugador polsa la tecla escape per activar l’animació que l’ensenya i pararà el temps i activarà l’animació per mostrar el menú de pausa. Pause i Resume pausen i tornen a la normalitat el temps de l’aplicació per pausar i engegar el joc.

```
  public void PauseTime(){
    Time.timeScale =0;
  }
  public void ResumeTime(){
    Time.timeScale =1;
  }
  void Update()
  {
    if(Input.GetKeyDown(KeyCode.Escape))
    {
      Scene currentScene = SceneManager.GetActiveScene();
      string sceneName = currentScene.name;
      if(currentScene.name == "Win")
      {
        return;
      }
      PauseTime();
      anim.SetTrigger("Show");
    }
  }
```

9.1.3 Menú de final de victòria i derrota

Aquests menús són els menús que indiquen el final de la partida on el jugador retornarà al menú principal. Aquests dos menús funcionen de la mateixa manera, en seleccionar tornar al menú principal el joc s’esborrarà tots els GameObjects que tinguin la classe Mantenir, ja que aquesta classe, evita que s’esborri un GameObject al canviar d’escena, fent així que es faci un reinici complet i no deixar rastre de l’anterior partida. El menú de pausa en seleccionar el botó EXIT funciona de la mateixa forma.

```
public void returnMainMenu(string sceneName)
  {
    foreach (GameObject o in Object.FindObjectsOfType<GameObject>()) {
     if(o.GetComponent<Mantenir>() != null)
     {
       Destroy(o);
     }

    }
    GameManager.instance.experience = 0;
    GameManager.instance.gold = 0;
    GameManager.instance.ChangeWorld(sceneName);
    if(Time.timeScale ==0)
    {
     Time.timeScale = 1;
    }
  }
```

9.2 Dungeon

Al començar la partida el primer que farà el joc és crear la Dungeon aleatòriament. L’escena del primer món compta amb un GameObject anomenat RoomController, el qual tindrà 2 scripts acoblats: El RoomController que s’encarregarà de crear i manipular les habitacions i el DungeonGenerator que crearà totes les posicions on s’hauran de crear les sales.

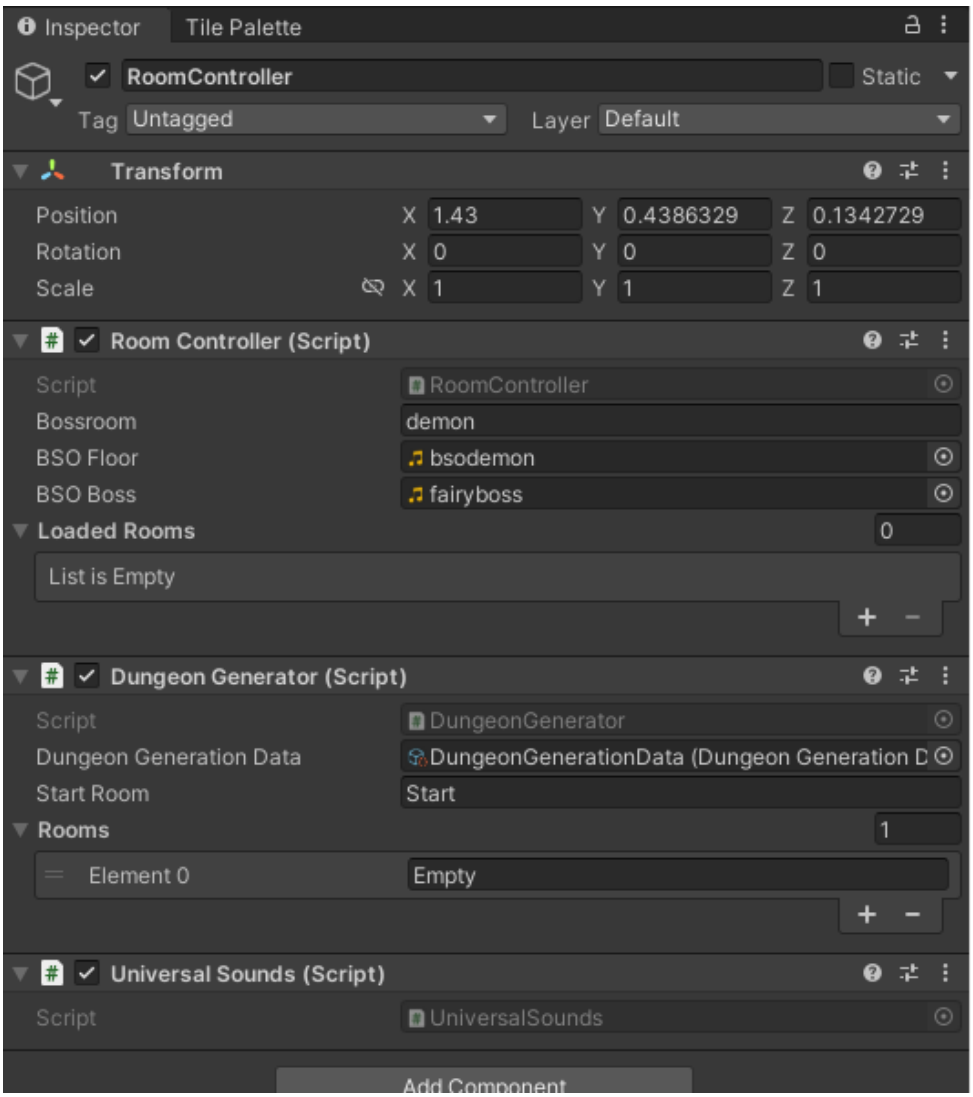


Figura 102 Components del RoomController

En entrar a l’escena el script DungeonGenerator cridarà una funció de la classe DungeonCrawler passant per paràmetre les dades del ScriptableObject DungeonGeneratorData, que indicaran el nombre mínim i màxim d’habitacions i el nombre de buscadors amb els quals generarà una llista de vectors2Int, on cada vector representarà una posició on seguidament es crearà una habitació. En tenir aquest vector cridarà la funció de RoomController per carregar una habitació de la llista d’habitacions disponibles per aquest món, per a cada vector. Veure Figura 103.

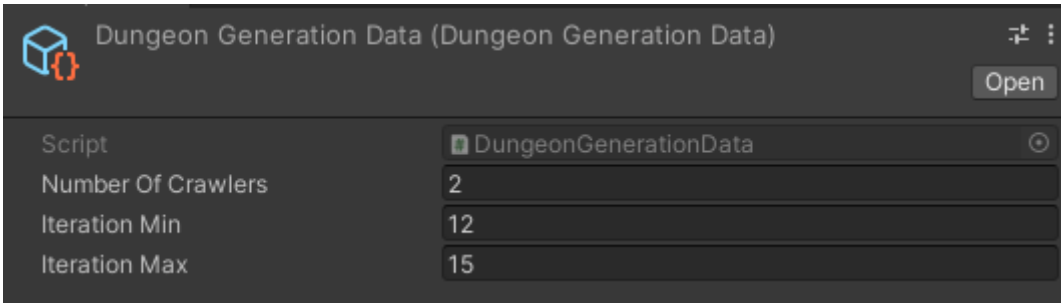


Figura 103 Script amb la classe DungeonGenerator

```
private void Start()
{
   dungeonRooms = DungeonCrawlerController.GenerateDungeon(dungeonGenerationData);
   SpawnRooms(dungeonRooms);
}

private void SpawnRooms(IEnumerable<Vector2Int> rooms)
{
   RoomController.instance.LoadRoom(StartRoom,0, 0);
   foreach(Vector2Int roomLocation in rooms)
   {


      RoomController.instance.LoadRoom(Rooms[Random.Range(0,Rooms.Count)], roomLocation.x, roomLocation.y);

   }
}
```

El DungeonCrawler crearà el nombre de rastrejadors apuntats al dungeonGenerationData amb la posició inicial al (0,0) passat per paràmetre, i anirà cridant la funció move() passant-li un Dictionary per a què es moguin una posició dels quatre costats. En retornar un vector havent-se mogut una posició el DungeonCrawlerController, comprovarà si aquest vector ja l’havia visitat anteriorment algun dels rastrejadors, de no ser el cas, l’afegirà a la llista positionsVisited i un cop aquesta llista tingui una mida entre el mínim i màxim de sales que ha de tenir, el món retornarà aquesta llista al DungeonGenerator.

```
public static List<Vector2Int> GenerateDungeon(DungeonGenerationData dungeonData)
  {
    if(positionsVisited != null)
    {
      positionsVisited.Clear();
    }
    List<DungeonCrawler> dungeonCrawlers = new List<DungeonCrawler>();

    for(int i = 0; i<dungeonData.numberOfCrawlers;i++)
    {
      dungeonCrawlers.Add(new DungeonCrawler(Vector2Int.zero));
    }

    int iterations = Random.Range(dungeonData.iterationMin, dungeonData.iterationMax);

    while(iterations>positionsVisited.Count)
    {
      foreach(DungeonCrawler dungeonCrawler in dungeonCrawlers)
      {
        Vector2Int newPos = dungeonCrawler.Move(directionMovementMap);
        if(!positionsVisited.Contains(newPos) && positionsVisited.Count < iterations)
        {
          positionsVisited.Add(newPos);
        }
      }
    }
    return positionsVisited;
  }
```

```
public Vector2Int Move(Dictionary<Direction, Vector2Int> directionMovementMap)
  {
     Direction toMove = (Direction)Random.Range(0, directionMovementMap.Count);
     Position += directionMovementMap[toMove];
     return Position;
  }
```

En tenir les posicions i haver cridat la funció LoadRoom enviant el nom i la posició el RoomController s’encarregarà de crear cada sala. El RoomController crearà un objecte RoomInfo que guardarà la posició i el nom, i aquest mateix es posarà a la cua.

```
public void LoadRoom(string name, int x, int y)
  {
     if(DoesRoomExist(x, y)){
        return;
     }
     RoomInfo newRoomData = new RoomInfo();
     newRoomData.name = name;
     newRoomData.X = x;
     newRoomData.Y = y;

     loadRoomQueue.Enqueue(newRoomData);
  }
```

La funció Update s’encarregarà de comprovar si encara queden RoomInfo a la cua per carregar. En cas de ser així, en traurà de la cua una i l’enviarà a la funció que crearà l’habitació. Si la cua està buida enviarà a crear l’habitació de l’enemic únic a la posició de l’última sala creada i un cop creada l’habitació de l’enemic únic, agafarà cada habitació creada i mirarà totes les portes de cada habitació si tenen una habitació a l’altra banda. De no ser el cas, traurà la funcionalitat de la porta i no s’obrirà en cap moment. Un cop remogudes les funcionalitats de les portes que no portaven enlloc, l’únic que vigilarà és que, en cas del jugador trobar-se en una sala de l’enemic únic, un cop derrotat tots els enemics de la sala farà aparèixer les escales per anar al següent món.

```
void Update()
  {
     UpdateRoomQueue();
     if(updatedRooms)
     {
        if(currentRoom.enemyCounter()==0 && openRooms ==  false)
        {
           openRoom();
           if(currentRoom.name.Contains("End"))
           {
              GameObject Stairs = GameObject.Find("Stairs");
              Stairs.GetComponent<BoxCollider2D>().enabled = true;
              Stairs.GetComponentInChildren<SpriteRenderer>().enabled = true;
           }
        }
     }
  }
```

```
void UpdateRoomQueue()
  {
    if(isLoadingRoom)
    {
      return;
    }
    if(loadRoomQueue.Count == 0)
    {
      if(!spawnedBossRoom)
      {
        spawnedBossRoom = true;
        StartCoroutine(SpawnBossRoom());
      }
      else if(spawnedBossRoom && !updatedRooms)
      {
        foreach(Room room in loadedRooms)
        {
          room.RemoveUnconnectedDoors();
        }
        updatedRooms = true;
      }
      return;
    }
    currentLoadRoomData = loadRoomQueue.Dequeue();
    isLoadingRoom = true;

    StartCoroutine(LoadRoomRoutine(currentLoadRoomData));
  }
```

Cada habitació és una escena amb un GameObject que conté la classe Room que es carregarà de manera asíncrona que un cop creada afegirà el GameObject creat a la llista d’habitacions de la classe RoomController.

```
IEnumerator LoadRoomRoutine(RoomInfo info)
  {
    string roomName = currentWorldName + info.name;

    AsyncOperation loadRoom = SceneManager.LoadSceneAsync(roomName, LoadSceneMode.Additive);

    while(loadRoom.isDone == false)
    {
      yield return null;
    }
  }
```

Per crear l’habitació de l’enemic únic guarda la posició de l’última sala creada i la canvia per la sala de l’enemic únic.

```
IEnumerator SpawnBossRoom()
  {
    yield return new WaitForSeconds(0.15f);
    if(loadRoomQueue.Count == 0)
    {
      Room bossRoom = loadedRooms[loadedRooms.Count -1];
      Room tempRoom = new Room(bossRoom.X, bossRoom.Y);
      Destroy(bossRoom.gameObject);
      var roomToRemove = loadedRooms.Find(r => r.X == tempRoom.X && r.Y == tempRoom.Y);
      loadedRooms.Remove(roomToRemove);
      LoadRoom(bossroom+"End", tempRoom.X,tempRoom.Y);
    }

      openRoom();
  }
```

Un cop creada l’habitació el GridController s’encarregarà de crear la zona on poden aparèixer els enemics i cridar la funció pertinent per crear els enemics.

```
public void GenerateGrid()
  {
    grid.verticalOffset += room.transform.localPosition.y;
    grid.horitzontalOffset += room.transform.localPosition.x;
    Debug.Log("ofsset grid "+ grid.verticalOffset+grid.horitzontalOffset);
    for(int y = 0; y < grid.rows ; y++)
    {
      for(int x = 0; x < grid.columns ; x++)
      {
        GameObject go = Instantiate(gridTile, transform);
        go.transform.position = new Vector2(x -(grid.columns -grid.horitzontalOffset),y - (grid.rows - grid.verticalOffset));
        go.name = "X: "+ x + "Y: " + y;
        availablePoints.Add(go.transform.position);
      }
    }
    if(GetComponentInParent<Room>().GetComponent<BossSpawn>() == null)
    {
      GetComponentInParent<ObjectRoomSpawner>().InitObjectSpawner();
    }
  }
```

ObjectRoomController Compta amb una llista de ScriptableObject que cada un conté una llista d’enemics i un interval mínim i màxim d’enemics a invocar. Amb aquesta informació agafarà cada objecte i invocarà una quantitat aleatòria dins els intervals indicats dels enemics de la llista a una posició aleatòria.

```
public void InitObjectSpawner()
  {
    foreach(RandomSpawner rs in spawnerData)
    {
      SpawnObjects(rs);
    }
  }
  void SpawnObjects(RandomSpawner data)
  {
    int randomIteration = Random.Range(data.spawnerData.minSpawn, data.spawnerData.maxSpawn + 1);

    for(int i = 0; i < randomIteration; i++)
    {
      int  randomPosition = Random.Range(0, grid.availablePoints.Count - 1);
      GameObject go = Instantiate(data.spawnerData.itemToSpawn[Random.Range(0,
data.spawnerData.itemToSpawn.Count)], grid.availablePoints[randomPosition]+roomCenter, Quaternion.identity, transform) as
GameObject;
      grid.availablePoints.RemoveAt(randomPosition);
    }
  }
```

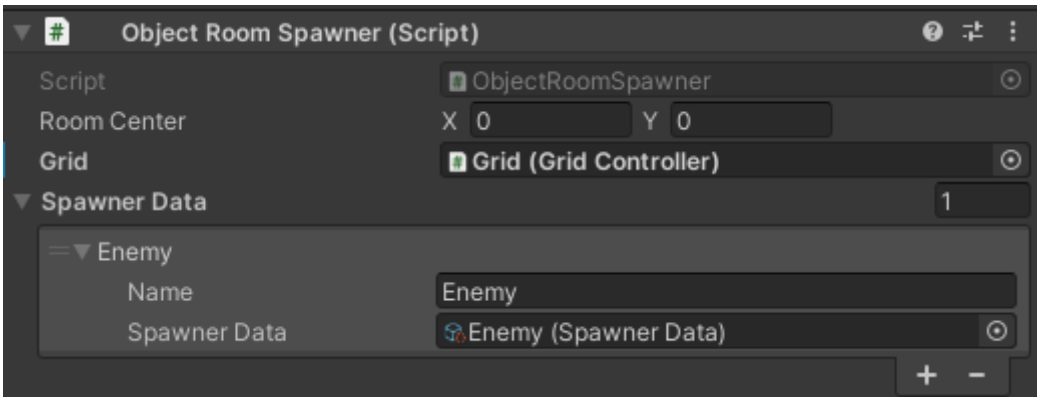


Figura 104 Script que genera els enemics

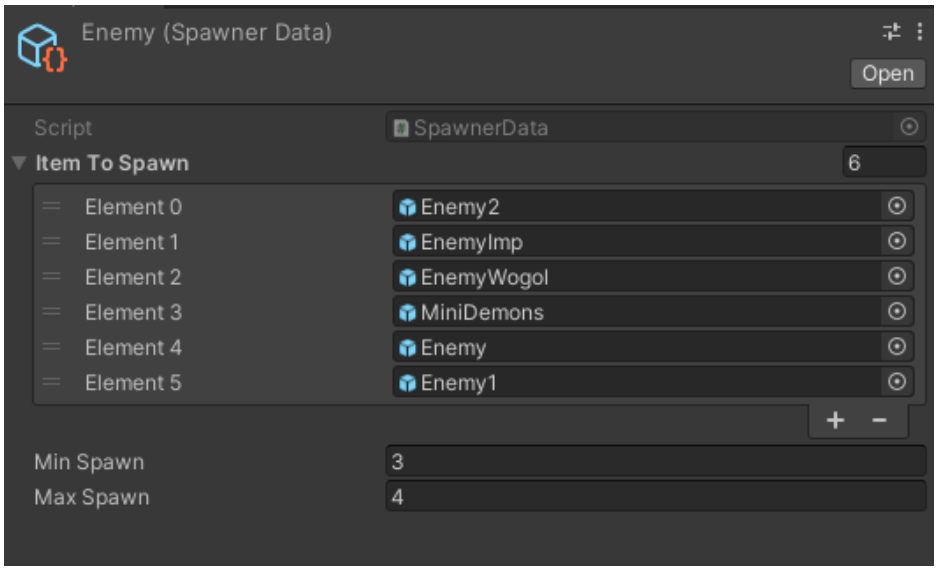


Figura 105 Llista d'enemics i interval d'aparició

9.3 Lluitadors

Tant el controlador dels enemics com el del personatge controlat pel jugador hereten d’una classe en comú, La classe Fighter. Aquesta classe conté les estadístiques de vida i s’encarrega de baixar la vida i rebre el retrocés del lluitador que rebi qualsevol classe d’atac.

```
protected virtual void ReceiveDamage(Damage dmg){
    if (Time.time - lastImmune > immuneTime)
    {
        lastImmune = Time.time;
        hitpoint -= dmg.damageAmount;
        pushDirection = (transform.position - dmg.origin).normalized * dmg.pushForce;
        audioManager.PlaySFX(audioManager.hit);

        GameManager.instance.ShowText(dmg.damageAmount.ToString()+" dmg", 15, Color.red, transform.position, Vector3.up *
100, 0.5f);

        if (hitpoint <= 0)
        {
            hitpoint = 0;
            Death();
        }
    }
}
```

Els lluitadors que comptin amb atacs a distància portaran un script d’una classe que hereta de la classe Shooting. Aquesta classe dispara un projectil a la direcció que li passen per paràmetre.

```
public void Shoot(Vector3 Direction)
{
    this.GetComponent<Rigidbody2D>().velocity = Vector3.zero;
    Direction.Normalize();
    Vector3 pos = Camera.main.WorldToScreenPoint(transform.position);
    Vector3 dir = Input.mousePosition - pos;
    float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
    if(GameManager.instance.weapon.weaponLevel>bulletprefab.Count-1)
    {
        bullet = Instantiate(bulletprefab[bulletprefab.Count-1], firePoint.position, Quaternion.AngleAxis(angle,
Vector3.forward));
    }else{
        bullet = Instantiate(bulletprefab[GameManager.instance.weapon.weaponLevel], firePoint.position,
Quaternion.AngleAxis(angle, Vector3.forward));
    }
    Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
    rb.AddForce(Direction * bulletSpeed, ForceMode2D.Impulse);
}
```

9.3.1 Player

El personatge que controla el jugador consta de dues parts importants, el propi cos del jugador que contindrà la classe que hereta de fighter i a part li donarà el moviment al i la seva arma, que portarà el script Weapon i s’encarrega de fer l’animació i les conseqüències produïdes d’un atac. El jugador es mourà amb les tecles W,A,S,D, que la classe Player agafarà i en traurà la direcció que té i calcularà amb la direcció seleccionada i la velocitat de moviment del jugador, el moviment que vol fer el personatge, que aplicarà una força al Rigidbody del personatge sumant la força que pugui tenir de retrocés en cas d’haver rebut un atac.

```
void FixedUpdate()
  {
     Vector2 move = rb.position + movement * actualMoveSpeed * Time.fixedDeltaTime;
     if(Mathf.Abs(pushDirection.x) + Mathf.Abs(pushDirection.y) > 0.3 ){
        move += (pushDirection-movement) * actualMoveSpeed * Time.fixedDeltaTime;
     }

     pushDirection = Vector2.Lerp(pushDirection, Vector2.zero, pushRecoverySpeed);
     rb.MovePosition(move);
  }
```

Per executar una envestida el jugador canviarà la velocitat actual durant un petit període de temps per una velocitat molt més alta. A l’acabar el temps tornarà a la velocitat normal i després començarà el compte enrere fins a poder tornar a realitzar-ne un altre.

```
  void Update()
  {
     movement.x = Input.GetAxisRaw("Horizontal");
     movement.y = Input.GetAxisRaw("Vertical");
     animator.SetFloat("Horizontal", movement.x);
     animator.SetFloat("Vertical", movement.y);
     animator.SetFloat("Speed", movement.sqrMagnitude);

     if(Input.GetKeyDown(KeyCode.Space))
     {
        if(dashCoolCounter<= 0 && dashCounter<= 0)
        {
           actualMoveSpeed = dashSpeed;
           dashCounter = dashLength;
        }
     }
     if(dashCounter>0)
     {
        dashCounter -= Time.deltaTime;
        if(dashCounter<=0)
        {
           actualMoveSpeed = moveSpeed;
           dashCoolCounter = dashCooldown;
        }
     }
     if(dashCoolCounter > 0)
     {
        dashCoolCounter -= Time.deltaTime;
     }
  }
```

En morir, es mostrarà la pantalla de mort i es pararà el temps de la partida.

```
  protected override void Death(){
    GameObject A = GameObject.Find ("death");
    animator = A.GetComponent<Animator>();
    animator.SetTrigger("Show");
    Time.timeScale = 0;
  }
```

L’atac del personatge l’executa completament la seva arma. L’arma anirà canviant de cantó encarant sempre cap on es dirigeix el jugador. Quan el jugador cliqui el botó esquerre del ratolí l’arma activarà l’animació d’atac.

```
protected override void Update()
{
   base.Update();
   if(Input.GetKeyDown(KeyCode.Mouse0)){
      if(Time.time  - lastswing > cooldown){
         lastswing = Time.time;
         Swing();
      }
   }
   if(playerAnim.GetCurrentAnimatorClipInfo(0)[0].clip.name.Contains("mirrored"))
   {
      anim.SetFloat("Direction", -1f);
   }else{
      anim.SetFloat("Direction", 1f);
   }
}
```

```
private void Swing(){
   anim.SetTrigger("Swing");
   audioManager.PlaySFX(audioManager.swing);
}
```

En cas que l’arma en atacar trobí amb un enemic, li enviarà el mal i retrocés utilitzant la funció OnCollide(), heretada de la classe Collidable.

```
protected override void OnCollide(Collider2D coll){

   if( coll.tag == "Fighter" && coll.name != "Player"){
      Damage dmg = new Damage
      {
         damageAmount = damagePoint[weaponLevel],
         origin = transform.position,
         pushForce = pushForce[weaponLevel]
      };
      coll.SendMessage("ReceiveDamage", dmg);
   }
}
```

Els personatges que disparen a distància al clicar el botó esquerre del ratolí cridaran la funció Shoot() de la classe PlayerShooting, enviant la direcció que es troba el ratolí respecte al personatge.

```
void Update()
   {
      if(firePoint==null && GameObject.Find("Weapon")!=null)
      {
         firePoint = GameObject.Find("Weapon").GetComponent<Transform>();
      }
      if(Input.GetKeyDown(KeyCode.Mouse0)){
         if(Time.time  - lastswing > cooldown){
            lastswing = Time.time;
            Vector3 screenPoint = Camera.main.WorldToScreenPoint(transform.position);
            Vector3 Direction = (Vector3)(Input.mousePosition-screenPoint);
            Shoot(Direction);
         }
      }
   }
```

Els projectils disparats pel personatge portaran la classe playerprojectile que portarà la funció OnCollide heretada i, en cas de tenir-ne una classe Explosion que en col·lidir, el projectil crearà una explosió.

```
protected override void OnCollide(Collider2D coll){

    if( coll.tag == "Fighter" && coll.name != "Player"){
        Damage dmg = new Damage
        {
            damageAmount = damagePoint,
            origin = transform.position,
            pushForce = pushForce
        };
        coll.SendMessage("ReceiveDamage", dmg);

    }
    if(coll.name != "Player" && coll.name != "Weapon")
    {
        //posar explosio
            if(explosion != null)
        {
            Instantiate(explosion, transform.position, Quaternion.identity);
            audioManager.PlaySFX(audioManager.explode);

        }
            Destroy(gameObject);
    }
}
```

```
protected override void OnCollide(Collider2D coll){
    if( coll.tag == "Fighter" && coll.name!= "Player"){
        Damage dmg = new Damage
        {
            damageAmount = damagePoint,
            origin = transform.position,
            pushForce = pushForce
        };
        coll.SendMessage("ReceiveDamage", dmg);

    }
}
```

9.3.2 Enemics

Els enemics utilitzen l’algorisme de cerca de camins pathfinding A* que calcularà el camí més ràpid per arribar al jugador, s’anirà actualitzant el camí cada poc temps. Aquest algorisme començarà a calcular un cop el jugador entri a l’habitació.

```
public void StartMoving()
{
   InvokeRepeating("UpdatePath", 0f, .5f);
}
 void UpdatePath(){
   if(seeker.IsDone()){
      seeker.StartPath (rb.position, playerTransform.position, OnpathComplete);
   }
}
void OnpathComplete(Path p)
{
   if(!p.error){
      path = p;
      currentWaypoint = 1;
   }
}
```

Un cop el camí calculat es quedarà guardat a la variable path. Pel moviment de l’enemic, primer es controla si la distància entre el jugador i l’enemic és més gran o més petita que la variable DistanceFromPlayer, de ser el cas la direcció calculada s’invertirà per permetre a l’enemic, la direcció es calcula agafant el pròxim punt del camí respecte del rigidbidy i després normalitzant-la. Un cop està la direcció calculada, es calcularà la velocitat amb la qual viatjarà, aquesta força es calcula amb la direcció, la velocitat i la força d’un possible retrocés per un atac. Un cop calculat s’utilitza la funció addforce() per donar una força al cos de l’enemic.

```
void FixedUpdate()
{
    if(path==null){
        return;
    }
    if(currentWaypoint >= path.vectorPath.Count){
        reachedEndofPath = true;
        return;
    }else{
        reachedEndofPath = false;
    }
    if(Vector2.Distance((Vector2)playerTransform.position,rb.position)<DistanceFromPlayer){
        direction = -((Vector2)path.vectorPath[currentWaypoint]-rb.position).normalized;

    }else{
        if(((Vector2)path.vectorPath[currentWaypoint]-rb.position)!=Vector2.zero)
        {
            if(((Vector2)path.vectorPath[currentWaypoint]-rb.position)!=Vector2.zero)
            {
                direction = ((Vector2)path.vectorPath[currentWaypoint]-rb.position).normalized;
            }
        }
    }
    Vector2 force = direction * speed * Time.deltaTime;
    if(Mathf.Abs(pushDirection.x) + Mathf.Abs(pushDirection.y) > 0.3 ){
        force += (pushDirection-direction) * speed * Time.fixedDeltaTime;

    }
    if(pushRecoverySpeed<0)
    {
    }
    pushDirection = Vector2.Lerp(pushDirection, Vector2.zero, pushRecoverySpeed);
    rb.AddForce(force);
    float distance = Vector2.Distance(rb.position, path.vectorPath[currentWaypoint]);

    if(distance < nextWaypointDistance){
        currentWaypoint++;
    }
}
```

El combat dels enemics en el cas del cos a cos el controla la classe EnemyHitbox que té el mal i força de retrocés i la funció heretada OnCollide().

```
protected override void OnCollide(Collider2D coll)
    {
        if ( coll.tag == "Fighter" && coll.name =="Player")
        {
            Damage dmg = new Damage
            {
                damageAmount = damage,
                origin = transform.position,
                pushForce = pushForce
            };

            coll.SendMessage("ReceiveDamage", dmg);
        }
    }
```

En el cas dels enemics amb atacs a distància, a part del component anterior també comptaran amb la classe EnemyShooting, aquesta classe dispararà el projectil que tingui posat amb la direcció i cadència establerta. Aquests enemics començaran a disparar un cop el jugador entri a l’habitació.

```
public void StartShooting()
  {
     InvokeRepeating("ShootBullet", 2.0f, cadence);
  }
  void Update()
  {
     if(Direction==null && GameObject.Find("Player")!=null)
     {
        Direction = GameObject.Find("Player").GetComponent<Transform>();
     }
  }

  void ShootBullet(){
     if(invertedtornado)
     {
        Shoot(dir1);
        dir1 = Quaternion.AngleAxis(-20f, Vector3.forward) * dir1;
        Shoot(dir2);
        dir2 = Quaternion.AngleAxis(-20f, Vector3.forward) * dir2;
        Shoot(dir3);
        dir3 = Quaternion.AngleAxis(-20f, Vector3.forward) * dir3;
        Shoot(dir4);
        dir4 = Quaternion.AngleAxis(-20f, Vector3.forward) * dir4;
     }
     if(tornado)
     {
        Shoot(dir1);
        dir1 = Quaternion.AngleAxis(20f, Vector3.forward) * dir1;
        Shoot(dir2);
        dir2 = Quaternion.AngleAxis(20f, Vector3.forward) * dir2;
        Shoot(dir3);
        dir3 = Quaternion.AngleAxis(20f, Vector3.forward) * dir3;
        Shoot(dir4);
        dir4 = Quaternion.AngleAxis(20f, Vector3.forward) * dir4;
     }
     if(coordinates){
        Shoot(firePoint.up);
        Shoot(firePoint.right);
        Shoot(-firePoint.up);
        Shoot(-firePoint.right);
     }
     if(player){
        Shoot(Direction.position-firePoint.position);
     }
  }
```

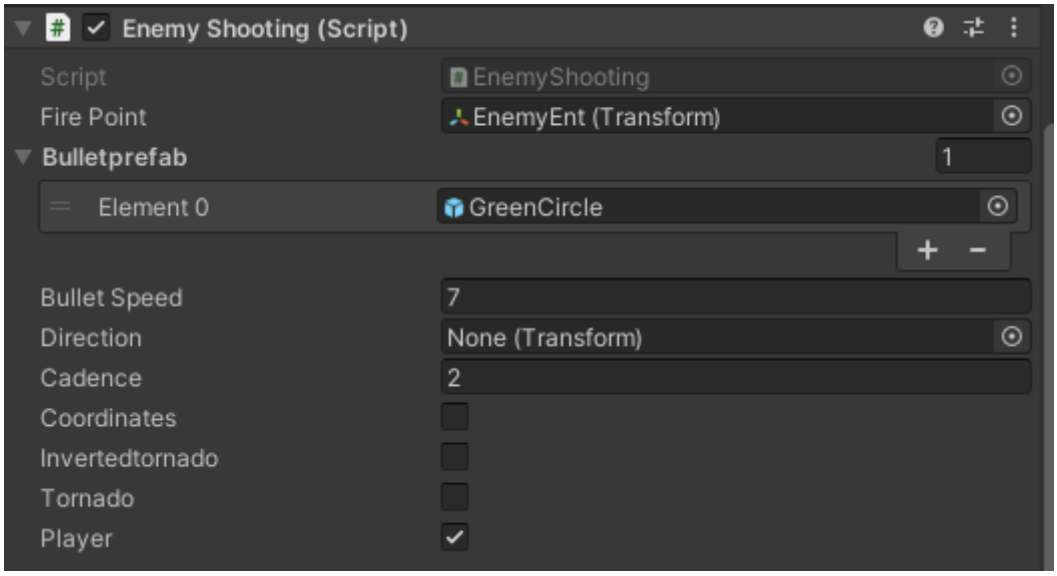


Figura 106 Script que utilitzen els enemics per disparar

En el cas de l’enemic únic del segon món que dispara projectils, la classe no té una elecció que disparar, en el seu cas, cada triarà una de les opcions a l’atzar i dispararà una ràfega i al cap d’uns segons començarà una altra ràfega a una direcció triada aleatòriament.

```
public void doHability()
  {
    random = Random.Range(0,4);
    StartCoroutine("Fade");
  }
  IEnumerator Fade()
  {
    Debug.Log("random: "+random);
    InvokeRepeating("ShootBullet", 2.0f, 0.5f);
    yield return new WaitForSeconds(5f);
    CancelInvoke("ShootBullet");
  }
```

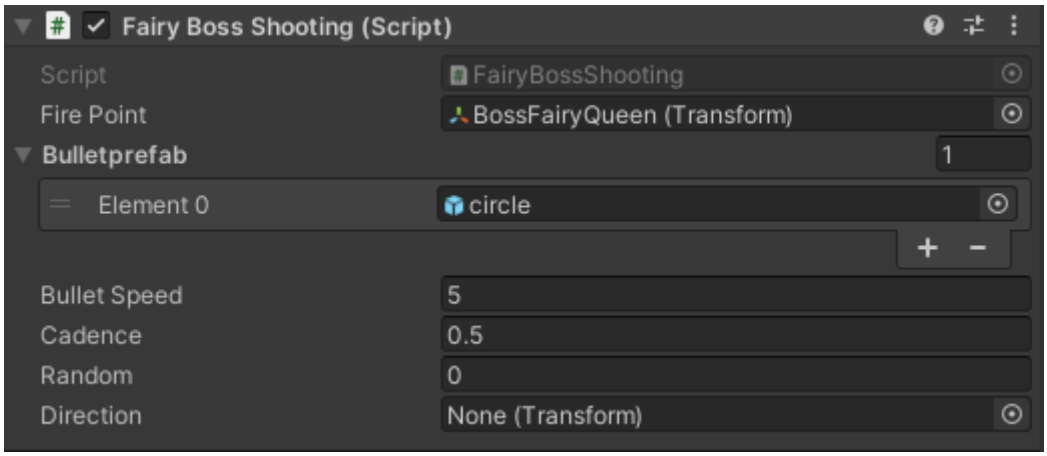


Figura 107 Script que utilitza l'enemic únic del segon món per disparar

Els projectils enemics, igual que els projectils del jugador, tenen una classe heretada de Collidable on utilitzen la funció OnCollide.

```
protected override void OnCollide(Collider2D coll){
    if( coll.tag == "Fighter" && coll.name == "Player"){
      Damage dmg = new Damage
      {
        damageAmount = damagePoint,
        origin = transform.position,
        pushForce = pushForce
      };
      coll.SendMessage("ReceiveDamage", dmg);

    }
    if(coll.tag != "Fighter" || coll.tag == "Fighter" && coll.name == "Player")
    {
      if(coll.tag != "Bullet"){
        Destroy(gameObject);
      }
    }
  }
```

Amb la funció Death heretada de Fighter en morir, l’enemic donarà l’experiència i or al jugador.

```
protected override void Death(){
    Destroy(gameObject);
    GameManager.instance.GrantXp(xpValue);
    if(xpValue>0)
    {
      GameManager.instance.ShowText("+"+xpValue+" exp",30,  Color.cyan,transform.position , Vector3.up*40, 1.0f );
    }
    GameManager.instance.gold += gold;
    if(gold>0)
    {
      GameManager.instance.ShowText("+"+gold+" gold",30,   Color.yellow, transform.position ,Vector3.up*20, 1.0f );
    }
  }
```

9.4 Partida

En una partida, el GameManager s’encarrega de guardar la informació que només forma part d’aquesta, com és l’or i experiència del personatge, igual que porta les funcions per fer pujar l’experiència i nivell del personatge o nivell de l’arma.

```
public bool tryUpgradeWeapon()
{
    if(weaponPrices.Count <= weapon.weaponLevel)
    {
        return false;
    }
    if(gold >= weaponPrices[weapon.weaponLevel])
    {
        gold -= weaponPrices[weapon.weaponLevel];
        weapon.UpgradeWeapon();
        return true;
    }
    return false;
}
private void Update()
{
    if(floatingTextManager==null)
    {
        floatingTextManager = GameObject.Find("FloatingTextManager").GetComponent<FloatingTextManager>();
    }
    if(player==null && GameObject.Find("Player")!=null)
    {
        player = GameObject.Find("Player").GetComponent<Player>();
    }
    if(weapon==null && GameObject.Find("Weapon") != null)
    {
        weapon = GameObject.Find("Weapon").GetComponent<Weapon>();
    }
}
public int GetCurrentLevel()
{
    int r = 0;
    int add = 0;

    while ( experience >= add){
        add += xpTable[r];
        r++;
        if (r == xpTable.Count)
        {
            return r;
        }
    }
    return r;
}
public int GetXpToLevel(int lvl)
{
    int r=0;
    int xp=0;
    while(r<lvl)
    {
        xp +=xpTable[r];
        r++;
    }
    return xp;
}
public void GrantXp(int xp)
{
    int currentLvl = GetCurrentLevel();
    experience+= xp;
    if (experience >= GetXpToLevel(currentLvl)&&currentLvl<3)
    {
        LvlUp();
    }
}
public void LvlUp()
{
    player.maxHitpoint= System.Convert.ToInt32((player.maxHitpoint*GetCurrentLevel())/1.5);
    player.hitpoint = player.maxHitpoint;
    ShowText(" LvlUp", 15, Color.yellow, transform.position, Vector3.up * 100, 0.5f);
}
```

També s’encarrega de funcions generals de canviar d’escena.

```
public void ChangeWorld(string newScene)
  {
    Scene currentScene = SceneManager.GetActiveScene();
    if(player==null)
    {
      player = GameObject.Find("Player").GetComponent<Player>();
    }
    if(weapon==null)
    {
      weapon = GameObject.Find("Weapon").GetComponent<Weapon>();
    }
    player.transform.position = new Vector3(0,0,0);
    SceneManager.LoadScene(newScene);
  }
```

O cridar la funció per fer aparèixer texts per pantalla.

```
public void ShowText(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration)
  {

    floatingTextManager.Show(msg,fontSize,color,position,motion,duration);
  }
```

Els missatges que surtin per pantalla són objectes FloatingText amb les característiques passades per paràmetre. Aquest missatge apareixerà a la posició passada per paràmetre i pujarà fins a uns segons després desaparèixer.

```
private void Update()
  {
    foreach (FloatingText txt in floatingTexts)
      txt.UpdateFloatingText();
  }
  public void Show(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration)
  {
    FloatingText dataText = GetFloatingText();

    dataText.txt.text = msg;
    dataText.txt.fontSize = fontSize;
    dataText.txt.color = color;
    dataText.go.transform.position = Camera.main.WorldToScreenPoint(position);
    dataText.motion = motion;
    dataText.duration = duration;

    dataText.Show();
  }
```

Per mostrar i desaparèixer el text activarem o desactivarem l’objecte que el porta, i la funció UploadFloatingText() anirà modificant la posició del text i, un cop passat el temps màxim que hauria d’estar el text a la pantalla, cridar la funció Hide().

```
public void Show()
 {
    active = true;
    lastShow = Time.time;
    go.SetActive(active);
 }
 public void Hide()
 {
    active = false;
    go.SetActive(active);
 }
 public void UpdateFloatingText()
 {
    if(!active)
    {
       return;
    }

    if(Time.time - lastShow > duration)
    {
       Hide();
    }
    go.transform.position += motion * Time.deltaTime;
 }
```

La càmera estarà fixa al centre de l'habitació i anirà posant la posició contínuament perquè quan el jugador canviï d'habitació la càmera també ho farà. La zona on efectuarà l'algorisme de cerca A* estarà fixat a la càmera i en detectar que canvia de posició la càmera també canviarà de posició.

```
void Update()
  {
    UpdatePosition();
  }
  void UpdatePosition()
  {
    if(currentRoom == null)
    {
       return;
    }

    Vector3 targetPos = GetCameraTargetPosition();

    transform.position = Vector3.MoveTowards(transform.position, targetPos, Time.deltaTime *
moveSpeedWhenRoomChange);
  }
  public Vector3 GetCameraTargetPosition()
  {
    if(currentRoom == null)
    {
       return Vector3.zero;
    }

    Vector3 targetPos = currentRoom.GetRoomCenter();
    targetPos.z = transform.position.z;

    return targetPos;
  }
  public bool IsSwitchingScene()
  {
    return transform.position.Equals(GetCameraTargetPosition()) == false;
  }
```

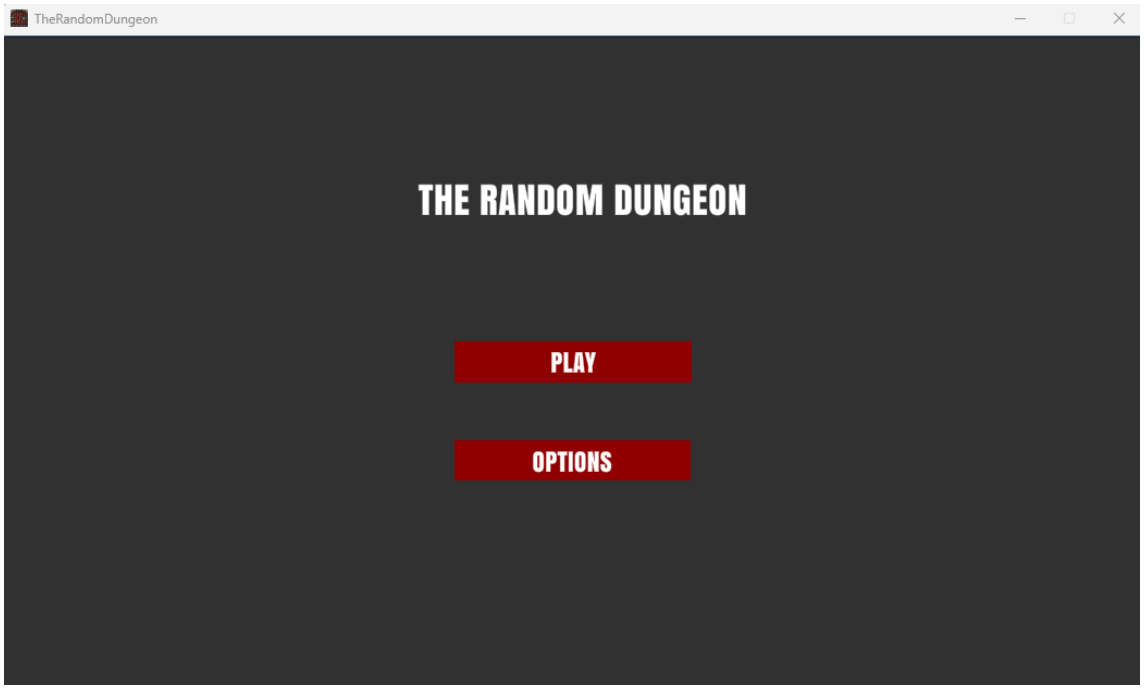
Al començar el primer món la classe ChangeLocationPath() s’encarregarà de crear el grid on Astar treballarà, i anirà canviant la seva posició seguint la posició de la càmera.

```
void Start()
  {
    GameObject A = GameObject.Find ("Astar");
    astar = A.GetComponent("AstarPath") as AstarPath;
    data = AstarPath.active.astarData;
    gg = data.AddGraph(typeof(Pathfinding.GridGraph)) as Pathfinding.GridGraph;

    gg.is2D = true;
    gg.width = 25;
    gg.depth = 12;
    gg.nodeSize = 1f;
    gg.collision.use2D = true;
    gg.center=camera.GetCameraTargetPosition();
    gg.UpdateSizeFromWidthDepth();
    AstarPath.active.Scan();
  }
  private void Update()
  {
    if(camera.IsSwitchingScene())
    {
      ChangeGridLocation();
    }
  }
  private void ChangeGridLocation()
  {
    gg.center = camera.GetCameraTargetPosition();
    AstarPath.active.Scan();
  }
```

10. Implantació i resultats

A continuació es mostren diverses captures de pantalla del videojoc:



Menú principal on el jugador podrà modificar la música o anar a selecció de personatge

Figura 108 Pantalla principal

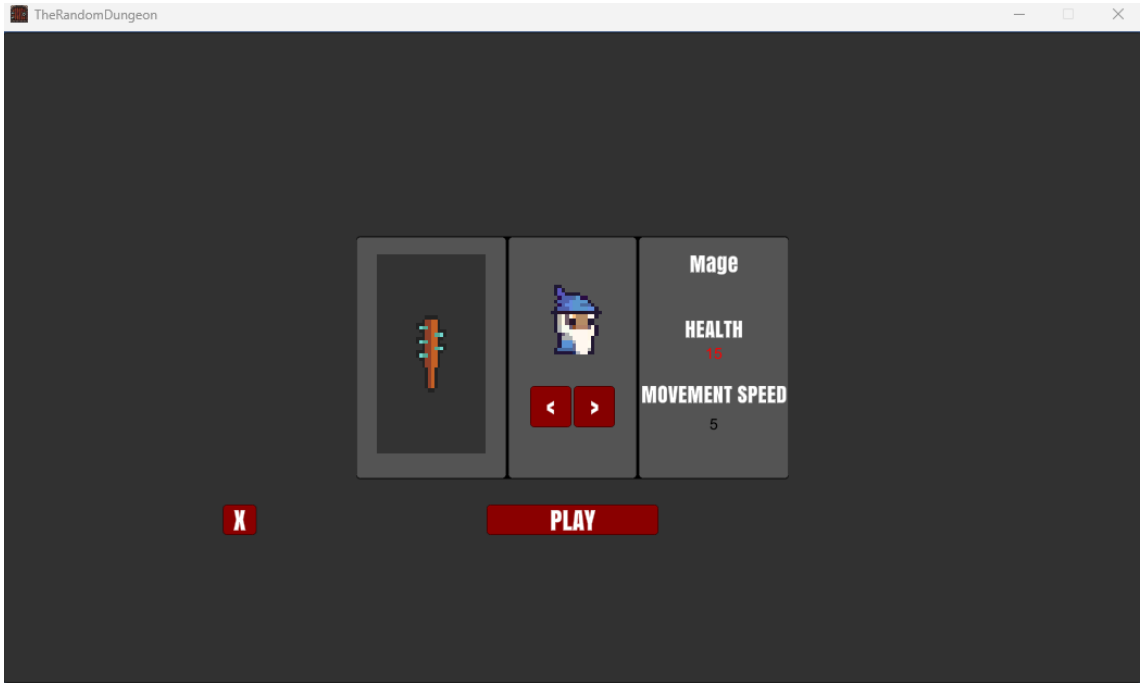


Figura 109 Pantalla de selecció de personatge

Menú on el jugador podrà veure els diferents personatges, les seves estadístiques i triar amb quin vol començar la partida.

Figura 110 Combat dins d'una sala al primer món

Sala on el jugador està en combat amb enemics del primer món

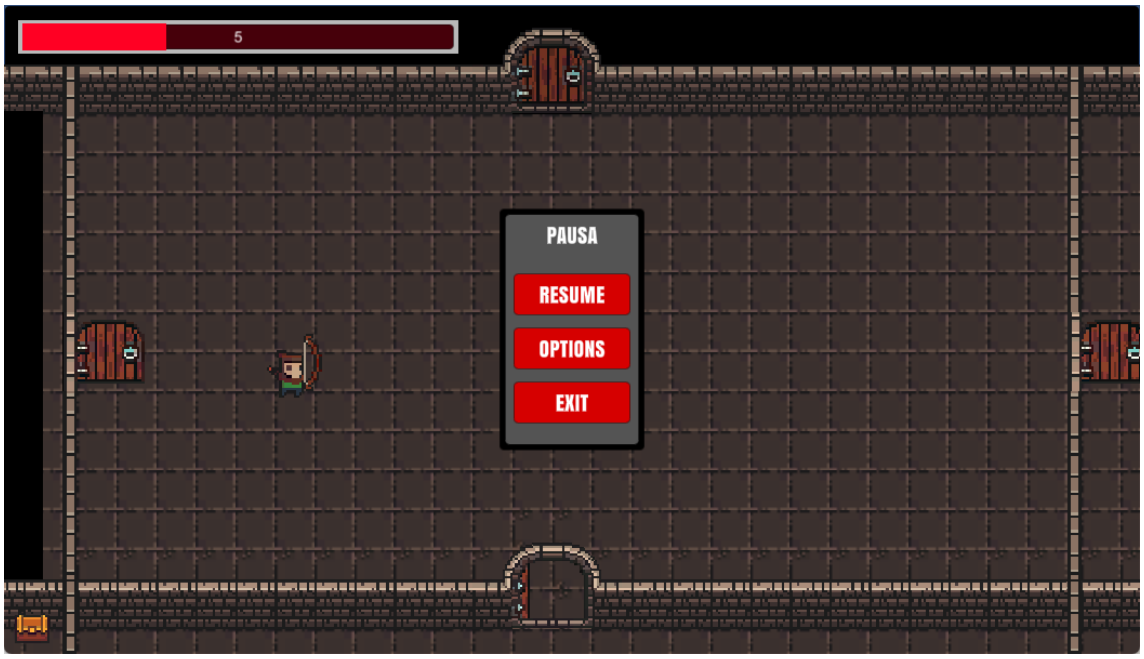


Figura 111 Pantalla de pausa

Menú de pausa que pot obrir el jugador durant la partida

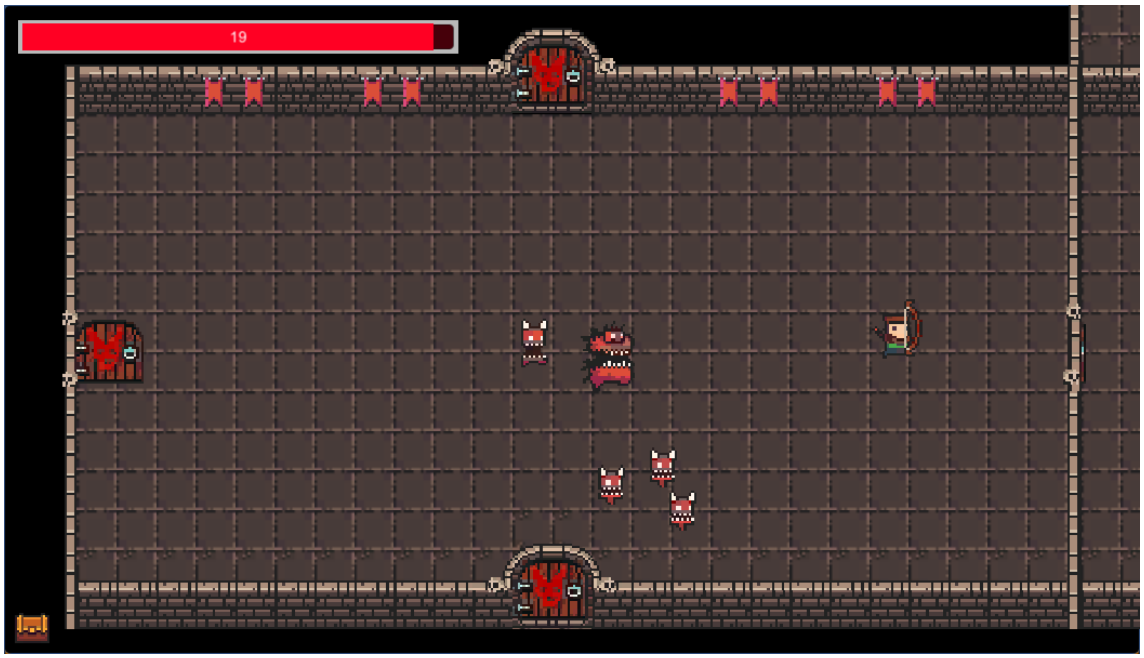


Figura 112 Enemic únic del primer món

Enemic únic del primer món amb varis súbdits invocats.

Figura 113 Combat dins d'una sala al segon món

Sala on el jugador està en combat amb enemics del segon món

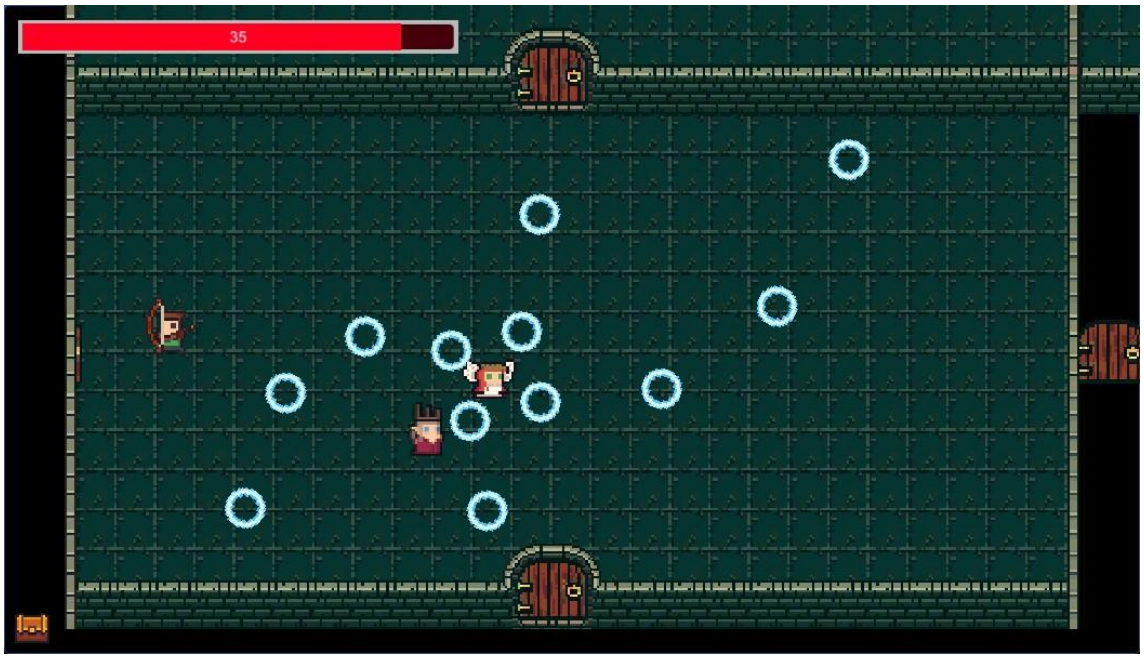


Figura 114 Enemic únic del segon món

Enemic únic del segon món amb la reina executant una ràfega de dispars.

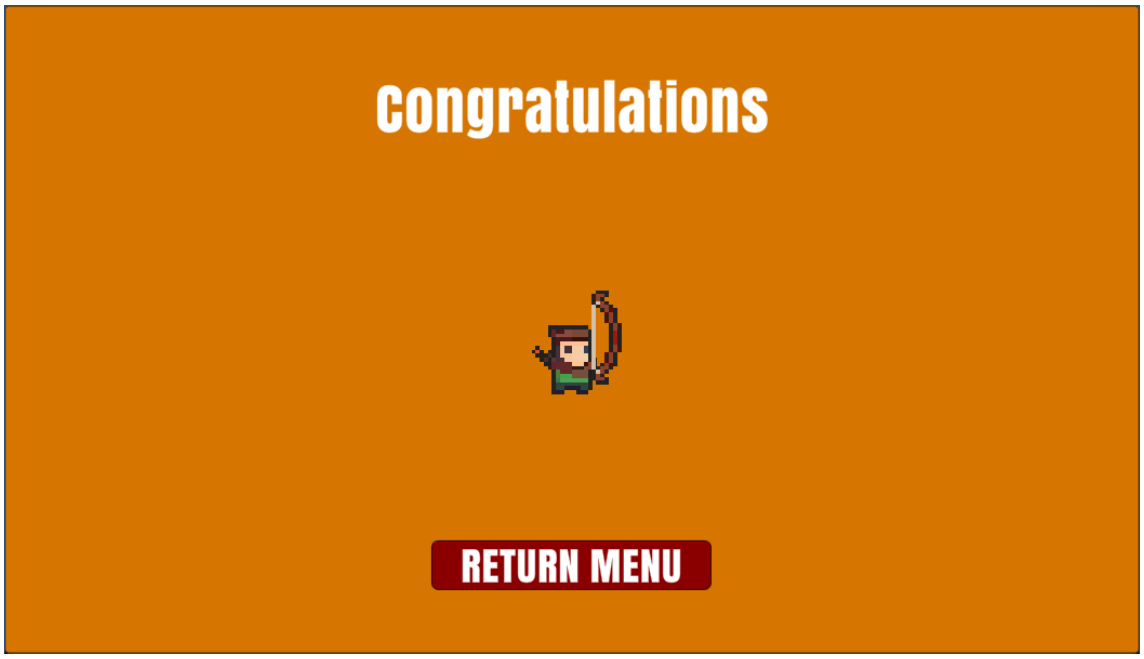


Figura 115 Pantalla de victòria

Pantalla de victòria mostrada al guanyar l’enemic únic del segon món

11. Conclusions

En aquest projecte hem desenvolupat un videojoc en 2 dimensions on el jugador haurà d’escapar d’una masmorra on, tant els enemics que et trobaràs com la forma de la masmorra seran elegits de forma aleatòria. El sistema de combat està compta amb atacs a distància i atacs cos a cos, i enemics únics amb mecàniques personalitzades per a cada un.

El desenvolupament d'aquest projecte a set el projecte més gran al que he realitzat a part de ser el primer cop que m'he enfrontat al desenvolupament d'un videojoc i un munt de coses que hi comporta. D'aquest projecte m'emporto una sèrie de coses bones i dolentes que m'ajudaran a millorar per a un projecte futur.

En el cas de les coses bones una de les coses més importants que he après és a planificar un projecte de grans magnituds, saber veure les diferents parts que comporta i saber fer una planificació de temps adequada per cada cosa per no anar malament. Una altra cosa molt rellevant que he après és la importància d'escriure un codi ben estructurat i separat correctament, així en expandir-se el codi no s'haurà d'anar modificant el codi anterior.

En el cas de les coses dolentes la més destacat a set el no haver pogut executar animacions del tot correctament, perquè tot i que el joc té animacions, en el cas dels atacs cos a cos han quedat molt ortopèdics.

12. Treball futur

Aquest projecte ha estat pensat i creat amb ment de poder futurament implementar algunes millores. Aquestes millores juntament amb millores o remodelacions d’alguna part que no estigui com esperaria I durant o al final del projecte, són les següents:

- Afegir més diferents estils de sales a cada món, actualment només es compta amb una sala si no es té en consideració la de començament i d'enemic únic.

- Afegir més mons al joc. Ara hi ha només 2 mons, no és necessari tenir un enemic únic a cada món, podria tenir-se un enemic únic a l'últim món abans de canviar a un altre ambient.

- Posar un sistema per poder guardar la partida i continuar-la més endavant.

- Arreglar animacions que hagin quedat molt ortopèdiques. Sobretot les animacions dels atacs cos a cos.

- Millorar el joc visualment, en l'actualitat els menús són molt bàsics igual que la font utilitzada.

13. Bibliografia

35mm - Cuánto gana un guionista de videojuegos - (2023)

https://35mm.es/cuanto-gana-guionista-videojuegos/

PromocionMusical - Compositor de Música para Videojuegos - (s.d.)

https://promocionmusical.es/salidas-profesionales/compositor-musica-videojuegos/#Salario

Maria Lopez - ¿Cuánto cobra un desarrollador en España? - (2023)

https://www.softonic.com/articulos/cuanto-cobra-un-desarrollador-en-espana

Glassdoor - Sueldos para el puesto de Artista 2D en España - (2023)

https://www.glassdoor.es/Sueldos/artista-2d-sueldo-SRCH_KO0,10.htm

Jobted - Sueldo del Programador de Videojuegos en España - (s.d.)

https://www.jobted.es/salario/programador-videojuegos

EloLeChan - Elo's 8bit-style game music dump - (s.d.)

https://elolechan.itch.io/elos-8bit-style-game-music-dump

hzsmith - FREE 8BIT SOUNDTRACKS & SFX - (s.d.)

https://hzsmith.itch.io/vol2

0x72 - Dungeon tileset 2 – (s.d.)

https://0x72.itch.io/dungeontileset-ii

Unity Technologies - Unity -Manuals - (2022)

https://docs.unity3d.com/Manual/index.html

Asbjørn Thirslund - Brackeys - (s.d.)

https://www.youtube.com/user/Brackeys

Jake Brown - Jake Makes Games - (s.d.)

https://www.youtube.com/@JakeMakesGames

Michael Doyon - Mercenary Camp – (s.d.)

https://www.youtube.com/@N3KMercenaryCamp/featured

BravePixelG – (s.d.)

https://www.youtube.com/@BravePixelG

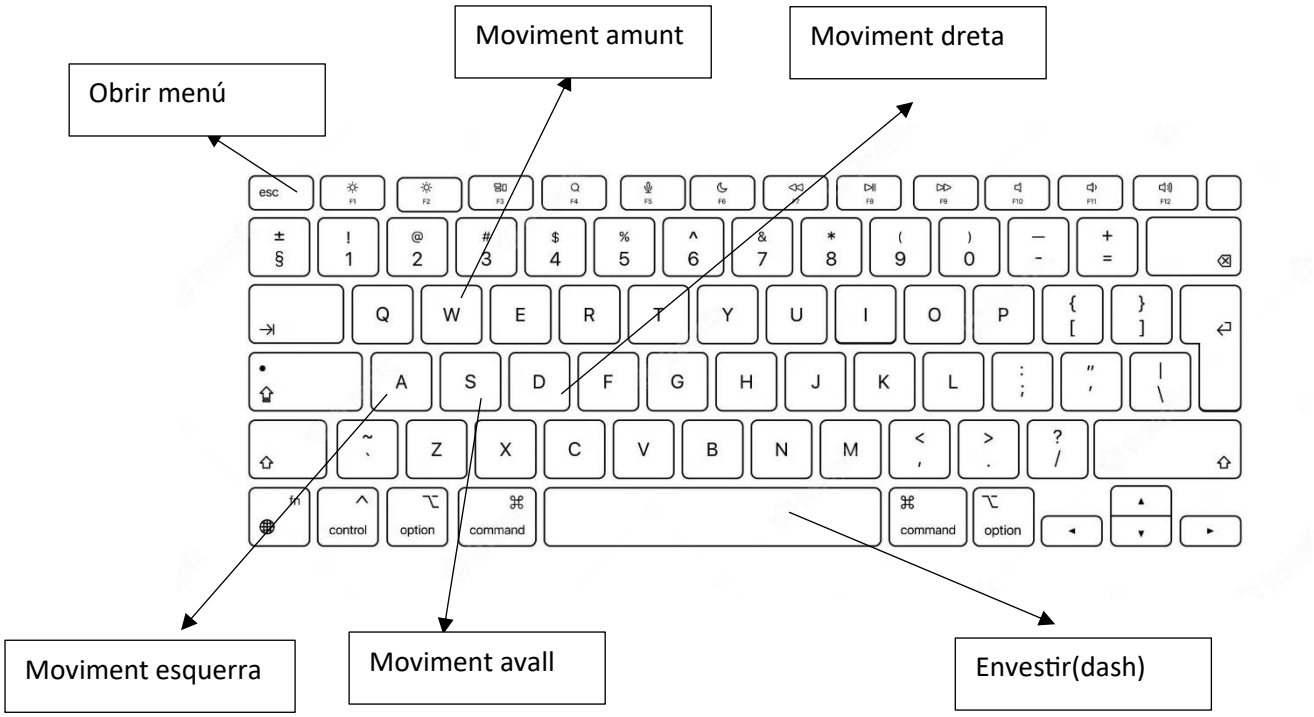
JVCOB – (s.d.)

https://www.youtube.com/@JVC0B/videos

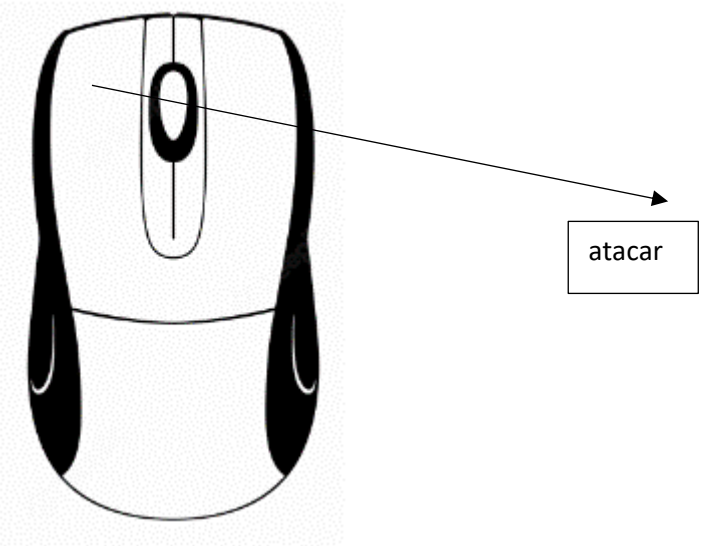
14. Manual d’usuari

Controls

El moviment del personatge funcionarà amb les tecles W,A,S,D i l’espai servirà per realitzar una envestida.



El botó esquerra del ratolí servirà per atacar amb el personatge.



Menú personatge

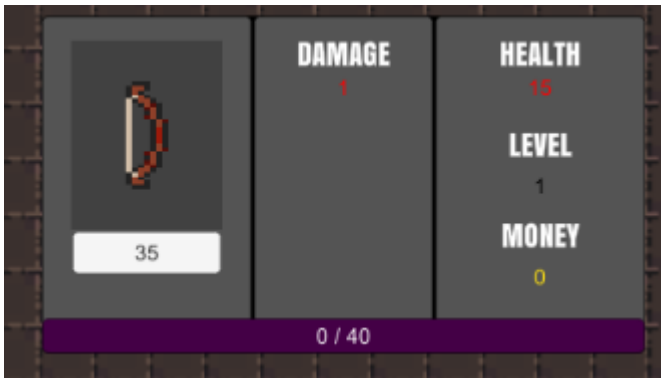


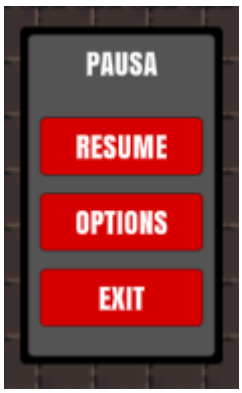
Figura 116 Menú de personatge

El personatge es pot millorar de dues maneres diferents, millorant l’arma i pujant de nivell:

Arma: cada personatge compta amb una arma i 2 millores que s’aconsegueixen gastant l’or que donen els enemics que va matant durant la masmorra. Cada millora canvia l’arma i puja el mal causat o, en cas de l’arquer, les fletxes.

Nivell: cada enemic dona un seguit d’experiència que, un cop arribat a l’experiència necessària, fa pujar de nivell pujant la vida màxima i recuperant tota la vida actual.

Menú d’opcions



Resume: Continuar el joc

Options: Obrir el menú d’opcions per canviar el volum

Exit: Tornar al menú principal

Figura 117 Menú de pausa

Recomanacions

- Els atacs cos a cos poden parar els projectils enemics

- Els atacs a distància quan es toquen amb projectils enemics desapareixen

- cada món té un enemic únic, recomanant pujar un nivell i millorar un cop l'arma cada món abans d’enfrontar l’enemic únic.

- La sala de l’enemic únic té algun distintiu a les portes per a distingir-lo de les sales normals.

Figura 118 Portes de les sales dels enemics únics