Universitat de Girona
Escola Politècnica Superior

Desenvolupament d'intel·ligències artificials dins un videojoc shooter en 2D

PROJECTE/TREBALL DE FI DE GRAU

Grau en Enginyeria Informàtica

Document: Memòria

Alumnes: Roger Canet

Tutor: Gustavo Patow

Departament: IMAE

Àrea: LSI

Convocatòria: 09/2022

Contingut

1 Introducció 5
1.1 Motivació 6
1.2 Objectius i propòsits del projecte 6
1.3 Organització del document 7
2 Estudi de viabilitat 9
2.1 Pressupostos inicials 9
2.2 Recursos humans 9
2.3 Avaluació de costos i mitjans 9
2.3.1 Estudi viabilitat tecnològica 9
2.3.2 Estudi viabilitat econòmica 10
3 Metodologia 11
4 Planificació 13
4.1 Tasques planificades 13
4.2 Temps estimat 15
5 Marc de treball i conceptes previs 16
5.1 Introducció als motors de videojocs 16
5.2 Exemples de motors de videojocs 16
5.2.1 Unreal engine 16
5.2.2 Unity 17
5.2.3 Game Maker Studio 18
5.2.4 Godot 19
5.3 Motor escollit 20
5.4 Introducció als arbres de comportament 20
6 Requisits del sistema 22
6.1 Requisits funcionals 22
6.2 Requisits no funcionals 22
7 Estudis i decisions 24
7.1 Unity 24
7.2 Visual Studio Code 24
7.3 GIMP 25
7.4 Microsoft Word 25
7.5 Assets de Unity 26
7.5.1 TheKiwiCoder Behaviour tree editor 26
7.5.2 Nav mesh plus 26

8 Anàlisi i disseny del sistema 28
8.1 Descripció general 28
8.2 Casos d'ús 28
8.2.1 Diagrames de casos d'ús 28
8.2.2 Fitxes de casos d'ús 31
8.2.3 Diagrames d'activitat 33
8.3 Scripts principals 36
8.3.1 Scripts del jugador. 36
8.3.2 Scripts dels enemics 36
8.3.3 Scripts de les bales 36
8.3.4 Script de control de nivells. 37
8.4 Enemics i arbres de comportament 37
8.4.1 Sniper 37
8.4.2 Zombie 38
8.4.3 Dash 38
8.4.4 Mutant 39
8.4.5 Shotgun 40
8.4.6 Bounce 41
8.4.7 Dodger 41
8.4.8 Clone 42
8.5 Bales 43
8.5.1 AllyBullet 43
8.5.2 EnemyBullet 44
8.5.3 BounceEnemyBullet 44
8.5.4 EnemySniperBullet 44
9 Implementació i proves 45
9.1 El desenvolupament a Unity 45
9.1.1 L'escena 45
9.1.2 Objectes 45
9.1.3 Atributs 46
9.1.4 Prefavs 48
9.1.5 Scripts 48
9.1.6 Nodes dels arbres de comportament. 49
9.2 Personatge principal 50
9.2.1 Scripts del jugador 50
9.3 Enemics 52

9.3.1 Scripts dels enemics....52
9.3.2 Nodes dels arbres de comportament creats....54
9.4 Bales....61
9.4.1 Scripts de les bales....61
9.4.2 Rebots de les BounceEnemyBullets....63
9.5 Menús....63
9.5.1 Menú principal i de nivells....63
9.5.2 Menú de pausa....64
9.6 Control de nivells i canvis d'escena....66
9.6.1 Control de nivells....66
9.6.2 Canvis d'escena....67
9.7 Problemes trobats durant el desenvolupament....67
9.7.1 NavMesh i NavMeshAgents....67
9.7.2 Errors en la construcció de l'executable del joc....67
10 Resultats....69
10.1 Legislació i normativa vigent....69
10.2 Captures de pantalla....69
11 Conclusions....73
12 Treball futur....74
13 Bibliografia....75
14 Manual d'usuari....76
14.1 Instal·lació....76
14.2 Objectiu del joc....77
14.3 Controls....77

1 Introducció

El sector dels videojocs esta en constant evolució des dels seus inicis amb jocs simples com el clàssic Pong fins a l'actualitat on s'estan desenvolupant nous videojocs a diari i amb noves eines de desenvolupament, aportant nous gràfics i física molt més realistes i fent de l'experiència mes atractiva, tant per nous públics com per a persones que ja gaudeixen d'aquests continguts.

Aquesta evolució ha anat acompanyada d'un increment en les vendes de videojocs i ha produït que més empreses estiguin interessades en el sector i es generi una major inversió, i així desenvolupant millors eines per facilitar el desenvolupament i alhora facilitant l'accés al desenvolupament de videojocs i garantint una major diversitat dels títols de manera que es pot atreure a major diversitat de públic cap aquest sector. Veure Figura 1.

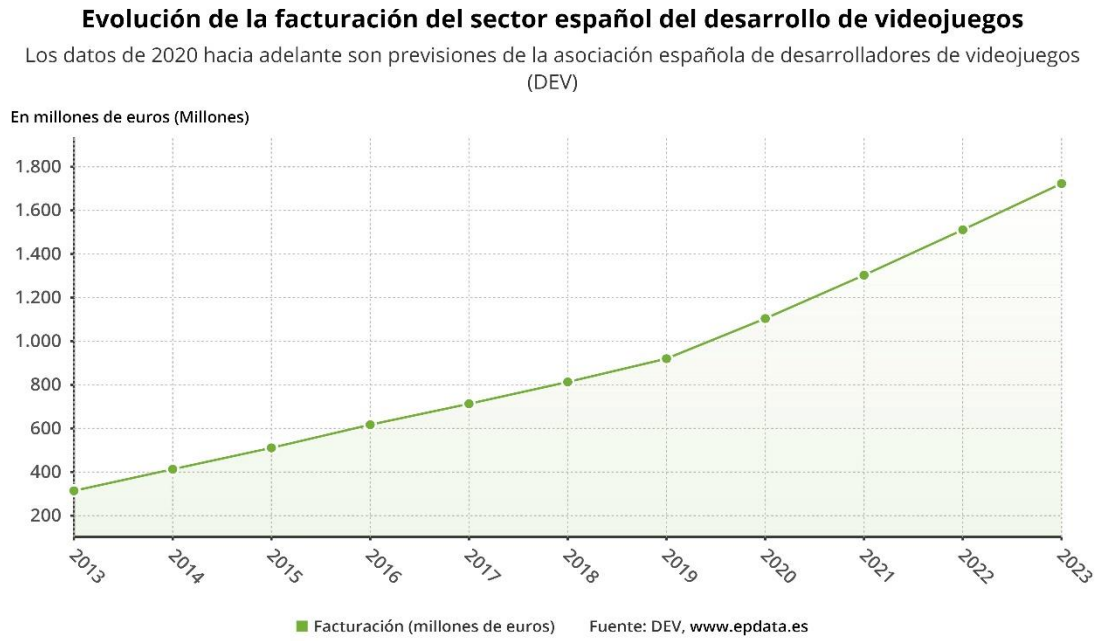

Figura 1. Gràfic de la evolució de la facturació del sector dels videojocs a Espanya, fet el 2020

Tot aquest desenvolupament ha desembocat en un sector on podem diferenciar dos tipus principals d'empreses desenvolupadores, les empreses "Indie", que son petits estudis de desenvolupament, i les empreses "triple A" , que son grans empreses dedicades al sector dels videojocs amb potents equips de desenvolupament. Generalment els llançament "triple A" solen ser videojocs molt populars tant per els estudis de màrqueting, que es fan com per la qualitat del videojoc utilitzant sempre les eines més modernes. En canvi els desenvolupaments "indie", es solen caracteritzar per uns desenvolupaments no tant populars, però entre aquests desenvolupaments pots trobar molta experimentació amb histories o mecàniques les quals no podries trobar en un "triple A", ja que a les grans empreses els interessa desenvolupar videojocs sense gaires conceptes experimentals per tal de assegurar beneficis un cop publicats.

1.1 Motivació

Els videojocs els podem classificar de moltes maneres, una de les més popular és segons la forma de jugar-se. D'aquesta manera podem distingir els jocs segons si són, per exemple, de rol, estratègia, plataformes, etc. En particular hi ha un gènere que sempre m'ha atret bastant, i aquest és el dels "Shooters", que es caracteritzen per ser videojocs d'acció i tirs. Actualment, la varietat d'aquest tipus de videojocs es escassa, la majoria son produïts per empresses "triple A" i repeteixen la saga amb petits canvis per tal de mantenir les ventes (Veure Figura 2).

Figura 2. Caratules de varies entreges de la saga Call of duty

Amb tanta repetibilitat, vaig descobrir els "shooters" en 2D, els quals et proposen una aproximació diferent als "shooters" tradicionals, proposant la perspectiva superior de manera que estan menys enfocats en la punteria del jugador, i més en la part estratègica, ja que faciliten una visió de tot el que envolta al jugador i proposen un repte per tal d'eliminar els enemics. L'únic problema que tenen aquests jocs és la poca quantitat que hi ha al mercat. Per tant vaig decidir que la base del videojoc seria d'aquest tipus, ja que podria buscar maneres de fer el gènere mes atractiu.

Un altre motiu pel desenvolupament d'aquest projecte ha estat la possibilitat de desenvolupar un videojoc, ja que poder desenvolupar el meu propi joc va ser el que em va portar a iniciar-me a la programació i també he pogut ajuntar-ho amb un dels camps de la informàtica que més curiositat em dona, que es la intel·ligència artificial. De manera que unint aquests dos camps, he decidit centrar el meu projecte en desenvolupar intel·ligències artificials per a videojocs

1.2 Objectius i propòsits del projecte

L'objectiu d' aquest projecte es el desenvolupament de la base d'un videojoc shooter 2D centrar en les IA dels enemics.

1.3 Organització del document

Aquesta memòria del projecte de final de grau esta dividida en 14 capítols en els quals es recopila tota la informació del projecte i les conclusions finals.

1. **Introducció:**
   En aquest capítol s'explica els motius pels quals he decidit fer aquest projecte, els propòsits i objectius d'aquest i també es defineix l'estructura del desenvolupament.
2. **Estudi de viabilitat:**
   En aquest punt es defineixen i justifiquen els paràmetres necessaris en el desenvolupament del projecte
3. **Metodologia:**
   Aquí es defineix la metodologia de treball que s'ha dut a terme al desenvolupar el projecte
4. **Planificació:**
   En aquest apartat, es defineix l'estratègia utilitzada per assolir els objectius del projecte, juntament amb la temporització d'aquesta.
5. **Marc de treball i conceptes previs:**
   En aquesta secció es realitza una introducció dels aspectes generals del projecte de manera que proporcioni un millor enteniment dels següents capítols de la memòria. Així mateix, hi haurà explicades les parts més importants de les primeres etapes del disseny i les parts del aprenentatge realitzant previ a la part de desenvolupament.
6. **Requisits del sistema:**
   En aquest capítol, s'especifiquen els requisits funcionals i no funcionals del sistema. Dins d'aquests hi ha inclosos els objectius i les funcionalitats que volem obtenir de l'aplicació.
7. **Estudi i decisions:**
   En aquest apartat s'especifiquen totes les eines utilitzades en el desenvolupament del projecte i es realitza una explicació de cada una d'elles.
8. **Anàlisi i disseny del sistema:**
   Aquí es defineix la investigació realitzada així com els problemes que s'han solucionat durant el desenvolupament. També es realitza una explicació dels detalls més importants del projecte, i esquemes de la implementació de les classes i mètodes.
9. **Implementació i proves:**
   Aquesta secció està dedicada a desenvolupar el procés de creació del videojoc, juntament amb una explicació de les classes i mètodes implementats que resultin necessaris per una major comprensió.
10. **Resultats:**
    En aquest apartat es mostraran les proves d'execució de la nostra aplicació, incloent tot el que l'usuari final podrà apreciar
11. **Conclusions:**
    Aquest capítol està dedicat a exposar les conclusions del projecte sobre els resultats finals obtinguts
12. **Treball futur:**
    Aquest apartat ens servirà per desenvolupar les possibles ampliacions o millores que es podrien realitzar al projecte.
13. **Bibliografia:**
    Aquí es podran trobar les referenciés a material extern al nostre treball utilitzades

14. **Manual d’usuari:**
    En aquest apartat desenvoluparem un petit manual per poder fer ús de l’aplicació.

2 Estudi de viabilitat

Pel desenvolupament d'aquest projecte no han estat necessaris grans costs de producció i el cost en quant a la infraestructura és nul.

El software utilitzat al llarg del desenvolupament d'aquest projecte ha estat:

- Windows 10 Home
- Motor de videojocs Unity
- Visual Studio code
- Gimp (Editor d'imatges)

El material tecnològic utilitzat ha consistit solament d'un ordinador de sobretaula.

Des de l'inici del projecte ja es comptava amb el material necessari per desenvolupar-lo. En quant als softwares utilitzats s'ha usat la versió gratuïta de tots ells.

2.1 Pressupostos inicials

Aquest projecte no requereix cap inversió inicial ja que consisteix en un projecte personal per fer una introducció dins les intel·ligències artificials i per tant no hi ha intencions de publicar aquest videojoc.

2.2 Recursos humans

Aquest projecte ha estat desenvolupat únicament per una sola persona, ja que consisteix en un projecte personal per tal de introduir-me en el mon del desenvolupament de videojocs i especialment l'apartat d'intel·ligència artificial.

Tot i això, per fer un projecte que fos atractiu visualment s'han utilitzat sprites i altres elements visuals de tercers d'ús gratuït.

2.3 Avaluació de costos i mitjans

Pel desenvolupament del projecte s'haurà de fer un estudi de la viabilitat d'aquest des dels punts de vista econòmic, tècnic i legal. Però, al tractar-se d'un videojoc, no ens hauria de presentar problemes legals.

2.3.1 Estudi viabilitat tecnològica

Aquest estudi es centra en definir el hardware necessari per desenvolupar el nostre projecte i els costos d'aquest.

Nosaltres ja disposem d'aquests requeriments de hardware, que consisteix en un ordinador de gamma mitja que es capaç de dur a terme el projecte, i per tant no hauríem de tenir cap problema en quant al hardware utilitzat.

2.3.2 Estudi viabilitat econòmica

L'estudi de viabilitat econòmica estarà dividit en dues parts, la part de costs humans i la de costos de maquinaria.

2.3.2.1 Costs de recursos humans

Aquest projecte tenia una intenció d'aprenentatge i desenvolupament personal dins el mon del desenvolupament de videojocs. Per tant, al ser un projecte de formació personal, partirem de que ha estat desenvolupat per una persona i assignarem a totes les hores el mateix cost, d'uns 12€/h com a costos d'un developer junior.

Tasca	**Hores**	**Cost**
Investigació	60h	720€
Estudi del motor unity	40h	480€
Disseny de la base del joc	20h	240€
Recerca d'art per l'apartat gràfic	4h	48€
Implementació de la base del joc	60h	720€
Disseny de les intel·ligències artificials	40h	480€
Desenvolupament de les IA	80h	960€
Implementació de la interfície d'usuari	10h	120€
Creació de nivells	20h	240€
Proves i testos	40h	480€
Memoria	70h	840€
Total		5328€

2.3.2.2 Costs de màquinaria

En quant a la maquinaria, hem usat software lliure i software del qual ja disposàvem. Per tant no hi ha cap altre cost que el desgast de la maquinaria usada, el qual definirem com un 20% del preu de la maquina usada, que consisteix en uns 500€.

D'aquesta manera podem assumir que el cost total d'aquest projecte és de 5828€

3 Metodologia

En quant a la metodologia utilitzada, es va decidir amb el tutor no fer servir una metodologia estandarditzada, sinó una de personalitzada per el projecte per adaptar-nos millor als objectius del projecte.

Els passos a seguir son els següents:

1. Escollir el tema del treball.
2. Decidir eines a utilitzar i aprendre el llenguatge que utilitzen i funcionament d'aquestes.
3. Estructurar projecte en parts per facilitar el desenvolupament.
4. Realitzar el desenvolupament d'una part.
5. Comprovar el correcte funcionament d'aquesta part.
    a. Si ens trobem errors en la part, es torna al punt 4 i continuem el desenvolupament.
    b. Si la part funciona correctament, es desenvolupa una nova part (punt 4) o s'uneix a la resta del projecte (punt 6).
6. S'uneixen diverses parts i es comprova que no ens produeixi errors.
    a. En el cas de que es produeixin errors tornaríem al desenvolupament de la part (punt 4).
    b. Si funciona correctament, es passaria a desenvolupar una nova part (punt 4) o quan totes les parts hagin estat acabades pasariem al punt 7.
7. Crear models de probes per fer comprovacions del correcte funcionament del sistema. Si hi ha algun error es torna al punt 4 per corretgir el problema dins la part corresponent.
8. Finalitzar i pulir la redacció de la documentació realitzada al llarg del projecte.

El diagrama de flux quedaria com es veu a la Figura 3.

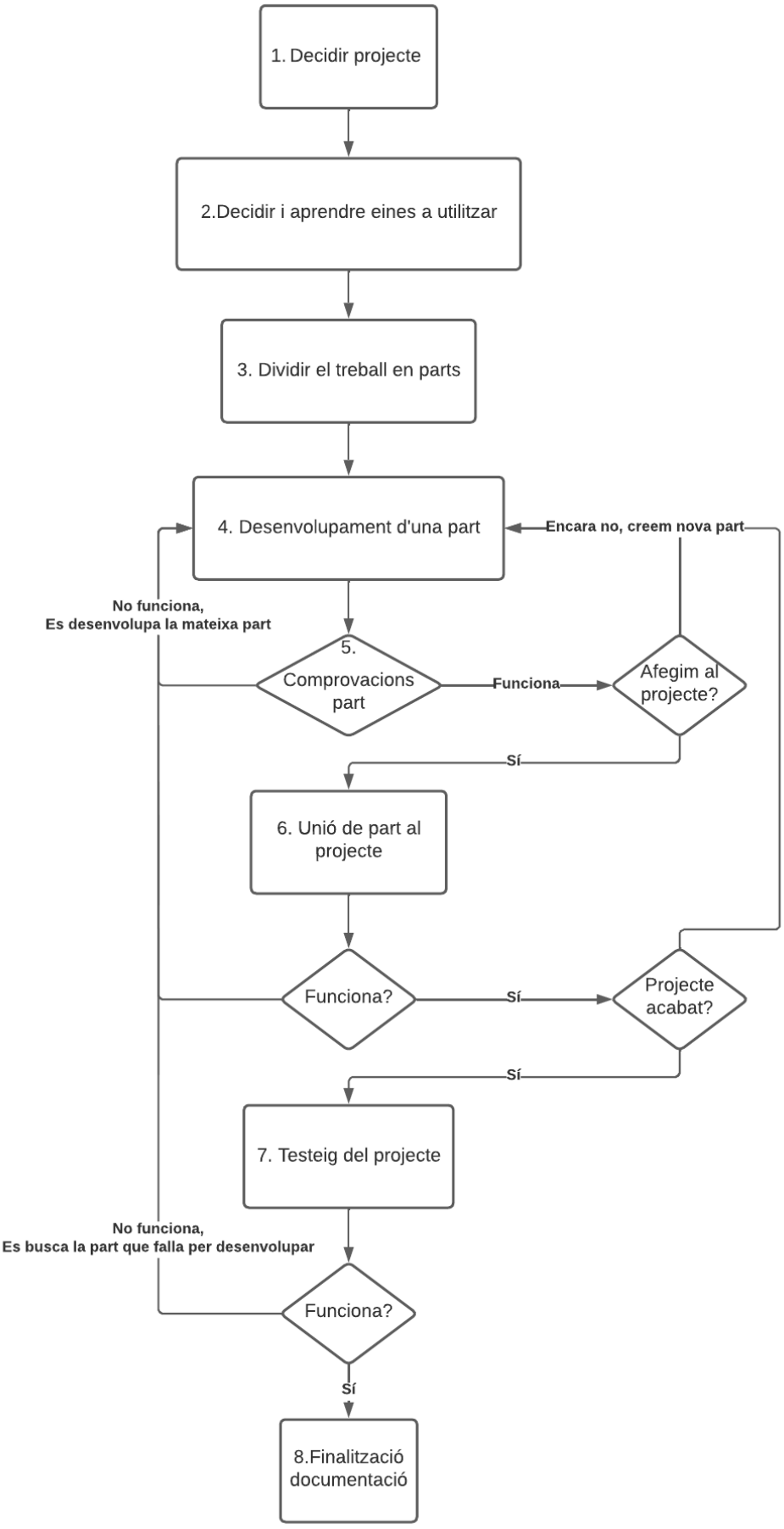

Figura 3 Diagrama de flux de la metodologia utilitzada

4 Planificació

Per fer una fàcil planificació del projecte el vam dividir en tasques concretes per poder fer un millor seguiment en hores.

4.1 Tasques planificades

Planificació del joc.

Aquesta tasca consistia en definir en quin tipus de joc consistiria en la base per la implementació de les IA

Estudi de motors de videojocs.

En aquest apartat es va fer un estudi dels diversos motors de videojocs per seleccionar el més adient per projecte

Estudi de Unity i el llenguatge C#.

En aquesta fase es va fer un estudi del funcionament de Unity i el desenvolupament en c# dins d'aquesta plataforma, per facilitar el futur desenvolupament, ja que mai abans havia desenvolupat usant Unity.

Disseny i implementació de la base del videojoc.

La tasca consisteix en un primer disseny i desenvolupament de les funcions bàsiques que volem dins el joc i el control del personatge per part del jugador.

Disseny dels enemics i les corresponents IA.

Apartat de disseny de els diferents tipus d'enemics, les funcions que duran a terme i característiques principals de cadascun per tal de fer-los diferents entre ells

Estudi de assets necessaris per les IA.

Per a la implementació d'intel·ligències artificials dins Unity ha estat necessari importar paquets per ampliar les funcions del motor (Assets). Hem realitzat un estudi per trobar i comparar els diferents paquets per trobar els més adients per el nostre programa.

Cerca d'elements gràfics.

La tasca consisteix en una cerca de material gràfic per fer el videojoc més atractiu a la vista i entenedor.

Implementació dels enemics i corresponents IA.

Implementació de les intel·ligències artificials prèviament dissenyades.

Testing del funcionament dels algoritmes de la IA.

Realització de tests per assegurar el correcte funcionament de les IA desenvolupades.

Disseny i implementació de diversos nivells per una Demo.

Creació de nivells per tal de fer mes amigable a l'usuari el resultat del projecte.

Creació de la interfície d'usuari

Desenvolupament d'una senzilla interfície per tal de facilitar la navegació entre els nivells de Demo creats

Documentació

Redacció de l'informe documentant el treball desenvolupat.

4.2 Temps estimat

S'ha realitzat un diagrama de Gantt per il·lustrar el temps estimat de les tasques a desenvolupar. Veure Figura 4.

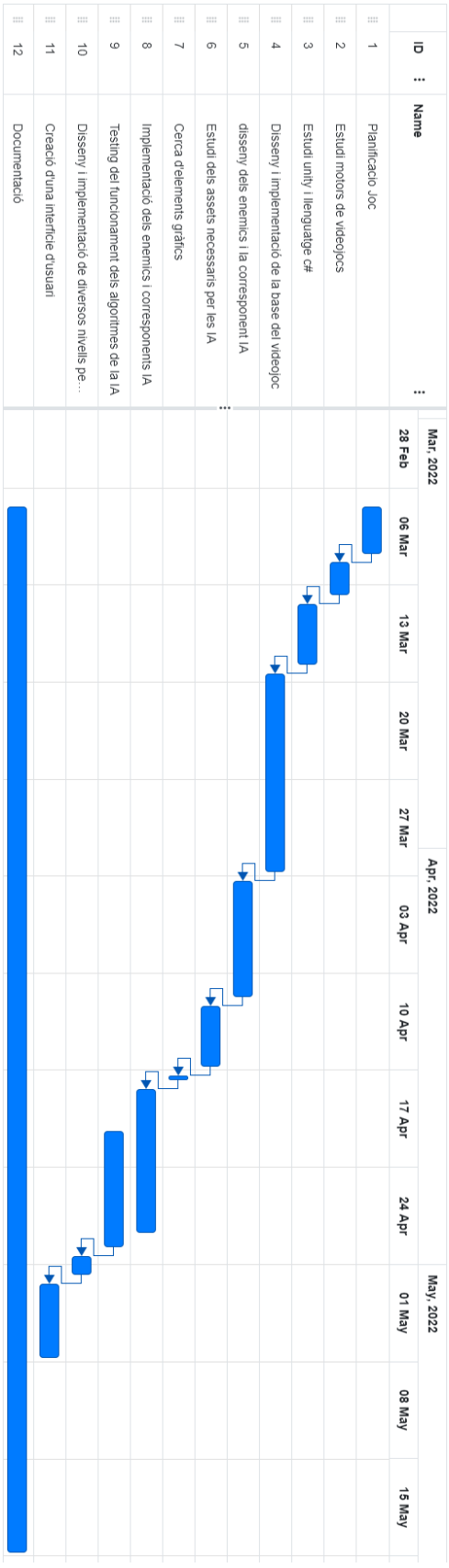

Figura 4 Diagrama de Gantt de la temporització estimada del projecte

5 Marc de treball i conceptes previs

En aquest punt tractarem els conceptes de l'etapa de preparació del projecte. També tractarem el perquè de l'elecció del motor de videojocs usat.

5.1 Introducció als motors de videojocs

Per facilitar el desenvolupament de videojocs és molt comú fer us de motors de videojocs, ja que son eines que ens permeten tenir en un mateix programari totes les funcions que necessitem d'un videojoc, com la programació, el disseny, el material gràfic, la física ...

Principalment un motor de videojocs serveix per facilitar la feina als desenvolupadors de videojocs de manera que els evita haver de treballar amb els aspectes més tècnics del sistema, els quals el motor de videojocs s'encarrega de simplificar-nos.

A l'hora de fer un videojoc és molt important que el desenvolupador triï un bon motor adient pel projecte.

Els avantatges de triar un bon motor serien:

- **Facilitat** en el desenvolupament del videojoc
- **Flexibilitat multi plataforma**, ja que el motor s'encarrega d'adaptar el videojoc a les plataformes que pugui suportar.
- **Facilitat d'ús**, ja que un bon motor pot reduir el nivell tècnic necessari i fer més senzill involucrar-se en el desenvolupament
- **Capacitat d'ampliació**, amb modificacions de l'eina i ampliacions(assets) creats pels desenvolupadors de l'eina o per la comunitat, es poden realitzar feines més complexes que originalment no ens permetia el motor.

5.2 Exemples de motors de videojocs

Aquests son alguns dels motors de videojocs més famosos

5.2.1 Unreal engine

Figura 5 Logotip d'Unreal engine

Unreal engine va ser creat el 1998 per la companyia Epic Games, i va començar centrat en el desenvolupament de jocs en primera persona. Aquest motor ha anat evolucionant amb

millores gràfiques i una alta portabilitat, fent d'aquest motor un dels més usats en grans desenvolupaments actuals. Veure Figura 6

Figura 6 Captura de pantalla videojoc Fornite creat amb Unreal engine

5.2.2 Unity

Figura 7 Logotip de Unity

Creat el 2005 per Unity tecnologies, és un motor de videojocs que ens permet una alta portabilitat i es considerat un dels mes senzills per desenvolupadors que s'inicien en el món del desenvolupament de videojocs. Aquest motor ens permet desenvolupar videojocs en 3D i 2D usant el llenguatge C#.

Unity és un motor molt popular dins del videojoc Indie i a més permet compartir assets (extensions del programa) a través de la seva pròpia asset Store. Veure Figura 8.

Figura 8 Captura de pantalla de My friend Pedro videojoc creat amb Unity

5.2.3 Game Maker Studio

Figura 9 Logotip de Game Maker Studio 2

Game Maker Studio és un motor de videojocs dissenyat el 1999 per Yoyo games. Va ser creat específicament per videojocs 2D i accessible per a desenvolupadors amb pocs coneixements de programació. Aquesta accessibilitat ve donada perquè el motor funciona amb un sistema de programació "drag and drop" que permet al desenvolupador no haver de modificar codi, tot i això per als programadors avançats també es poden crear scripts en un llenguatge conegut com Game Maker Language. Veure Figura 10.

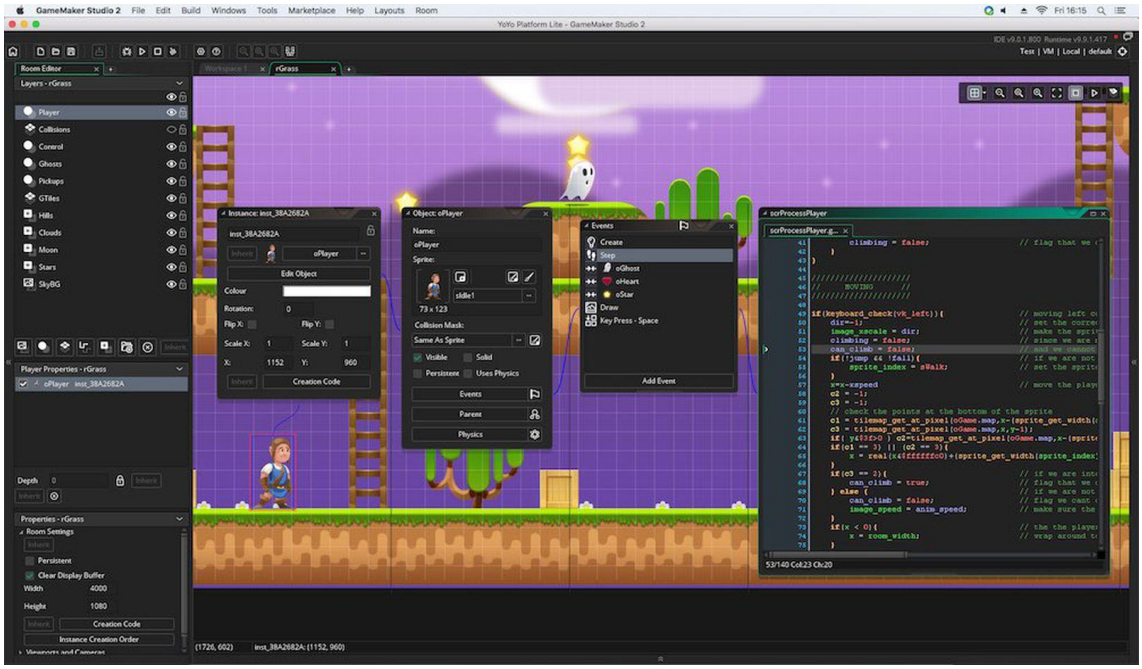

Figura 10 Interfície del Game Maker Studio 2

5.2.4 Godot

Figura 11 Logotip Godot

Godot és un motor de videojocs llançat el 2007 i desenvolupat per la comunitat de Godot. És un programa de software lliure i codi obert, sota la llicencia MIT.

Godot ens permet realitzar desenvolupaments de videojocs tant en 2D com 3D i es poden exportar a PC dispositius mòbils i HTML5. Principalment la programació amb aquest motor es fa amb un pseudocodi anomenat GDscript, però també es pot fer en C++, C# i VisualScritpt. Veure Figura 12.

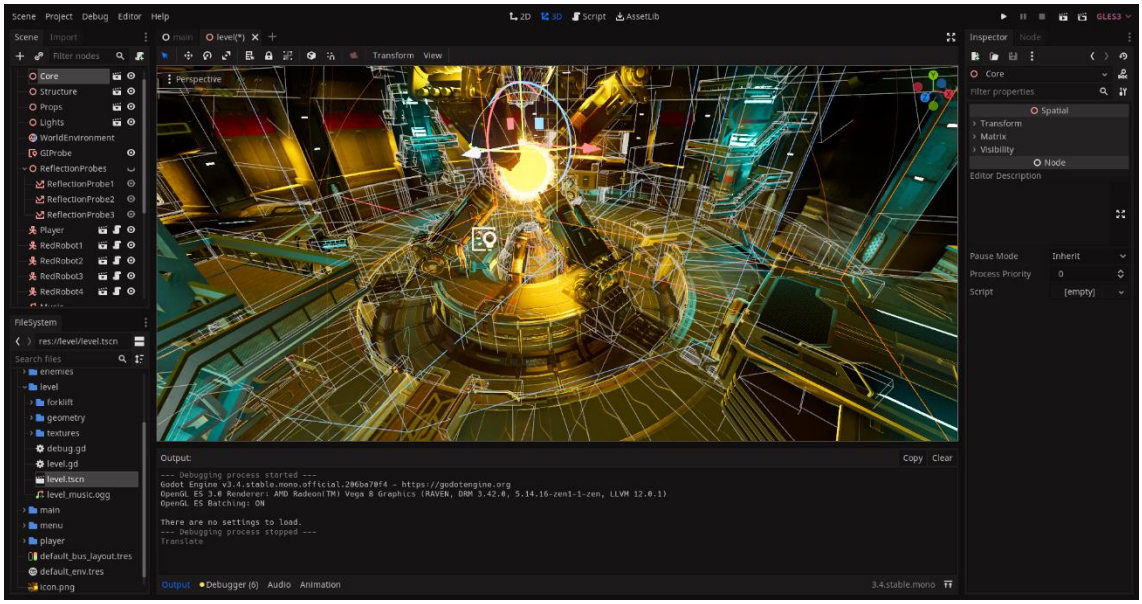

Figura 12 Interfície del motor de videojocs Godot

5.3 Motor escollit

Un cop realitzat l'estudi de diversos possibles motors, s'ha decidit usar el motor Unity. S'ha escollit aquest motor per la seva popularitat i l'accessibilitat per a desenvolupadors novells. També ens aporta una documentació excel·lent i té una comunitat activa plena de tutorials i altres informacions útils a l'hora de solucionar problemes concrets.

A més aquest com hem mencionat anteriorment, aquest motor compta amb la asset Store que és una eina molt bona a l'hora d'afegir components al motor base per fer desenvolupaments més complexos, i en el cas d'aquest projecte hem fet us d'aquests assets per tal de fer un millor desenvolupament de les intel·ligències artificials.

5.4 Introducció als arbres de comportament

En aquest projecte, pel desenvolupament de les intel·ligències artificials, hem utilitzat arbres de comportament per tal de programar les intel·ligències dels enemics.

Un arbre de comportament s'encarrega de determinar l'acció que durà a terme un determinat NPC (personatge no controlat per el jugador). Per fer-ho es desenvolupa un arbre matemàtic on el node arrel llença un senyal que recorre l'arbre, començant pels nodes de més a l'esquerra fins que arriba a l'arrel un valor d'èxit. Els nodes interiors s'anomenen nodes de flux o control i les fulles, nodes d'execució.

Els nodes d'execució són aquells que porten les tasques a fer del personatge, es a dir les accions que volem que l'arbre ens torni com a resultat. Aquestes accions tenen tres estats, activa, èxit i fracàs. L'estat actiu, vol dir que el node encara esta treballant i per tant encara no podem continuar. Èxit o fracàs ho obtindrem un cop acabi la tasca i aquest resultat ens servirà perquè els nodes de flux i control puguin continuar operant fins que l'arrel rebi un senyal d'èxit. En el cas que l'arrel rebi un senyal de fracàs voldrà dir que hi ha un problema en el codi i ens servirà per detectar problemes.

Els nodes de flux i control, s'encarreguen de modificar el senyal que reben de manera que es puguin realitzar varies operacions més complexes amb els fills i així permetre'ns realitzar múltiples tasques dins l'arbre.

Per exemple entre els nodes de flux i control més comuns podem trobar:

- **Selector:** Aquest node recorre els fills d'esquerra a dreta fins que rep un fill amb èxit i retorna un èxit al pare. Si no rep cap èxit dels fills, retorna fracàs.
- **Seqüenciador:** Al contrari que el selector, aquest node recorrerà els fills mentre retornin èxit. Si acaba de recórrer els fills, retornarà al pare un èxit, però si rep un fracàs retornarà fracàs.
- **Selector aleatori:** El node triarà aleatòriament un dels fills i el resultat del fill serà el que retorni al pare

També podem trobar nodes de flux que només poden tenir un fill, per exemple:

- **Repetició:** Aquests nodes s'encarreguen de repetir la tasca o tasques que realitzi el seu fill fins que aquest retorni èxit o fracàs segons s'hagi configurat el node. Un cop acabat retornarà l'estat corresponent que ha trencat el bucle, es a dir, si repetia fins a rebré un fracàs, retornarà fracàs.
- **Inversor:** La funció d'aquest tipus de nodes es invertir el resultat que els retorni el fill, si reben èxit retornaran fracàs i a l'invers.

Un exemple d'un arbre de comportament acabat quedaria com es veu a la Figura 13 on podem veure l'arrel com el node en vermell, els nodes de flux i control de taronja i blau, i els nodes d'execució de verd.

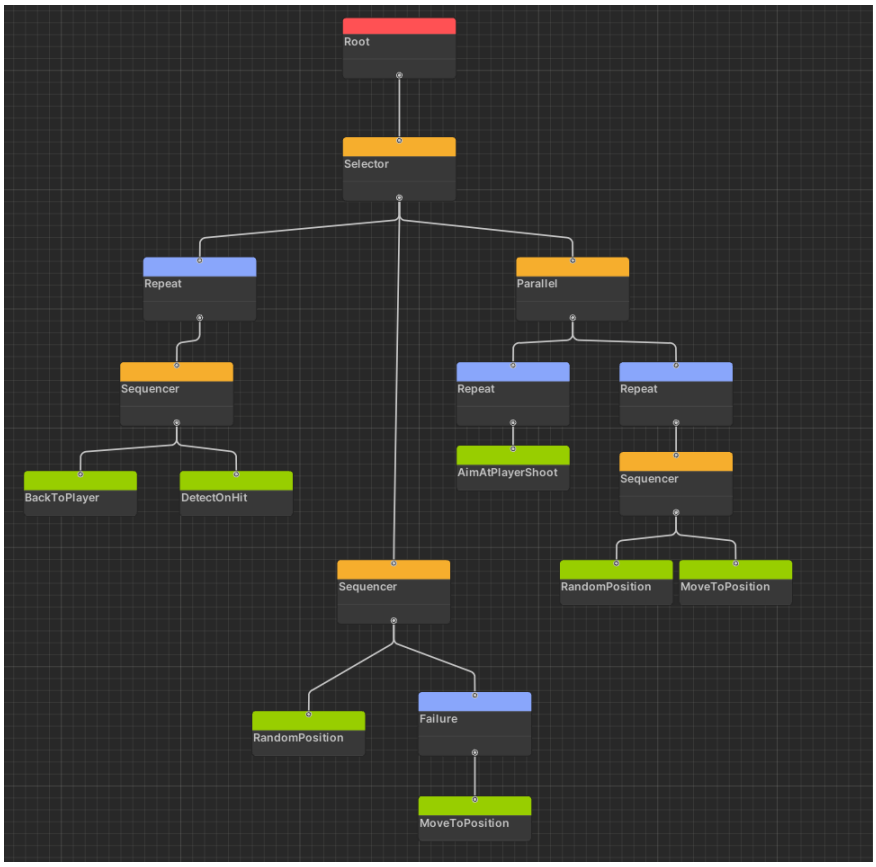

Figura 13 Arbre de comportament desenvolupat en el projecte

6 Requisits del sistema

Aquest apartat especificarem els requeriments de l'aplicació dividint-los entre funcionals i no funcionals.

6.1 Requisits funcionals

Aquests són els requeriments sobre les funcionalitats que ha de dur a terme el sistema.

Els requisits funcionals del videojoc seran:

- A l'iniciar el joc, el jugador haurà de poder començar una partida, seleccionar un nivell o sortir.
- Dins dels nivells, el jugador haurà de ser capaç de desplaçar-se amb el teclat i apuntar i disparar amb el ratolí.
- EL jugador haurà de derrotar els enemics per poder passar de nivell.
- El jugador haurà de disparar a un enemic per derrotar-lo.
- Els enemics hauran de disparar al jugador i utilitzar diferents estratègies per derrotar-lo.
- Tant el jugador com els enemics seran derrotats amb un impacte de bala.
- El jugador podrà distingir els diferents enemics i les respectives estratègies.
- El jugador podrà fer pausa en qualsevol moment del nivell.
- Amb el nivell pausat, el jugador podrà continuar-lo, tornar al menú principal o sortir del joc.
- Si eliminen al jugador, aquest repetirà el nivell.

Al ser un videojoc, tots els usuaris disposaran dels mateixos privilegis a l'usar l'aplicació. Per tant podem considerar que només tindríem un actor per aquesta aplicació, que seria el jugador.

6.2 Requisits no funcionals

Els requeriments no funcionals son aquells externs a l'aplicació que es necessiten perquè aquesta funcioni correctament. És a dir, són els recursos necessaris per a garantir un bon funcionament de l'aplicació.

Aquest programa només es pot fer servir en mode usuari i tampoc guarda cap tipus de dades confidencials, per tant no conta amb cap tipus de protecció de dades.

Les restriccions de plataforma són les del motor, és a dir Unity.

Els dispositius d'entrada necessaris per tal que l'usuari interactuï correctament amb l'aplicació consistirien en un teclat i un ratolí.

En quant als requeriments tècnics, l'aplicació ha estat testejada en un ordinador de sobretaula amb les següents característiques i sense problemes de rendiment:

- CPU: Intel Core i7-12700K 3.9 GHz
- GPU: GeForce RTX 3070 Ti
- RAM: 32GB
- SO: Windows 10 Home 64 Bits

Tot i això els requisits mínims serien els requisits mínims que ens marca el motor, en el nostre cas Unity. Veure Figura 14

Minimum requirements	Windows	macOS	Linux
Operating system version	Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only.	High Sierra 10.13+ (Intel editor) Big Sur 11.0 (Apple silicon Editor)	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	X64 architecture with SSE2 instruction set support	X64 architecture with SSE2 instruction set support (Intel processors) Apple M1 or above (Apple silicon-based processors)	X64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, and DX12-capable GPUs	Metal-capable Intel and AMD GPUs	OpenGL 3.2+ or Vulkan-capable, Nvidia and AMD GPUs.
Additional requirements	Hardware vendor officially supported drivers	Apple officially supported drivers (Intel processor) Rosetta 2 is required for Apple silicon devices running on either Apple silicon or Intel versions of the Unity Editor.	Gnome desktop environment running on top of X11 windowing system, Nvidia official proprietary graphics driver or AMD Mesa graphics driver. Other configuration and user environment as provided stock with the supported distribution (Kernel, Compositor, etc.)
	For all operating systems, the Unity Editor is supported on workstations or laptop form factors, running without emulation, container or compatibility layer.		

Figura 14 Requisits mínims Unity

7 Estudis i decisions.

En aquest apartat realitzarem una descripció dels programes usats i també els assets de unity que s'han utilitzat. Per cada programa s'explicara l'ús que s'en ha fet i perquè s'ha escollit.

7.1 Unity

Figura 15 Logotip de Unity

Tal com s'ha explicat al punt 5.3, hem decidit usar unity perquè és un motor molt usat i que ofereix moltes facilitats a nous developers. També ens permet ampliar les seves funcions a través de les assets que ens permetran crear les IA.

7.2 Visual Studio Code

Figura 16 Logotip de Visual Studio Code

Visual Studio code és un editor de codi font desenvolupat per Microsoft. S'ha escollit pel desenvolupament dels scripts en C# per la seva simplicitat i facilitat d'ús, també és un editor el qual ja havia usat anteriorment i hi tenia preferència sobre altres.

A més, a part de ser un editor molt simple, ens permet afegir-hi extensions per tal de fer més còmode el desenvolupament en qualsevol llenguatge. Per desenvolupar aquest projecte he usat principalment dues extensions, una per tal que l'editor corregeixi el codi en C# i l'altre perquè aquest em detecti classes especials de Unity.

Finalment, un altre factor que ha fet que preferíssim aquest editor és que és un programari de codi obert i, per tant, no ens ha suposat un cost extra.

7.3 GIMP

Figura 17 Logotip de GIMP

GIMP és un programa d'edició d'imatges gratuït i de codi obert, que forma part del projecte GNU. Aquest editor està especialitzat en la edició i retoc d'imatges, el qual treballa les imatges en format de capes. L'editor ens ha servit per modificar, retallar i atorgar els formats adequats als sprites descarregats de https://opengameart.org/ per tal de fer-los aptes per Unity.

He decidit usar aquest programa per la prèvia experiència que ja tenia amb aquest i també perquè és un dels millors editors en quant a eines que ofereix i sent gratuït.

7.4 Microsoft Word

Figura 18 Logotip de Word

El Word és un software de processament de texts creat per Microsoft dins del paquet de Microsoft Office. Aquest software, juntament amb altres de disseny de gràfics, s'ha utilitzat a l'hora de redactar la memòria.

En el nostre cas, hem triat aquest software ja que, tot i no ser gratuït, he pogut utilitzar la llicencia gratuïta d'estudiant que ens ofereix la universitat. També és un dels millors editors de text del mercat i un dels més utilitzats i és l'editor de text on hi tinc més experiència.

7.5 Assets de Unity

Per el desenvolupament del videojoc s'han necessitat alguns assets per afegir funcionalitats a unity que no teníem.

7.5.1 TheKiwiCoder Behaviour tree editor

Consisteix d'un editor gràfic de arbres de comportament per a Unity. Aquest editor de arbres de comportament va ser desenvolupat per l'usuari de Youtube TheKiwiCoder i el va desenvolupar en varis vídeos en el seu canal, i posteriorment va fer l'asset públic i gratuït.

Vaig preferir aquest asset abans que altres de abres de comportament, per l'ajuda proporcionada per l'autor i la guia de com usar-lo, ja que, amb altres assets, em vaig trobar problemes a l'hora de crear nodes customitzats i ,en canvi, amb aquest asset el desenvolupament de nous nodes era més simplificat i el desenvolupament dels arbres era més entenedor.

Aquest asset ens va servir per la creació de les intel·ligències artificials dels enemics i és una de les parts principals del projecte.

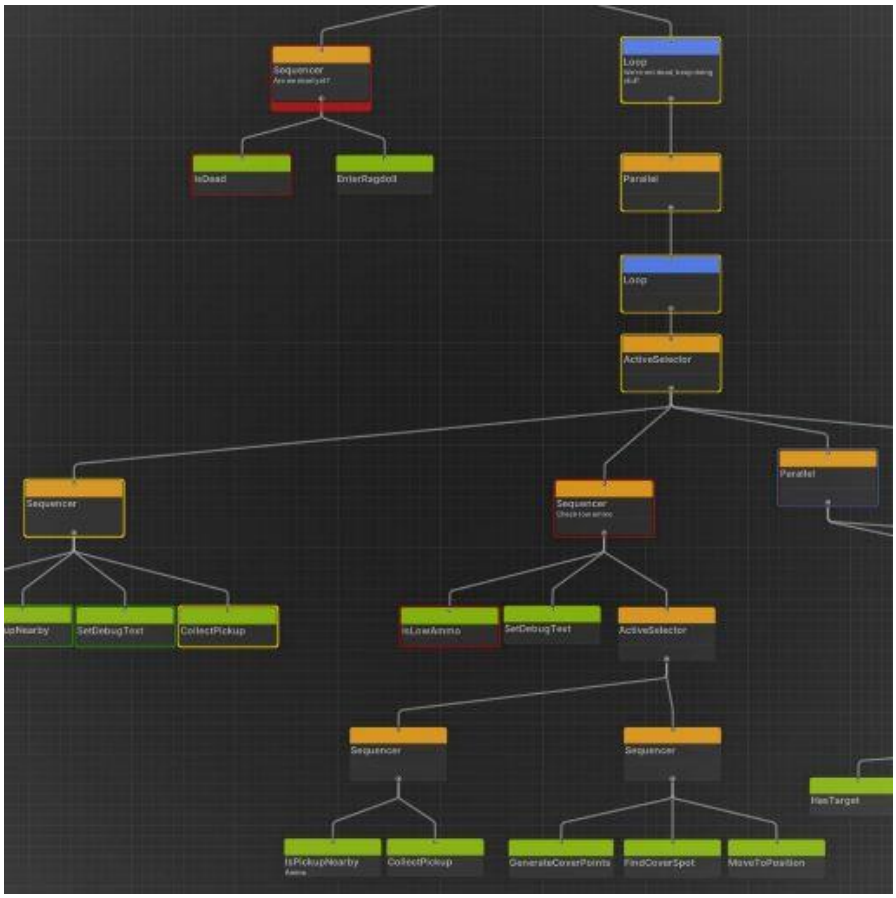

Figura 19 Visualització d'un arbre de comportament amb l'editor de TheKiwiCoder

7.5.2 Nav mesh plus

Desenvolupat per l'usuari h8man juntament amb aportacions d'altres desenvolupadors a github.

Aquest asset va ser necessari durant el desenvolupament ja que, a l'hora de moure els NPCs, s'ha de definir l'espai del joc on es poden moure per tal que no col·lisionin amb l'escenari. Per fer-ho es sel·lecciona la zona per on es poden moure amb un nav mesh.

En principi Unity ofereix un nav mesh, però durant el desenvolupament vaig trobar-me amb el problema que el nav mesh de Unity estaba només adaptat per 3D, i vaig necessitar fer ús d'aquest asset per tal de poder crear un nav-mesh dins del videojoc 2D.

Principalment la funció d'aquest asset és donar les eines del nav mesh 3D de Unity per aplicar-les en l'entorn 2D.

8 Anàlisi i disseny del sistema

En aquest capítol farem una descripció general del videojoc desenvolupat i els diferents elements més importants que conté. Tota descripció serà feta de forma conceptual i sense entrar en la implementació.

8.1 Descripció general

El videojoc desenvolupat és un shooter en 2D utilitzant una càmera zenital estàtica col·locada en el centre del nivell de manera que el jugador pot veure tot l'escenari sense moure la càmera. Per part del jugador, els controls són simples, el moviment del personatge per l'escenari es realitzarà amb el teclat i el funcionament d'apuntar i disparar amb el ratolí.

El funcionament del videojoc consistirà en que el jugador ha d'eliminar a tots els enemics del nivell per poder continuar al següent nivell. Tant els enemics com el jugador disposen d'una vida, per tant el jugador ha de ser capaç d'eliminar als enemics amb un tir, però ell també pot ser eliminat d'una bala. Si els enemics aconsegueixen eliminar al jugador, aquest haurà de repetir el nivell. Cada enemic té unes característiques diferents de manera que es pugui sentir únic per al jugador. Aquestes característiques són tant visuals com del comportament, per tal que el jugador pugui distingir-los i adaptar la seva estratègia a l'hora d'enfrontar-se als nivells.

8.2 Casos d'ús

Aquí definirem com l'usuari podrà interactuar amb el joc. Per fer-ho definirem els casos d'ús del videojoc, que ens marcaran les accions que l'usuari podrà dur a terme i el comportament del sistema a partir de les seves interaccions.

8.2.1 Diagrames de casos d'ús

Al tractar-se d'un videojoc on no es necessita cap tipus d'usuari especial, tot actor que interactuï amb el videojoc el considerarem un jugador i tots rebran els mateixos permisos.

8.2.1.1 Diagrama de casos d'ús del menú principal

Des del menú principal l'usuari té les opcions de Iniciar una partida al primer nivell, carregar un nivell concret o sortir del joc. Veure Figura 20.

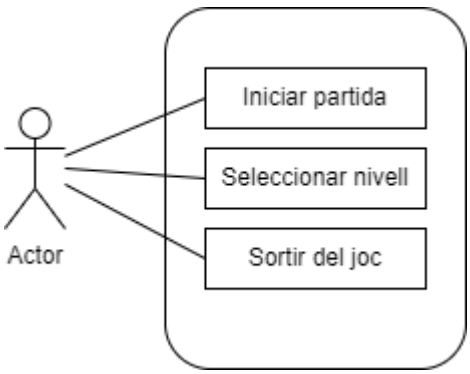

Figura 20 Diagrama de casos d'ús del menú principal

8.2.1.2 Diagrama de casos d'ús del menú de nivells

Des d'aquest menú l'usuari ha de poder seleccionar qualsevol dels nivells disponibles per jugar o tornar al menú principal. Veure Figura 21.

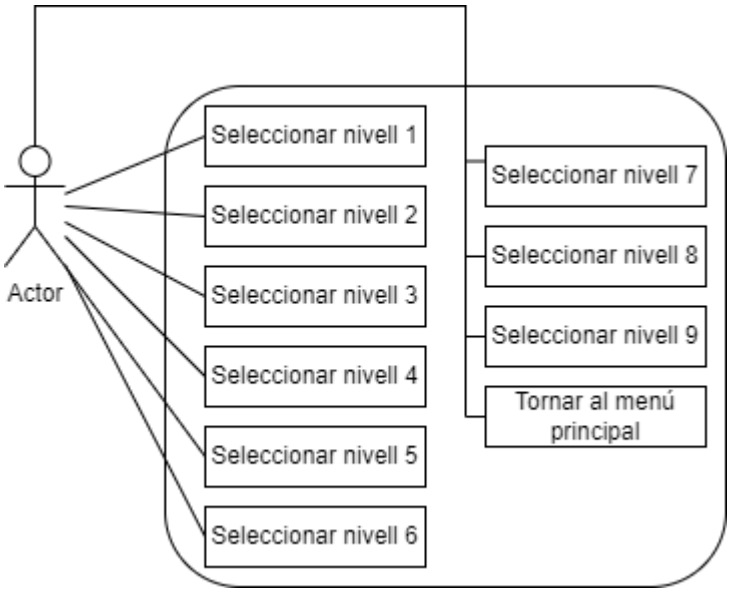

Figura 21 Diagrama de casos d'ús del menú de nivells

8.2.1.3 Diagrama casos d'ús del joc

Mentre l'usuari esta jugant ell pot moure's, disparar i fer pausa. Veure Figura 22.

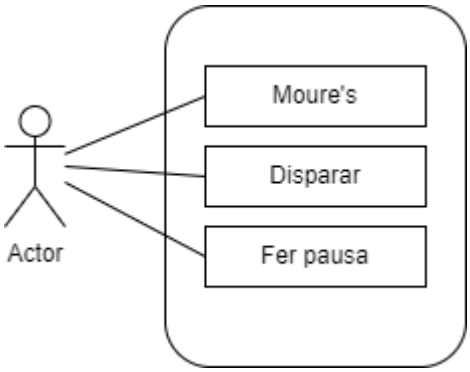

Figura 22 Diagrama casos d'ús del joc

8.2.1.4 Diagrama casos d'us del menú de pausa

Quan el jugador pausa la partida, pot continuar-la, sortir al menú principal o sortir del joc. Veure Figura 23.

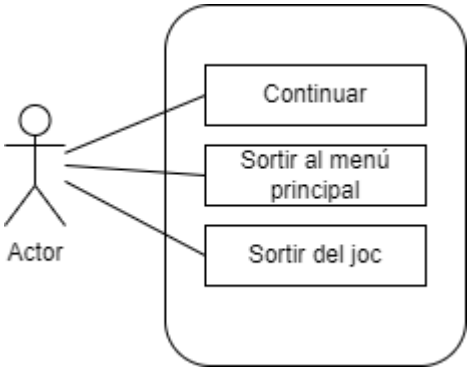

Figura 23 Diagrama casos d'us del menú de pausa

8.2.1.5 Diagrama casos d'ús fi de nivell

Un cop el jugador ha acabat un nivell, sigui perquè ha guanyat o ha perdut, arriba a aquest menú. Si el jugador ha perdut, podrà reiniciar el nivell i si el jugador ha guanyat podrà començar el següent nivell. Veure Figura 24.

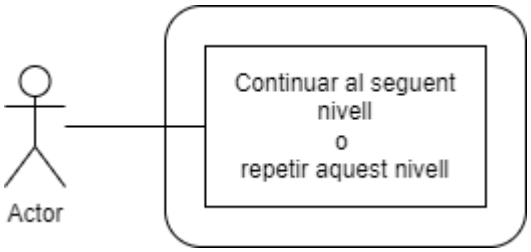

Figura 24 Diagrama casos d'ús fi de nivell

8.2.2 Fitxes de casos d'ús

Les fitxes de casos d'ús ens serviran per desenvolupar els casos d'ús incloent-hi la descripció, l'actor que ho realitza, les seves precondicions, els fluxos principal, les postcondicions i les observacions trobades.

Fitxa de cas d'ús: Iniciar partida	
Descripció	EL jugador ha seleccionat "Play" en el menú principal
Actor	Jugador
Precondició	Aplicació iniciada
Flux principal	1. S'ha escollit l'opció "Play" en el menú principal
Postcondició	S'ha carregat el primer nivell
Observacions	Cap

Fitxa de cas d'ús: Obrir menú seleccionar nivell	
Descripció	EL jugador ha seleccionat "Levels" en el menú principal
Actor	Jugador
Precondició	Aplicació iniciada
Flux principal	1. S'ha escollit l'opció "Levels" en el menú principal
Postcondició	S'ha carregat el menú per seleccionar el nivell
Observacions	Cap

Fitxa de cas d'ús: Sortir del joc	
Descripció	EL jugador ha seleccionat "Quit" en el menú principal o en el menú de pausa
Actor	Jugador
Precondició	Aplicació iniciada / Nivell iniciat i pausat
Flux principal	1. S'ha escollit l'opció "Quit" en el menú principal o en el menú de pausa
Postcondició	S'ha tancat l'aplicació
Observacions	Cap

Fitxa de cas d'ús: Seleccionar nivell	
Descripció	EL jugador ha seleccionat un nivell del menú de nivells.
Actor	Jugador
Precondició	S'ha accedit al menú de nivells
Flux principal	1. S'ha seleccionat un nivell
Postcondició	S'ha carregat el nivell corresponent
Observacions	Cap

Fitxa de cas d'ús: Tornar al menú principal	
Descripció	El jugador ha seleccionat "back" des del menú de nivells.
Actor	Jugador
Precondició	S'ha accedit al menú de nivells
Flux principal	1. S'ha seleccionat "back"
Postcondició	S'ha carregat el menú principal
Observacions	Cap

Fitxa de cas d'ús: Disparar	
Descripció	El jugador dispara un projectil en la direcció del ratolí.
Actor	Jugador
Precondició	S'ha iniciat una partida.
Flux principal	1. S'ha fet clic esquerre amb el ratolí 2. Es troba la posició del ratolí en aquell instant 3. Es genera una bala des de l'arma del personatge en direcció al ratolí 4. La bala segueix una trajectòria recta fins a colisionar amb un obstacle. 5. Si l'obstace és un enemic aquest s'elimina.
Postcondició	El personatge ha disparat
Observacions	Cap

Fitxa de cas d'ús: Moure's	
Descripció	El jugador es desplaça.
Actor	Jugador
Precondició	S'ha iniciat una partida
Flux principal	1. S'ha premut una de les tecles de moviment. 2. Es desplaça el personatge en la direcció corresponent.
Postcondició	El personatge s'ha desplaçat
Observacions	Cap

Fitxa de cas d'ús: Fer pausa	
Descripció	Durant una partida el jugador fa pausa
Actor	Jugador
Precondició	S'ha iniciat una partida.
Flux principal	1. S'ha premut la tecla de pausa. 2. Es desactiven els controls del jugador i s'atura tot el moviment del joc 3. Es mostra el menú de pausa
Postcondició	S'ha obert el menú de pausa i el joc esta parat
Observacions	Cap

Fitxa de cas d'ús: Continuar jugant	
Descripció	El jugador surt del menú de pausa per continuar la partida.
Actor	Jugador
Precondició	S'ha iniciat una partida i després s'ha pausat
Flux principal	1. S'ha seleccionat "resume" en el menú de pausa 2. Es reactiven els controls del jugador i tots els elements tornen a l'estat anterior a haver pausat la partida.
Postcondició	El Jugador continua la partida iniciada en el mateix estat que anteriorment havia pausat.
Observacions	Cap

Fitxa de cas d'ús: Sortir al menu	
Descripció	El jugador surt de la partida cap al menú principal usant el menú de pausa
Actor	Jugador
Precondició	S'ha iniciat una partida i després s'ha pausat
Flux principal	1. S'ha seleccionat "menu" en el menú de pausa 2. Es tanca el nivell i es carrega el menú principal
Postcondició	S'ha carregat el menú principal
Observacions	Cap

8.2.3 Diagrames d'activitat

Hem realitzat diversos diagrames d'estat per representar gràficament les accions del jugador i la resposta per part del sistema.

8.2.3.1 Diagrama d'activitats dels menús principal i per seleccionar nivell

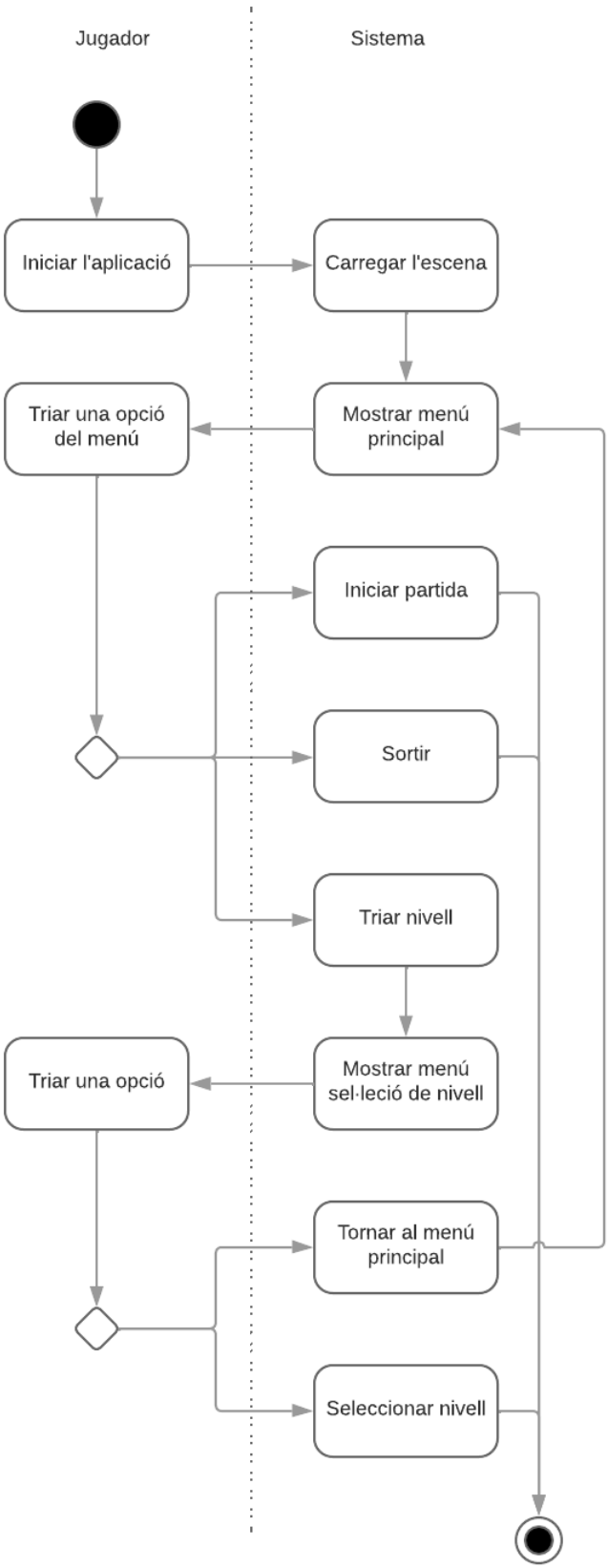

Figura 25 Diagrama d'activitats dels menús principal i per seleccionar nivell

8.2.3.2 Diagrama d'activitats d'un nivell

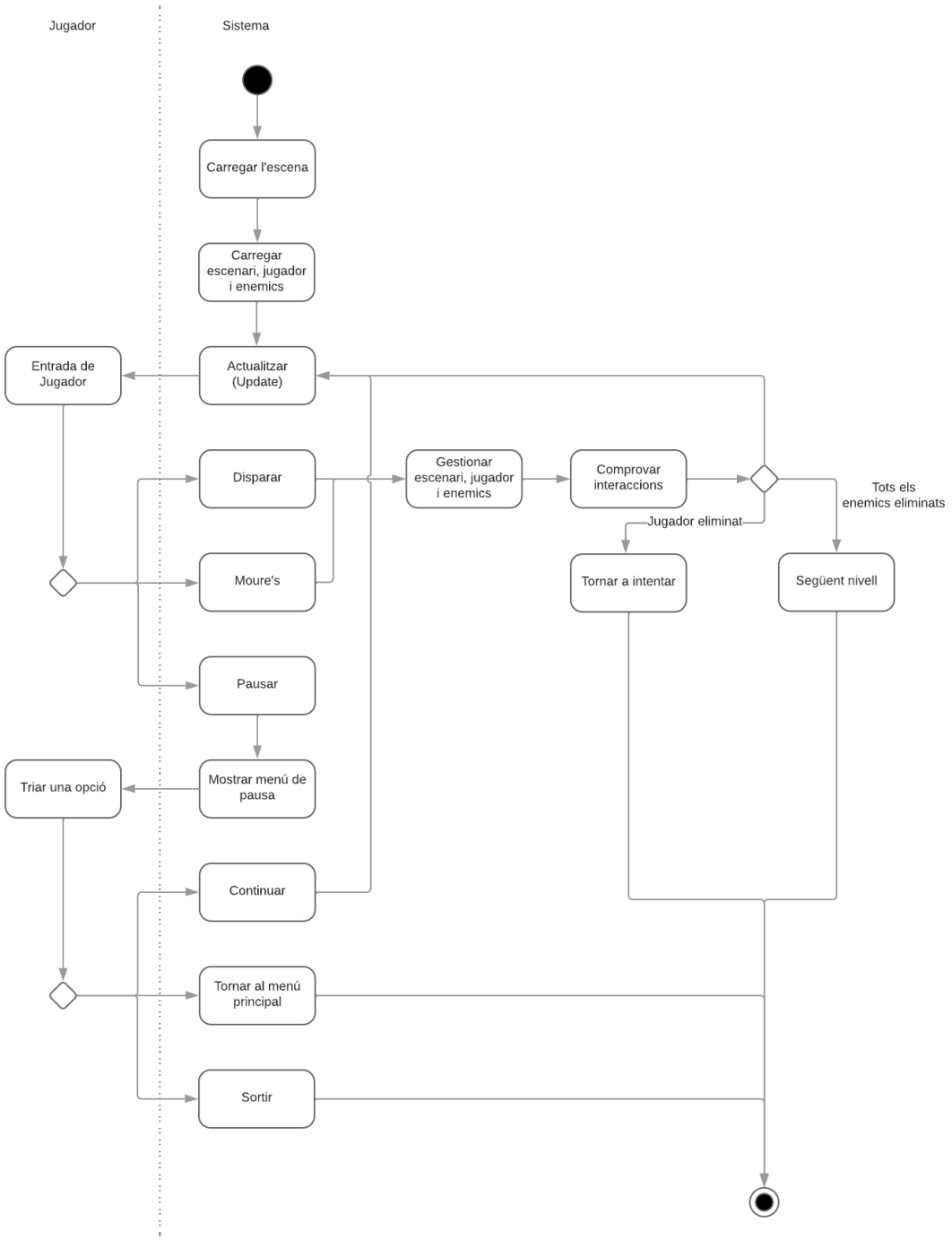

Figura 26 Diagrama d'activitats d'un nivell

8.3 Scripts principals

El desenvolupament de videojocs en Unity funciona a través d'scripts en C#, que consisteixen en classes de C# que ens serveixen per fer interactuar els objectes de l'escenari juntament amb la física corresponent. Els scripts s'han volgut mantenir simples i el codi està explicat més amb profunditat al Capítol 9.

8.3.1 Scripts del jugador.

Durant el joc, el jugador té de dos possibles accions a fer que serien moure's i disparar i podem definir aquestes accions usant dos scripts.

8.3.1.1 PlayerController

Utilitzat per rebre l'entrada de teclat de teclat del jugador i moure el personatge en la direcció corresponent. Utilitzant la física de Unity s'evita que el jugador pugui travessar parets o sortir del mapa.

8.3.1.2 PlayerWeapon

Utilitzat per el control de l'arma, amb aquest script aconseguirem que el jugador es mantingui apuntant al ratolí mentre juguem. A més, aquest script serà l'encarregat de detectar el clic del ratolí per crear una bala en la direcció del ratolí.

8.3.2 Scripts dels enemics

Pels enemics el seu funcionament principal, tant el moviment com l'apuntat, ha estat a través dels nodes dels arbres de comportament. Però, per la funció de disparar s'ha creat un script per tal de que tots els enemics partissin de la mateixa base i així facilitar la interacció dels arbres amb l'arma.

8.3.2.1 Weapon

Aquests script ens permet fer que l'enemic dispari. Ho fem a través de la funció shoot(), a més aquesta classe contindrà varies variables, que seran editables des de la interfície de Unity que ens permetran modificar la bala, el temps entre bales i el tipus de tir.

8.3.3 Scripts de les bales

En el joc s'ha dissenyat varis tipus de bales però totes parteixen d'uns scripts similars. La seva funció és moure l'element bala en una velocitat definida, fins a xocar amb un element. Un cop xoquen es destrueix la bala i si l'element era un personatge s'elimina. S'han creat tres scripts similars per definir els comportaments.

8.3.3.1 PlayerBulletController

Aquest script serveix per a les bales del jugador i detecta si xoquen contra un enemic i sí és així, el destrueixen. En canvi, no eliminen al jugador si el toquen.

8.3.3.2 *EnemyBulletController*

Es fa servir aquest script per a les bales dels enemics. Aquestes bales no eliminen als enemics si col·lideixen amb ells, però si xoquen amb el jugador l'eliminen.

8.3.3.3 *BounceEnemyBulletController*

Aquest script funciona igual que l'anterior, però esta modificat per unes bales que reboten amb les parets. Aquesta classe compta amb una variable per establir un màxim de rebots, un cop la bala ha arribat al màxim es destrueix.

8.3.4 Script de control de nivells.

Per tal de mantenir un control de la vida del jugador i els enemics, s'ha realitzat un script (LevelControl.cs) que actualitza si hi ha un jugador viu i enemics vius, per tal de saber l'estat de la partida i mostrar la pantalla de victòria o derrota respectivament.

8.4 Enemics i arbres de comportament

Per tal de crear unes intel·ligències artificials complexes s'han creat varis arbres de comportament que ens permeten donar-li als enemics un caràcter més únic.

8.4.1 Sniper

L'sniper, veure Figura 27, consisteix en un franctirador. Per tant, l'hem caracteritzat per una mobilitat nul·la, però a l'hora d'enfrontar-los es complica per la velocitat de les seves bales, fent que el jugador hagi de buscar estratègies per exposar-se el mínim.

Figura 27 Sprite de Sniper

Al ser un enemic estàtic, l'arbre de comportament ha resultat molt simple, però ha estat útil, ja que els nodes usats s'han fet servir en els altres arbres.

L'arbre, consisteix de tres nodes, l'arrel, un node per repetir les accions i finalment l'acció d'apuntar al jugador i disparar-lo. Veure Figura 28.

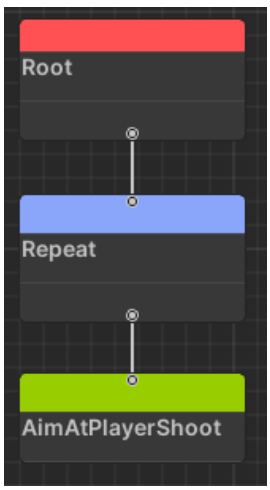

Figura 28 Arbre de comportament Sniper

8.4.2 Zombie

El zombie, veure Figura 29, consisteix en un enemic sense arma de tir. En canvi, aquest enemic elimina al jugador si el toca. Tot i això, l'enemic resta inactiu fins que el jugador s'apropa el suficient. Un cop el zombi detecta el jugador, aquest anirà a l'ultima posició on ha detectat al jugador. Aquesta detecció funciona a través de murs i permet que el jugador pugui sortir de l'àrea de detecció per tal d'escapar d'ells. El zombi, al no disparar, és un objectiu fàcil pel jugador, però a distancies curtes es perillós pel jugador.

Figura 29 Sprite de Zombie

L'arbre consisteix en repetir la seqüencia de buscar el jugador en una posició pròxima i moure's a la posició. Veure Figura 30.

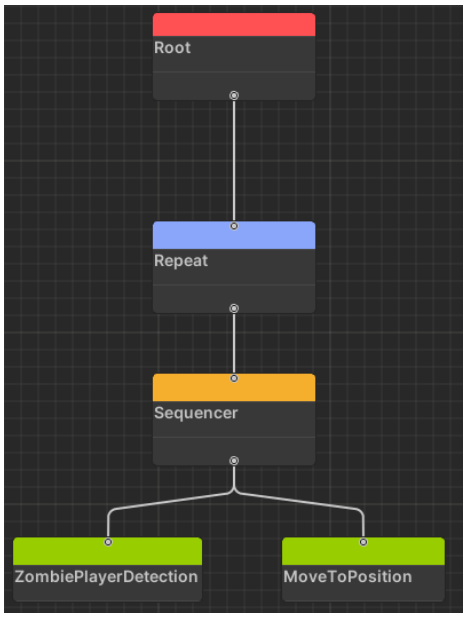

Figura 30 Arbre de comportament de Zombie

8.4.3 Dash

El Dash, veure figura 31, és un enemic que realitza desplaçaments curts i ràpids, i entre aquests desplaçaments, si té línia directe de tir al jugador, es a dir, no hi ha obstacles entre ells dos, realitza un tir per després continuar amb un altre desplaçament. Una característica d'aquest enemic és que, al disparar, ho fa en forma de con, disparant tres bales, una en línia recte i les altres dues amb una diferencia de +10º i -10º, respectivament, de la central. Veure Figura 32

Figura 31 Sprite de Dash

Figura 32 Dash disparant

L’ arbre consisteix en una repetició d’una seqüencia de 4 accions consecutives, que serien, triar una posició pròxima, moure’s a la posició, dispar al jugador i finalment esperar abans de fer el següent desplaçament. Veure Figura 33.

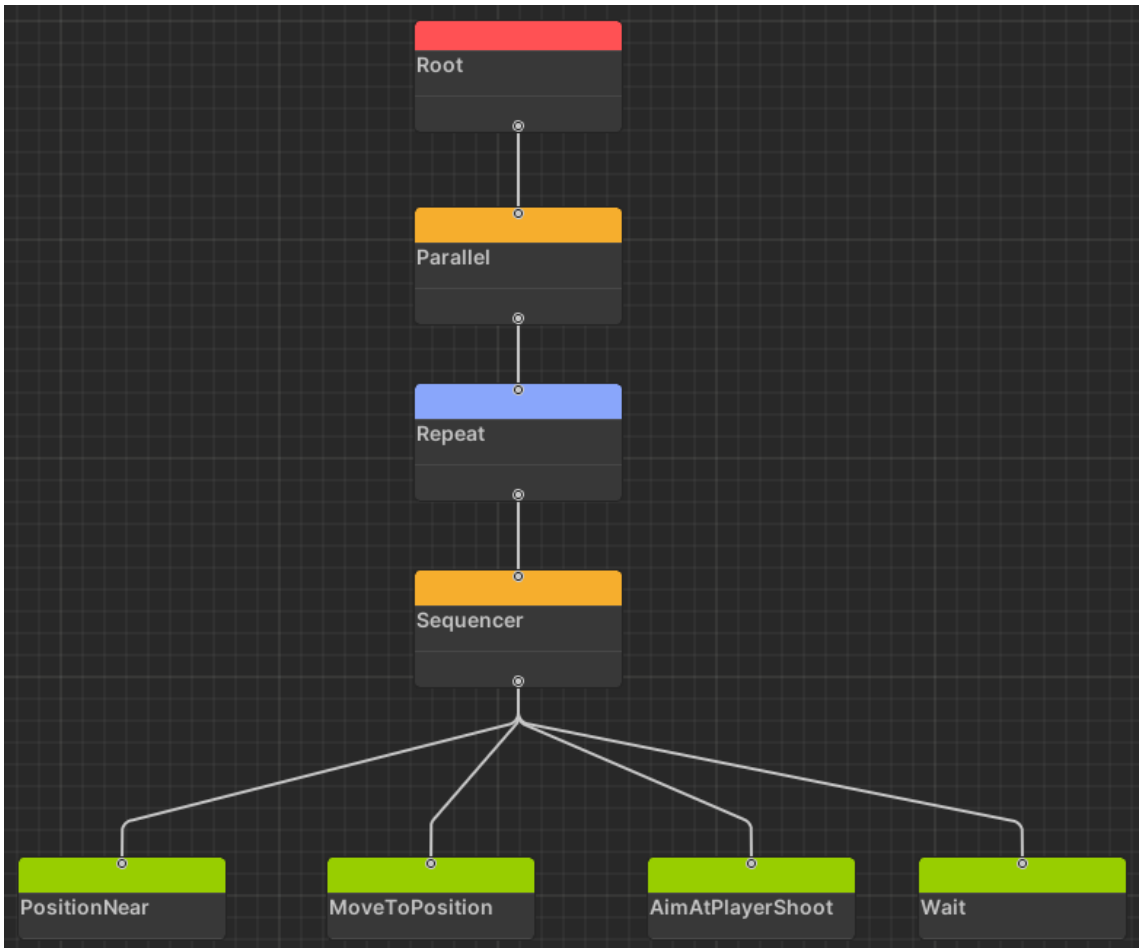

Figura 33 Arbre de comportament de Dash

8.4.4 Mutant

El Mutant, veure Figura 34, és un enemic gran que persegueix el jugador amb un arma amb alta cadència de tir, de manera que la seva funció és perseguir a l’usuari per atrapar-lo entre bales. Una avantatge que té el jugador contra aquest enemic és la mida del mutant, ja que, al ser més gros el jugador, pot tenir més opcions per eliminar-lo.

Figura 34 Sprite de Mutant

Ja que aquest personatge persegueix a l’usuari mentre dispara, ho podem veure refelectit a l’arbre de comportament, ja que usem un “Parallel” que dona senyal al dos fills alhora. Aquests fills son la funció de disparar i la de situar-se a prop del jugador. Veure Figura 35.

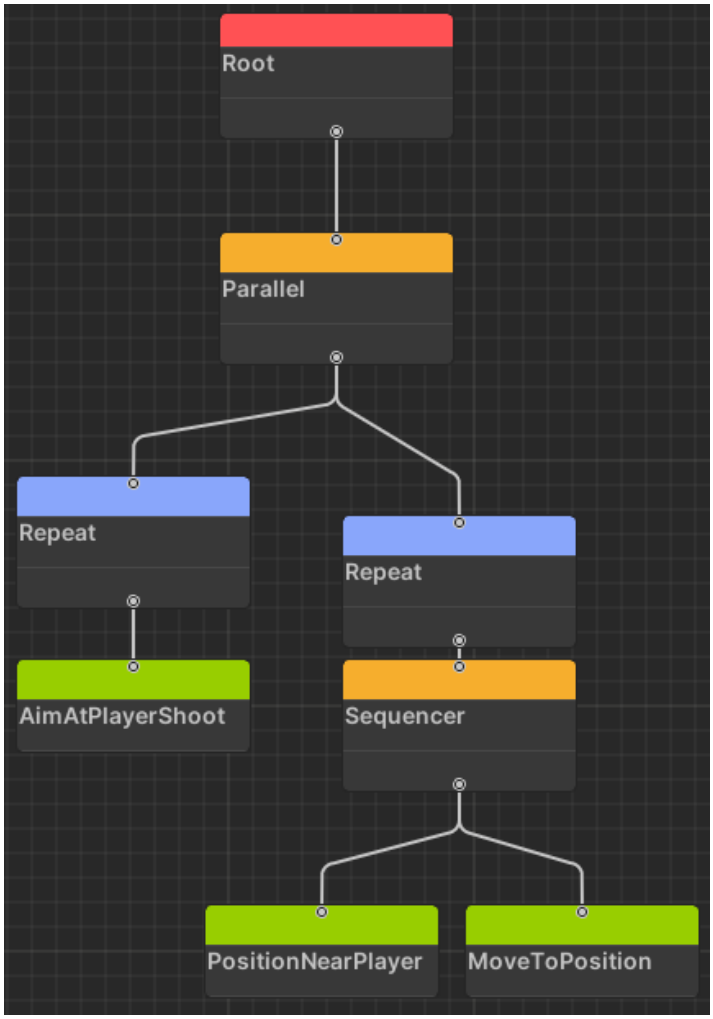

Figura 35 Arbre de comportament de Mutant i de Shotgun

8.4.5 Shotgun

Aquest personatge, veure Figura 36, esta creat com una altre opció per atrapar l'usuari amb un escenari amb moltes bales. A diferencia del Mutant, aquest personatge va equipat amb la mateixa arma que el Dash, es a dir, un arma que dispara tres bales en forma cònica.

Figura 36 Sprite de Shotgun

Tant el Shotgun com el Mutant tenen comportament molt similar, es dediquen a perseguir a l'usuari mentre el disparen, i per això comparteixen l'arbre de comportament. Però, a l'hora d'estar en una partida, les diferents formes de disparar dels dos marquen una a diferencia molt gran i el jugador necessitarà estratègies diferents.

8.4.6 Bounce

Aquest enemic, veure Figura 37, porta una nova variació a les bales que dispara, ja que aquestes reboten a les parets de manera que el jugador ha de anar amb copte amb els rebots. Aquest enemic també té la funció d'apuntar modificada, mentre que els altres enemics només disparen quan tenen visió directe del jugador, aquest personatge, si no té visió directe del jugador, agafarà un angle entre +90º i -90º en direcció al jugador i dispararà.

Figura 37 Sprite de Bounce

L'arbre de comportament d'aquest personatge és similar al mutant, però té el node d'apuntar i disparar canviat i en lloc de buscar al jugador, l'enemic es mourà per a prop d'on ell estava. Veure Figura 38.

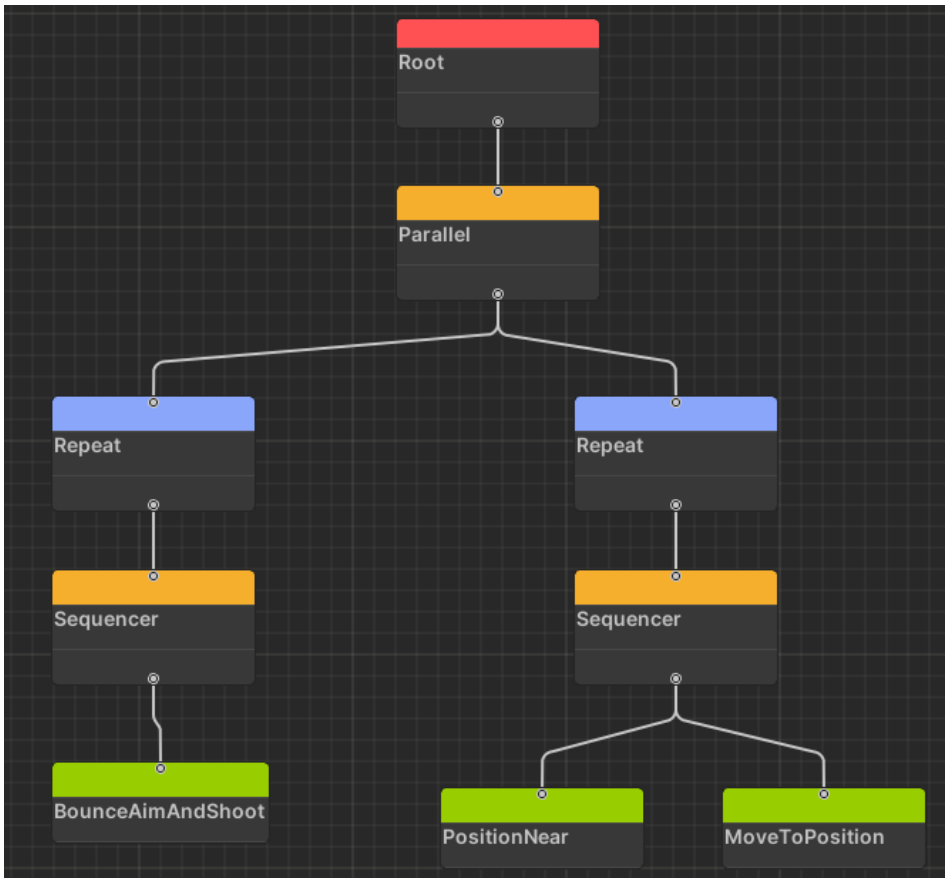

Figura 38 Arbre de comportament de Bounce

8.4.7 Dodger

El Dodger, veure Figura 39, és un enemic que intenta esquivar les bales del jugador. Aquest enemic inicia estàtic disparant al jugador, fins que detecta una bala del jugador. Al detectar-la, procedeix a desplaçar-se a prop de la seva ubicació inicial per tornar-se estàtic i així cada vegada que detecta una bala.

Figura 39 Sprite de Dodger

L'arbre d'aquest personatge consisteix en node d'apuntar que ens retorna fracàs si detecta una bala per tal que pare d'aquest el selector activa la

seqüencia de canviar de posició. Veure Figura 40.

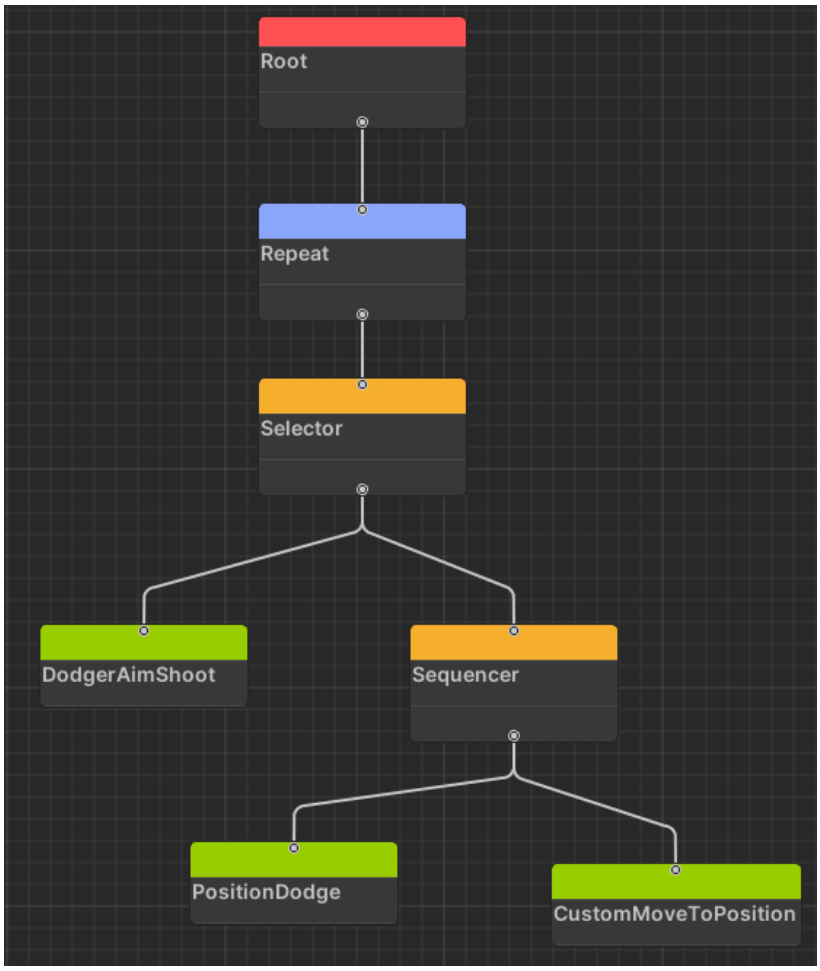

Figura 40 Arbre de comportament de Dodger

8.4.8 Clone

La intenció del Clone, veure Figura 41, és enganyar al jugador pensant que l'ha eliminat per després tornar a aparèixer i lluitar contra el jugador. Podríem definir el comportament del Clone com a tres fases, una primera on dona l'esquena al jugador i quan aquest el dispara es torna invisible i comença la segona fase, mentre és invisible es recol·loca en el mapa i després comença la tercera fase on lluita contra el jugador amb una velocitat de moviment més alta que els altres enemics.

Figura 41 Sprite de Clone

Observant l'arbre de comportament d'aquest personatge, veure Figura 42, podem veure com de l'arrel hi ha un selector, que és l'encarregat de fer els canvis de fase. Al primer semi-arbre a l'esquerra podem veure que té les funcions de donar l'esquena al jugador i detectar si l'han disparat. Un cop ha estat disparat retornarà fracàs i això provocarà que el selector fill del root ens canviï de fase. La segona fase consisteix en seleccionar una posició i moure'ns a ella. Per provocar la tercera fase, al moure'ns tenim un node que ens retorna fracàs si o si i així fem que el selector canviï de fase. A l'ultima fase tenim un semi-arbre molt similar als arbres que hem vist anteriorment on l'enemic es mourà i dispararà alhora.

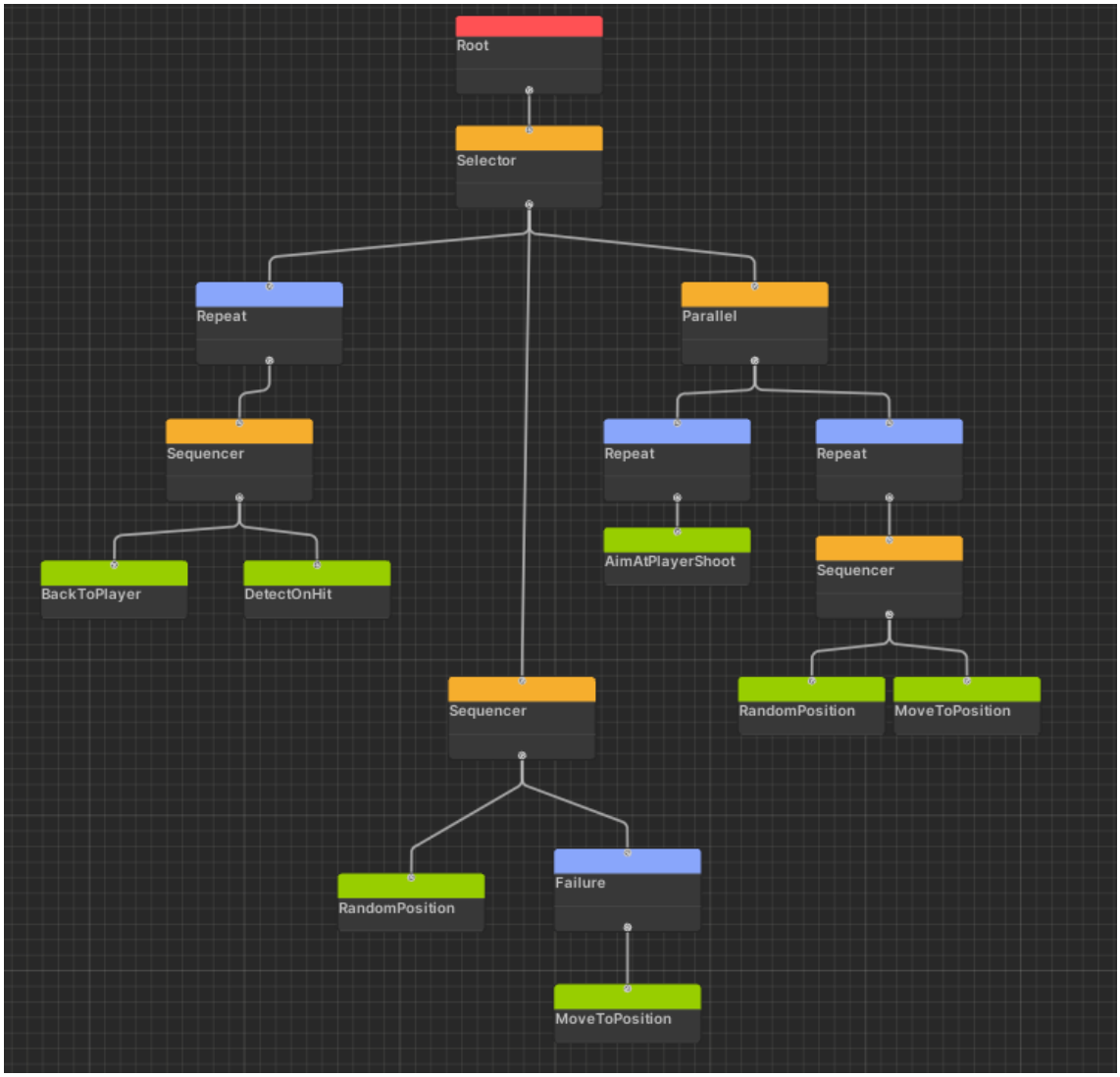

Figura 42 Arbre de comportament de Clone

8.5 Bales

Una altre característica que diferencia alguns enemics d'altres son les bales que utilitzen.

8.5.1 AllyBullet

Bala disparada per el jugador velocitat normal es destrueix al impactar amb qualsevol element, sí és un enemic també elimina a l'enemic.

Visualment resalten per una forma de gota de color blanc envoltat per un blau. Veure Figura 43.

Figura 43 Bales del jugador

8.5.2 EnemyBullet

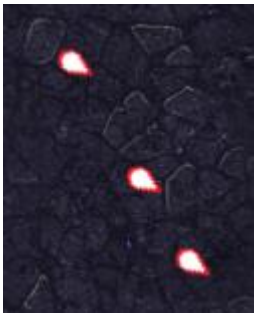

Figura 44 Bales enemigues

Bala enemiga, té el mateix comportament que les bales del jugador però no eliminen enemics, en canvi aquestes bales eniminen el jugador.

Visualment és igual a la bala del jugador però aquestes ressalten per estar envoltades per vermell en lloc del blau. Veure Figura 44.

8.5.3 BounceEnemyBullet

Bales enemigues que reboten un nombre limitat de vegades, funcionen igual que les bales enemigues, però no es destrueixen en els impactes fins que han superat el límit de rebots. El càlcul de la física de rebots s'ha fet amb materials de unity per tal que el motor faci els càlculs necessaris. Aquestes són les bales que disparen els Bounce. Visualment són de forma circular de color verd fluorescent i amb el centre transparent. Veure Figura 45.

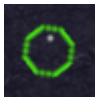

Figura 45 Bales enemigues que reboten

8.5.4 EnemySniperBullet

Aquestes bales tenen el mateix funcionament que les bales enemigues normals, però la velocitat és molt més elevada fent-les més difícils d'esquivar pel jugador. Aquestes són les bales que disparen els sniper Visualment tenen forma de coet de color blanc. Veure Figura 46.

Figura 46 Bales enemigues de sniper

9 Implementació i proves

En aquest apartat s'explicarà com s'han implementat les diferents parts del projecte i per tal d'arribar a l'objectiu final.

9.1 El desenvolupament a Unity

Abans d'explicar com s'ha desenvolupat el projecte amb Unity, hi ha varis conceptes importants a explicar per tal de facilitar la posterior descripció del projecte.

9.1.1 L'escena

L'escena del projecte consta dels objectes, és on podem veure l'apartat gràfic per tal de modificar manualment la posició dels elements. Un projecte no està limitat a una escena i normalment consten de varies escenes vinculades entre elles. Veure Figura 47.

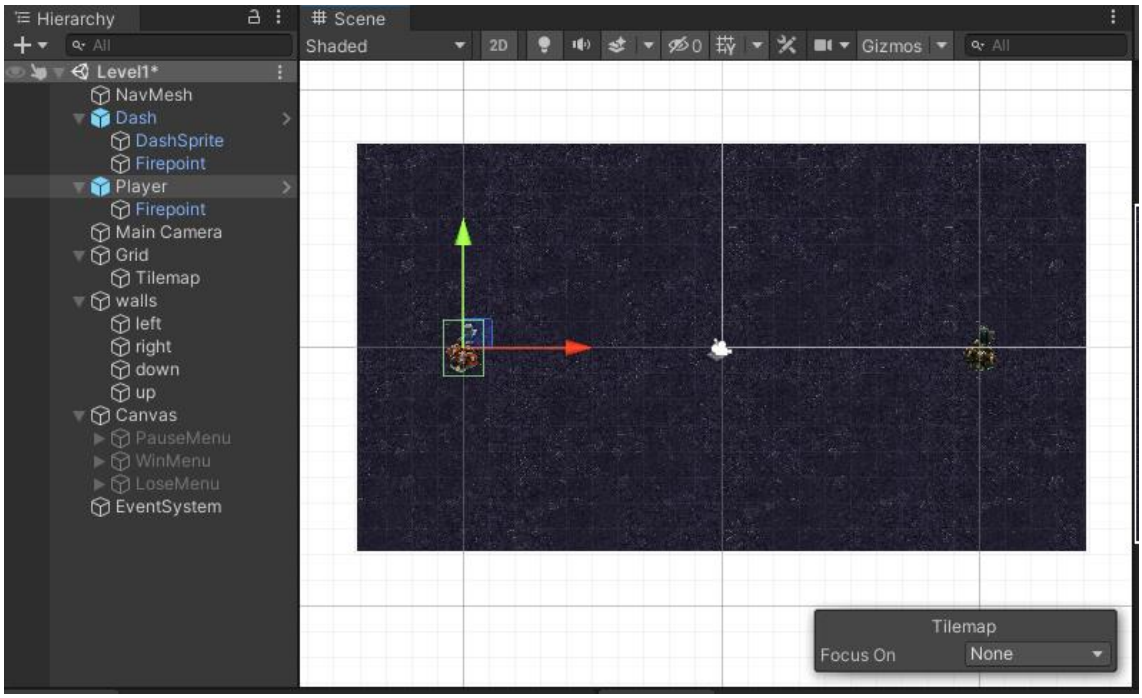

Figura 47 Interfície de l'escena i la llista d'objectes

9.1.2 Objectes

Els objectes són els elements que formen les escenes. Aquests elements poden ser des d'un element gràfic fins a un punt en l'espai. Els objectes poden ser pares d'altre objectes de manera que, si el pare realitza un moviment, el fill el segueix. Un exemple d'això pot ser el "firepoint" que hem afegit als personatges que disparen. El "firepoint" consisteix en un punt de l'espai respecte el personatge on hi ha el canó de l'arma. Aquest punt s'usa en la funció de disparar per marcar on es creen les bales que dispara. Per exemple, en la Figura 48, podem veure com està col·locat el "firepoint" per tal que es dispari des del canó de l'arma i no generar les bales des del centre del personatge.

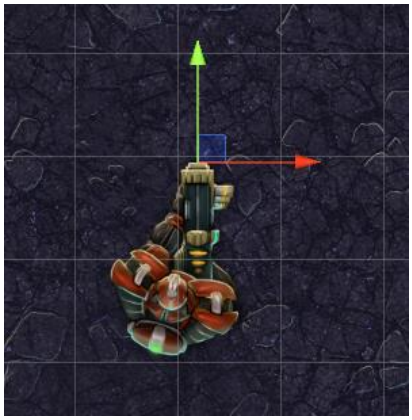

Figura 48 Firepoint del jugador vist amb l'editor

9.1.3 Atributs

Tots els objectes consten d'atributs que són els elements que el defineixen. Aquests atributs es poden modificar des de la interfície de Unity, però una forma molt comuna de modificar-los és utilitzant scripts per modificar-los mentre funciona el joc. D'aquests elements els més comuns són:

9.1.3.1 Transform

Aquest atribut ens defineix la posició a l'espai de l'element, així com la seva rotació i l'escala. Veure Figura 49.

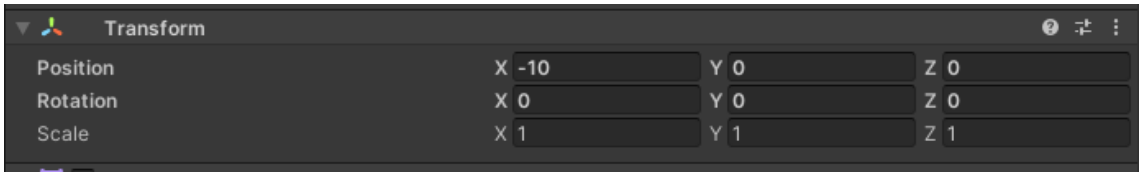

Figura 49 Atribut transform dins Unity

9.1.3.2 Sprite renderer

Aquest atribut ens permet mostrar els gràfics en els jocs 2D, és l'encarregat de carregar i mostrar l'element corresponent. Veure Figura 50.

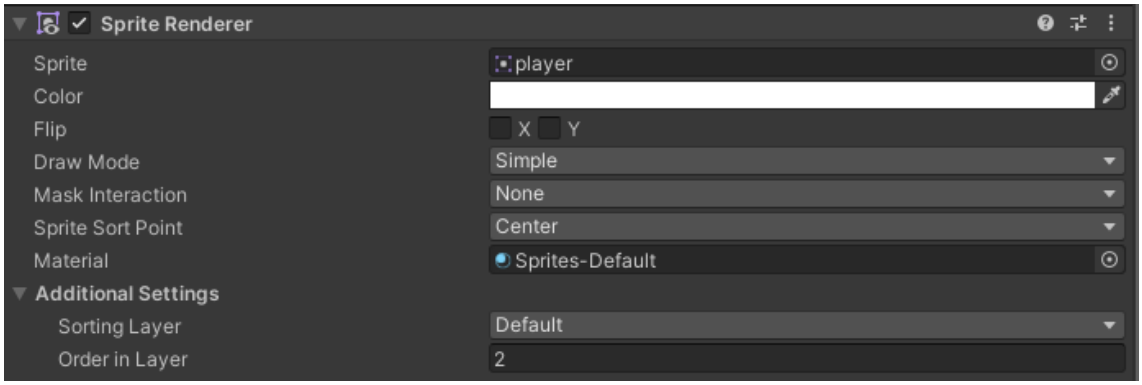

Figura 50 Atribut sprite renderer de Unity

9.1.3.3 Rigidbody

El "rigidbody" és un component dels objectes de Unity utilitzat per aplicar-li física. Aquest component ens permet aplicar-li forces a un objecte a través d'scripts, també porta incluides varies variables per fer la física més realistes amb la possibilitat d'asignar massa als objectes. Aquestes modificacions a la física d'un objecte es poden fer a través de l'interficie del motor de unity. Veure Figura 51.

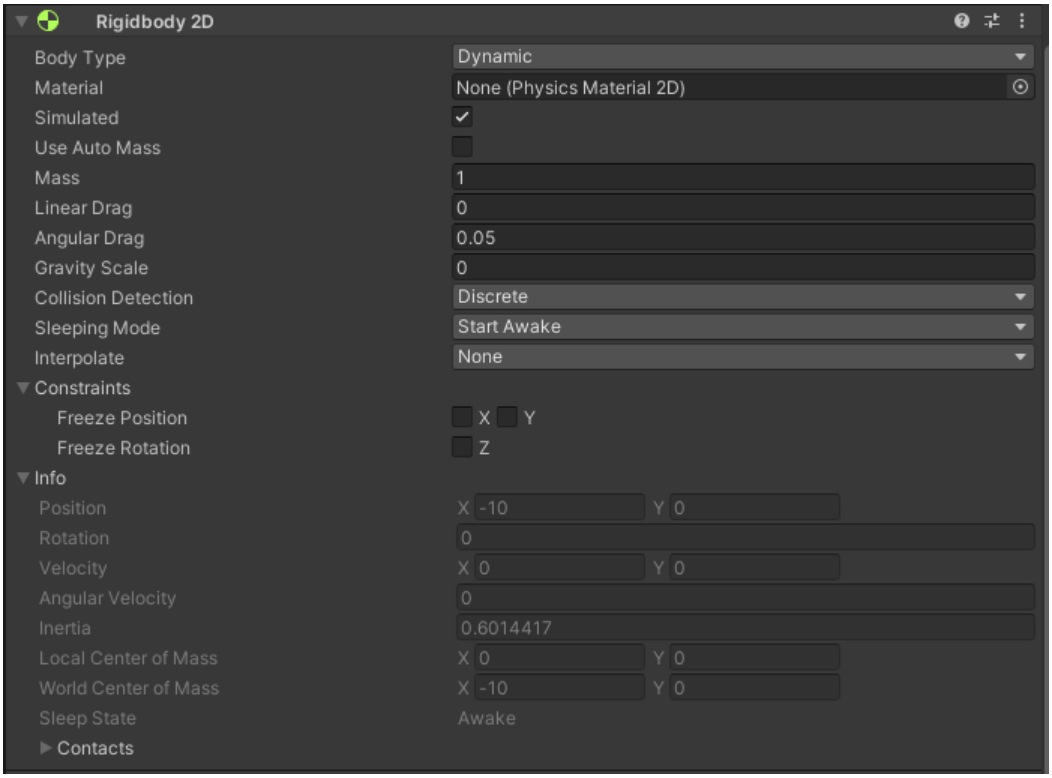

Figura 51 Interfície de edició d'un "rigidbody 2D

9.1.3.4 Box collider

El "box collider" servirà per les col·lisions entre objectes, per tal que no es superposin. Aquest atribut crea un àrea quadrada al voltant del element i ens permet detectar els elements que estan dins l'àrea. Una part interessant és la interacció entre dos "box colliders", ja que entre ells no permetran travessar-se. Per això es un element molt important tant pels personatges com per les parets, ja que si algun element esta sense el "box collider" no interactuarien entre ells i podrien travessar-se. A més un "box collider" ens permet crear interaccions entre objectes al xocar a través de funcions dins els scripts. Veure Figura 52.

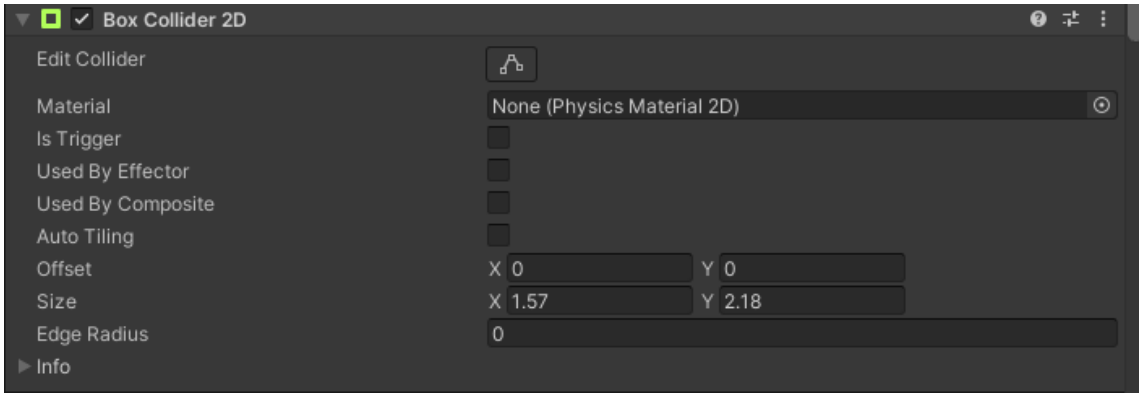

Figura 52 Interfície d'edició d'un "box collider 2D

9.1.3.5 Scripts com atributs

Els scripts de codi també formen part dels objectes del videojocs ja qué estan fets per interactuar amb l'objecte. Dins dels scripts es poden definir variables que es mostraran a l'interficie gràfica del motor per tal de poder modificar el seu valor de manera exterior al codi i facilitar el testeig i permetre fer modificacions al comportament de l'script si s'usa en diferents objectes. Veure Figura 53.

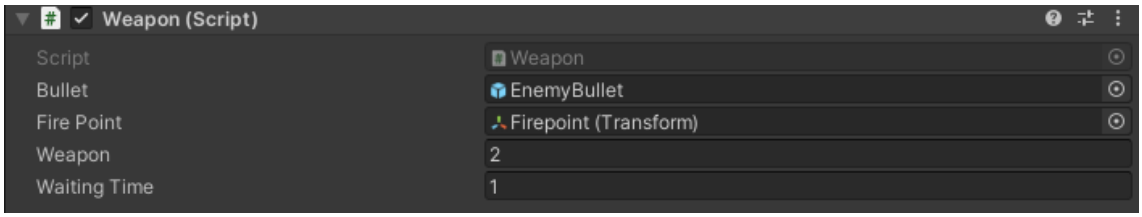

Figura 53 Script amb variables modificables dins l'interfície de Unity

9.1.4 Prefavs

Aquests elements son objectes ja creats que es poden guardar en el projecte per tal de crear copies de l'objecte. Un exemple serien les bales, que s'han creat com un prefav, ja que cada vegada que es dispara es crea una bala a l'escenari.

9.1.5 Scripts

Els scripts són les parts de codi que formen part del projecte. Aquests corresponen a una classe de C# i igual que una classe normal estan formats per variables i funcions.

9.1.5.1 Variables

Les variables funcionen igual que una classe de C#, però al tractar-se de scripts de Unity, tenim més varietat dintre els tipus de variables, de manera que ens permet utilitzar objectes o atributs d'aquests com a variables i utilitzar funcions específiques de Unity per modificar-los.

A més, com hem vist al punt 9.1.3.5, les variables de la classe es poden modificar a través de l'interfície de Unity. Per visualitzar les variables amb la interfície de Unity, es pot fer de dues maneres, fent les variables públiques, de manera que són accessibles per qualsevol classe, o

afegint [SerializeField] abans de declarar la variable privada, per tal d'indicar al codi que podem modificar el valor dins de l'editor. Veure Figura 56.

```
2 references
private Rigidbody2D RB;
1 reference
[SerializeField]private float moveSpeed;
```

Figura 54 Variables en un script de Unity

9.1.5.2 Funcions

Dintre la classe es poden crear funcions com si fos una classe de C#, però al tractar-se de Unity ens permet crear unes funcions específiques que s'executen automàticament mentre corre el joc. Les funcions més usades són:

9.1.5.2.1 Void Start()

Aquesta funció s'executa un cop al crear-se l'objecte a l'escenari, serveix principalment per definir l'estat inicial de l'objecte.

9.1.5.2.2 Void Update()

Aquesta funció s'executa a cada actualització del joc fins que l'objecte és eliminat. Principalment serveix per definir els comportaments de l'objecte durant l'execució.

9.1.5.2.3 Void FixedUpdate()

Aquesta funció té un comportament similar a la funció de Update, però realitza les actualitzacions de manera fixa, es a dir, sempre amb la mateixa freqüència. Aquest tipus d'update ens serveix principalment per treballar amb la física del videojoc ja que el "void Update()" ens dona diferents resultats segons la capacitat del PC ja que varia la freqüència d'actualització, en canvi amb aquest "Void FixedUpdate()" no dependrà del PC i sempre obtindrem els mateixos resultats independentment de les capacitats de l'ordinador.

9.1.5.2.4 Void OnTriggerEnter2D(Collider2D hit)

Aquesta funció la permet utilitzar el "collider". Bàsicament és una funció que s'executarà quan el "collider" detecti un xoc entre dos elements que tinguin "collider". Per tal d'interactuar amb l'altre objecte rebrem com a paràmetre el "collider" de l'altre objecte amb el que ha impactat.

9.1.6 Nodes dels arbres de comportament.

Per aquest projecte s'han desenvolupat nodes personalitzats per als arbres de comportament. Aquests nodes, igual que els scripts, consisteixen en classes de C#, però aquestes classes tenen tres mètodes principals.

9.1.6.1 Void OnStart()

Semblant al void Start(), però aquest s'executa quan el node rep senyal del pare.

9.1.6.2 State OnUpdate()

L'encarregat d'executar l'acció, és la funció principal i és obligatori. No es un Void com els altres sinó un State, ja que és l'encarregat de retornar al pare si l'execució ha estat èxit o fracàs.

9.1.6.3 Void OnStop()

S'executa un cop quan el node ha acabat el "OnUpdate".

9.2 Personatge principal

El personatge principal, dins del joc, pot realitzar dues accions principals, com ja s'ha explicat, aquetes consisteixen en el moviment per l'escenari i la capacitat de disparar.

S'ha desenvolupat un script per cada acció que pot fer el jugador de manera que es pot revisar el funcionament dels dos comportaments per separat.

Figura 55 Sprite del Jugador

9.2.1 Scripts del jugador

Clase PlayerController

```
 private Rigidbody2D RB;
    [SerializeField]private float moveSpeed;


    // Start is called before the first frame update
    void Start()
    {
        RB = gameObject.GetComponent<Rigidbody2D>();

    }

    // Update is called once per frame
    void FixedUpdate()
    {
        // Movement
            RB.velocity = new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical")) * moveSpeed;

    }
```

Amb la classe de PlayerController aprofitem el "rigidbody" del personatge per aplicar-li una força en la direcció de les tecles que apreta el jugador. Les funcions de "Input.GetAxisRaw("Horizontal")" i "Input.GetAxisRaw("Vertical")" ens serveixen per llegir l'entrada del jugador i aplcar una força a al personatge en la direcció que marca el jugador.

Classe PlayerWeapon

```
[SerializeField] private GameObject bullet;
    [SerializeField] private Transform firePoint;
    private Camera cam;
    // Start is called before the first frame update
    void Start()
    {
        cam = Camera.main;
    }

    // Update is called once per frame
    void Update()
    {
        if(!pauseMenu.isPaused){
            //Aim
            Vector3 mouse = Input.mousePosition;

                                            Vector3      screenPoint      =
cam.WorldToScreenPoint(transform.localPosition);

                Vector2  offset  =  new  Vector2(mouse.x  -  screenPoint.x,
mouse.y -screenPoint.y);

                    float   angle   =   Mathf.Atan2(offset.y,offset.x)   *
Mathf.Rad2Deg;

            transform.rotation = Quaternion.Euler(0f, 0f, angle -90);

            if(Input.GetMouseButtonDown(0) && !pauseMenu.isPaused){
                                    Instantiate(bullet,firePoint.position,
transform.rotation);
            }
        }
    }
```

Aquesta classe ens serveix per apuntar amb el jugador i disparar. Per tal d'apuntar s'han de fer varis càlculs. Primer es troba la posició el ratolí a la pantalla, després es fa el mateix per el personatge des del punt de la càmera, i així tenim els dos elements amb els mateixos eixos. Un cop fet això es troba el vector que uneix els punts i calcula l'angle. Finalment, per apuntar al ratolí es fa un rotació del personatge igual a l'angle - 90º. Restem els 90º degut a la posició de l'sprite inicial.

A l'hora de disparar es detecta si el botó del ratolí esta premut i si es així es crea una bala en el punt indicat en la direcció indicada.

9.3 Enemics

El funcionament dels enemics principalment està explicat a l'Apartat 8.4, per tant en aquest apartat ens centrarem en descriure el codi desenvolupat al respecte.

9.3.1 Scripts dels enemics

Classe Weapon

```
[SerializeField] private GameObject bullet;
    [SerializeField] private Transform firePoint;
    [SerializeField] private int weapon;
    private float timer = 0f;
    [SerializeField] private float waitingTime = 0f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        timer += Time.deltaTime;
    }

    public void shoot(){
        switch (weapon){
        case 0:
            break;
        case 1:
            if(timer > waitingTime){
                                    Instantiate(bullet,firePoint.position,
transform.rotation);
                timer = 0;
            }
            break;
        case 2:
        if(timer > waitingTime){
            Instantiate(bullet,firePoint.position, transform.rotation);
             Instantiate(bullet,firePoint.position, transform.rotation *
Quaternion.Euler(0,0,10));
             Instantiate(bullet,firePoint.position, transform.rotation *
Quaternion.Euler(0,0,-10));
            timer = 0;
        }
        break;
        default:
        break;
        }
```

```
    }
```

Tots els enemics menys el zombie comparteixen aquest script ja que és l'encarregat de fer-los disparar. Amb aquest script fem que al credar la funció "shoot()" l'enemic dispari. El tir funciona molt semblant al del jugador, però aquesta classe permet modificar les seves variables per modificar la forma en què dispara.

Els varis camps que utilitzem per variar la forma en que dispara són el temps entre bales, es a dir, quant ha d'esperar l'enemic per tornar a disparar, i el tipus d'arma, per tal de tenir totes les formes de disparar en un mateix codi, s'ha afegit una variable per seleccionar el tipus de tir. També es possible canviar el tipus de bala. Veure Figura 56

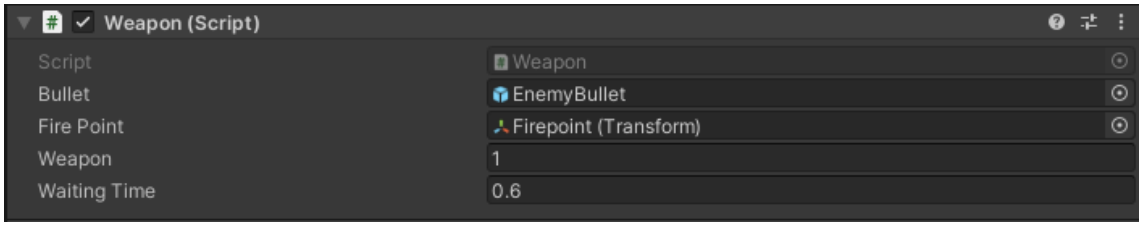

Figura 56 Atribut de l'script de weapon

Classe ZombieAtack

```
private void OnTriggerEnter2D(Collider2D hit)
    {
        if(hit.tag == "Player")
        Destroy(hit.gameObject);
    }
```

El Zombie, al no disparar, utilitza aquesta classe que detecta si toca un jugador i si es així l'elimina. Aquesta funció és molt similar a la que utilitzen les bales.

9.3.1.1 Script del Clone

El Clone té una característica, que el diferencia de tots els altres enemics es necessiten dos bales per eliminar-lo. Per fer-ho s'ha creat una capa superior que conté un "box collider" extra, de manera que així estem protegits de la primera bala. A l'hora de programar-ho, hem fet un script que detecti el primer impacte d'una bala del jugador i destrueixi el "box collider" exterior.

Classe BarrierHit

```
private BoxCollider2D col = null;
    // Start is called before the first frame update
    void Start()
    {
        col = gameObject.GetComponent<BoxCollider2D>();
    }

    private void OnTriggerEnter2D(Collider2D hit)
```

```
    {   if(hit.tag == "PlayerBullet"){
            Destroy(col);
        }
    }
```

9.3.2 Nodes dels arbres de comportament creats

Node AimAtPlayerShoot

```
private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        //Aim at player
        if(player !=null){
            Vector3 playerPosition = player.transform.position;

            Vector3 myPosition = context.transform.position;

                    Vector2  offset  =  new  Vector2(playerPosition.x  -
myPosition.x, playerPosition.y -myPosition.y);

                     float  angle  =  Mathf.Atan2(offset.y,offset.x)  *
Mathf.Rad2Deg;

             context.transform.rotation = Quaternion.Euler(0f, 0f, angle
-90);

                                              RaycastHit2D      hit      =
Physics2D.Raycast(context.transform.position, offset);
            if(hit.collider.CompareTag("Player"))
                context.gameObject.GetComponent<Weapon>().shoot();
        }
        else
        context.transform.rotation = Quaternion.Euler(0f, 0f, 90);
        return State.Success;
    }
```

Aquesta classe és semblant a la del jugador per apuntar i disparar, però en lloc d'apuntar al ratolí apunta al jugador. A més, abans de disparar, genera un raycast per detectar on apunta.

Això ens serveix per saber si tenim visio del jugador abans de disparar i així evitem que els enemics disparin a les parets si el jugador esta a cobert.

Node BackToPlayer

```
private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        //Aim back at player
        Vector3 playerPosition = player.transform.position;

        Vector3 myPosition = context.transform.position;

         Vector2 offset = new Vector2(playerPosition.x - myPosition.x,
playerPosition.y -myPosition.y);

        float angle = Mathf.Atan2(offset.y,offset.x) * Mathf.Rad2Deg;

           context.transform.rotation = Quaternion.Euler(0f, 0f, angle
+90);
        return State.Success;
    }
```

Aquest node és utilitzat a la primera fase del Clone i bàsicament és semblant al de disparar amb la variació que apunta en direcció contraria i no dispara.

Node BounceAimAndShoot

```
private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        //Aim at player
        Vector3 playerPosition = player.transform.position;

        Vector3 myPosition = context.transform.position;
```

```
          Vector2 offset = new Vector2(playerPosition.x - myPosition.x,
playerPosition.y -myPosition.y);

        float angle = Mathf.Atan2(offset.y,offset.x) * Mathf.Rad2Deg;

          context.transform.rotation = Quaternion.Euler(0f, 0f, angle -
90);

         RaycastHit2D hit = Physics2D.Raycast(context.transform.position,
offset);

        if(hit.collider.CompareTag("Player"))
            context.gameObject.GetComponent<Weapon>().shoot();
        else{
             context.transform.rotation = Quaternion.Euler(0f, 0f, angle
- Random.Range(0,180));
            context.gameObject.GetComponent<Weapon>().shoot();
        }
        return State.Success;

    }
```

Aquest node serveix per l'enemic Bounce, de manera que si no té visió del jugador dispararà en un angle aleatori (entre +90º i -90º) en la direcció del jugador.

Node DodgerAimShoot

```
 private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        //Aim at player
        Vector3 playerPosition = player.transform.position;

        Vector3 myPosition = context.transform.position;

          Vector2 offset = new Vector2(playerPosition.x - myPosition.x,
playerPosition.y -myPosition.y);

        float angle = Mathf.Atan2(offset.y,offset.x) * Mathf.Rad2Deg;

          context.transform.rotation = Quaternion.Euler(0f, 0f, angle -
```

```
90);

        RaycastHit2D hit = Physics2D.Raycast(context.transform.position,
offset);


        if(hit.collider.CompareTag("Player"))
            context.gameObject.GetComponent<Weapon>().shoot();
        else if(hit.collider.CompareTag("PlayerBullet"))
            return State.Failure;
        return State.Success;

    }
```

Aquesta és la variació del node de disparar del Dodger. Amb aquesta variació, al fer el raycast, si detectem una bala del jugador, ens retornarà fracàs per tal que passem als següents nodes de l'arbre que realitzaran un moviment del personatge.

Node CustomMoveToPosition

```
public float speed = 5;
    public float stoppingDistance = 0.1f;
    public bool updateRotation = true;
    public float acceleration = 40.0f;
    public float tolerance = 1.0f;

    protected override void OnStart() {
        context.agent.stoppingDistance = stoppingDistance;
        context.agent.speed = speed;
        context.agent.destination = blackboard.moveToPosition;
        context.agent.updateRotation = updateRotation;
        context.agent.acceleration = acceleration;
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        if (context.agent.pathPending) {
            return State.Running;
        }

        if (context.agent.remainingDistance < tolerance) {
            return State.Success;
        }

                            if      (context.agent.pathStatus      ==
UnityEngine.AI.NavMeshPathStatus.PathInvalid) {
            return State.Failure;
```

```
        }
            context.gameObject.GetComponent<Transform>().Rotate(90f,  0f,
0f);
        return State.Running;
    }
```

Aquest node és una modificació del node de moviment que venia per defecte amb l'asset dels arbres de comportament. S'ha realitzat la modificació perquè, originalment, estava adaptat al moviment en 3D i s'ha passat al 2D.

A l'hora de realitzar els moviments pel mapa ho realitzem amb dos nodes, un que tria el punt on volem anar i ho guarda en una variable blackboard que comparteix l'arbre, i el node que realitza el desplaçament,que es correspon al node.

Node PositionNear

```
[SerializeField]float maxMovement = 0f;
    protected override void OnStart() {
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        Vector3 myPosition = context.transform.position;
                    blackboard.moveToPosition.x    =    myPosition.x    +
Random.Range(maxMovement, -maxMovement);
                    blackboard.moveToPosition.y    =    myPosition.y    +
Random.Range(maxMovement, -maxMovement);
        return State.Success;

    }
```

Aquest node ens permet establir un rang màxim de moviment al voltant nostre i tria un punt aleatori dins aquest rang. Ens serveix per passar la posició al node que realitza el moviment.

Node PositionNearPlayer

```
[SerializeField]float offset = 0f;
    private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
```

```
        Vector3 playerPosition = player.transform.position;
                  blackboard.moveToPosition.x    =    playerPosition.x    +
Random.Range(offset, -offset);
                  blackboard.moveToPosition.y    =    playerPosition.y    +
Random.Range(offset, -offset);
        return State.Success;
    }
```

Aquest node compleix la mateixa funció que l'anterior, però els punts que ens dona són propers al jugador, i el rang establert és al voltant del jugador. Així aconseguim que els enemics persegueixin al jugador.

Node PositionDodge

```
[SerializeField]float maxMovement = 0f;
    private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        Vector3 playerPosition = player.transform.position;
        Vector3 myPosition = context.transform.position;
        Vector3 myNewPosition = player.transform.position;
        do{
            myNewPosition.x = myPosition.x + Random.Range(maxMovement, -
maxMovement);
            myNewPosition.y = myPosition.y + Random.Range(maxMovement, -
maxMovement);
             }while(myNewPosition  -  playerPosition  ==  myPosition  -
playerPosition && myNewPosition != myPosition);
                  blackboard.moveToPosition.x    =    myNewPosition.x    +
Random.Range(maxMovement, -maxMovement);
                  blackboard.moveToPosition.y    =    myNewPosition.y    +
Random.Range(maxMovement, -maxMovement);
        return State.Success;

    }
```

Aquest node, igual que els anteriors, també serveix per determinar una posició. Però, aquesta vegada intenta mantenir la distància amb el jugador. Aquesta classe l'utilitza el dodger ja que els moviments son per esquivar bales.

Node ZombiePlayerDetection

```
[SerializeField]float circleRad = 0;
    private GameObject player = null;
    protected override void OnStart() {
        if (player == null)
        player = GameObject.FindGameObjectWithTag("Player");
    }

    protected override void OnStop() {
    }

    protected override State OnUpdate() {
        Vector3 playerPosition = player.transform.position;
        Vector3 ownPosition = context.transform.position;
        if(Vector3.Distance(playerPosition,ownPosition) < circleRad){
            blackboard.moveToPosition.x = playerPosition.x;
            blackboard.moveToPosition.y = playerPosition.y;
        }
        else{
            blackboard.moveToPosition.x = ownPosition.x;
            blackboard.moveToPosition.y = ownPosition.y;
        }
        return State.Success;
    }
```

Amb aquest node, també obtenim una posició per fer un moviment, però per calcular la posició ho fem agafant la posició del jugador quan s'acosta més a prop d'un límit definit al personatge. Aquesta forma de trobar un punt l'usa el Zombie de manera que aquest perseguirà al jugador si entra en el seu radi de detecció.

Node DetectOnHit

```
private BoxCollider2D col = null;
    protected override void OnStart() {
        col = context.gameObject.GetComponent<BoxCollider2D>();
    }

    protected override State OnUpdate() {
        if(col==null){
            return State.Failure;
        }
        return State.Success;
    }
```

Aquesta classe és qui s'encarrega en el Clone de detectar si l'han disparat per començar la fase dos. La feina del node és revisar si encara té el collider exterior, de manera que un cop el perdi retornarà faracàs i començarà la següent fase.

9.4 Bales

Igual que el funcionament dels enemics, el funcionament de les bales principalment està explicat a l'apartat 8.5, per tant en aquest apartat ens centrarem en descriure el codi desenvolupat.

9.4.1 Scripts de les bales

Classe PlayerBulletController

```
[SerializeField] private float speed;
    private Rigidbody2D RB;
    // Start is called before the first frame update
    void Start()
    {
        RB = gameObject.GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        RB.velocity = transform.up * speed;
    }

    private void OnTriggerEnter2D(Collider2D hit)
    {   if(hit.tag != "Player"){
            if(hit.tag == "Enemy"){
                Destroy(hit.transform.parent.gameObject);
            }
            Destroy(gameObject);
        }
    }
```

Aquesta és la classe bàsica de les bales del jugador, consisteix en un element que es desplaça en una direcció fins a impactar, i si impacta amb un enemic l'elimina també.

Classe EnemyBulletController

```
    [SerializeField] private float speed;
    private Rigidbody2D RB;
    // Start is called before the first frame update
    void Start()
    {
        RB = gameObject.GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
```

```
        RB.velocity = transform.up * speed;
    }

    private void OnTriggerEnter2D(Collider2D hit)
    {   if(hit.tag != "Enemy" && hit.tag != "EnemyBullet"){
            if(hit.tag == "Player"){
                Destroy(hit.gameObject);
            }
            Destroy(gameObject);
        }
    }
```

Aquesta classe té el mateix funcionament que l'anterior, però esta modificada per tal que elimini al jugador i no a l'enemic. Les bales enemigues i les bales de sniper utilitzen aquest script, però les bales de sniper tenen la velocitat augmentada.

Classe BounceEnemyBulletController

```
[SerializeField] private float speed;
    [SerializeField] private int maxBounces;
    private int bounces = 0;
    private Rigidbody2D RB;
    private Vector3 lastVelocity;


    // Start is called before the first frame update
    void Start()
    {
        RB = gameObject.GetComponent<Rigidbody2D>();
        RB.velocity = transform.up * speed;
        bounces = 0;
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        lastVelocity = RB.velocity;
    }


    private void OnCollisionEnter2D(Collision2D col){
        bounces++;
        if(col.collider.tag == "Player"){
            Destroy(col.collider.gameObject);
            Destroy(gameObject);
        }
        if(bounces>maxBounces)Destroy(gameObject);

    }
```

Té el mateix comportament que l'anterior, però a l'impactar, en lloc de destruir-la, sumem 1 a rebots fins que arriba a la variable de maxBounces que és quan destruïm l'objecte. Aquest script és l'encarregat del comportament de les BounceEnemyBullets.

9.4.2 Rebots de les BounceEnemyBullets

Per fer que les bales poguessin rebotar amb les parets es van realitzar varies probes intentant realitzar els rebots amb un script que fes el càlcul de l'angle d'entrada i de sortida i la velocitat. Però finalment es va trobar una solució més fàcil afegint un material físic al "rigidbody" de la bala. Els materials físics permeten modificar la fricció i els rebots dels "rigidbodys", i es va crear un nou material sense fricció i amb rebot de manera que el motor feia tota la feina dels càlculs físics per calcular els rebots.

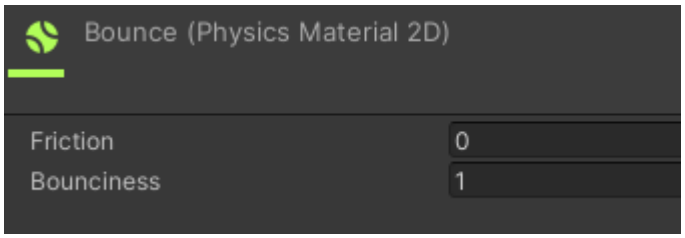

Figura 57 Material físic creat

9.5 Menús

9.5.1 Menú principal i de nivells

Aquests dos menús s'han creat dins la mateixa escena, veure Figura 58. Per crear aquests menús s'ha inserta un Canvas que permet crear elements davant la càmera. Dins del Canvas hem definit els dos menús i hem creat els botons corresponents a cadascun d'ells. Els canvis d'escena com començar un nivell o tancar l'aplicació, s'han realitzat a través d'un script dins de cada menú. Per al canvi de menú, s'ha realitzat a través dels botons que amaguen el menú principal i mostren el menú de nivell o a l'invers, respectivament.

Als botons de Unity es poden afegir funcionalitats a través dels atributs. Aquestes funcionalitats son respectives a un objectei el poden modificar o fins i tot executar funcions dels seus scripts. Veure Figura 59.

Figura 58 Escena del menú principal i de nivells

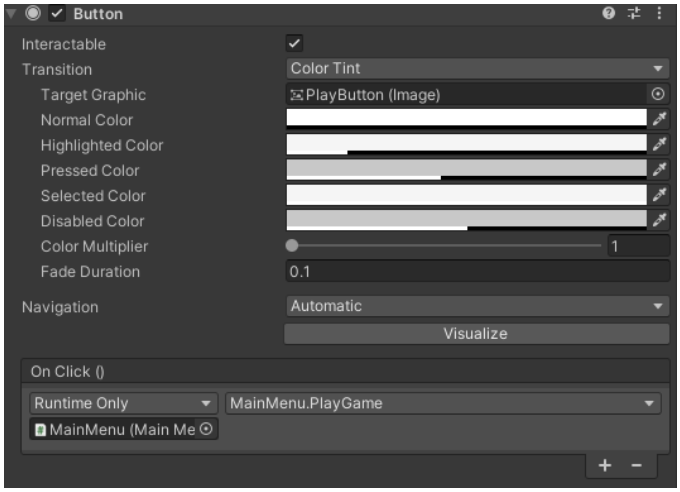

Figura 59 Atribut de botó de Unity

Classe MainMenu

```
public void PlayGame(){
          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
    }

    public void QuitGame(){
        Debug.Log("Quit!");
        Application.Quit();
    }
```

Aquesta classe és la que implementa el canvi a l'escena del primer nivell, i de tancar el joc del menú principal. A l'Apartat 9.6.2 s'explicarà com funcionen els canvis d'escena.

Classe LevelsMenu

```
public void PlayLevel(int index){
        SceneManager.LoadScene(index);
    }
```

Aquesta classe implementa el canvi d'escena del menú de nivells, el paràmetre d'índex ve donat per el botó que ha estat clicat.

9.5.2 Menú de pausa

El menú de pausa està desenvolupat dins d'un Canvas en els nivells. Aquest menú porta un script per detectar si s'ha premut el botó de pausa per mostrar-se i amagar-se, respectivament.

Classe PauseMenu

```
public static bool isPaused = false;

    [SerializeField]private GameObject pauseMenuUI;

    void Update(){
        if(Input.GetKeyDown(KeyCode.Escape)){
            if(isPaused){
                Resume();
            }
            else{
                Pause();
            }
        }
    }
    public void Resume(){
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
    }
    void Pause(){
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
    }
    public void MainMenu(){
        Time.timeScale = 1f;
        isPaused = false;
        SceneManager.LoadScene(0);
    }
    public void QuitGame(){
        Debug.Log("Quit!");
        Application.Quit();
    }
```

Per fer l'acció de pausar el joc quan el jugador prem el boto amb aquest script definim que l'escala del temps és 0 i per tant aquest no avança. Un cop premem el botó per sortir de pausa l'escala del temps passa a 1 i tot torna a moure's.

El menú de pausa, igual que el menú principal també compta amb les opcions de canviar d'escena i de sortir del joc.

9.6 Control de nivells i canvis d'escena

9.6.1 Control de nivells

El control de nivells ve donat per un script dins el mateix Canvas que el menú de pausa. Aquest script s'encarrega de controlar l'estat dels enemics i del jugador per mostrar la pantalla de victòria o derrota segons correspongui. A més, aquest script també té el control de reiniciar el nivell si s'ha perdut o continuar al següent si s'ha guanyat.

Classe LevelControl

```
 private int i = 0;
    [SerializeField]private int LastLvl = 9;
    [SerializeField]private GameObject WinUI;
    [SerializeField]private GameObject RetryUI;
    // Update is called once per frame
    void Update()
    {
        if(i<1){
                                            GameObject[]      enemies      =
GameObject.FindGameObjectsWithTag("Enemy");
            if (enemies.Length == 0){
                i++;
                Time.timeScale = 0f;
                WinUI.SetActive(true);
            }
                                             GameObject[]      player      =
GameObject.FindGameObjectsWithTag("Player");
            if (player.Length == 0){
                i++;
                Time.timeScale = 0f;
                RetryUI.SetActive(true);
            }
        }
    }

    public void Retry(){
        Time.timeScale = 1f;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void NextLevel(){
        Time.timeScale = 1f;
        if(SceneManager.GetActiveScene().buildIndex == LastLvl)
            SceneManager.LoadScene(0);
        else

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
    }
```

9.6.2 Canvis d'escena

Per realitzar els canvis d'escena, Unity proporciona un SceneManager amb les seves pròpies funcions, que hem estat utilitzant en aquests darrers scripts dels menús. Aquest SceneManager permet navegar per les escenes que s'ha definit en el menú per realitzar l'executable del videojoc, veure Figura 60. Dins d'aquest menú hem d'afegir les escenes que volguem i activar-les i ordenar-les de manera que podem definir que "nivell actual + 1 = nivell seguent" per tal de facilitar el codi de navegació pel joc.

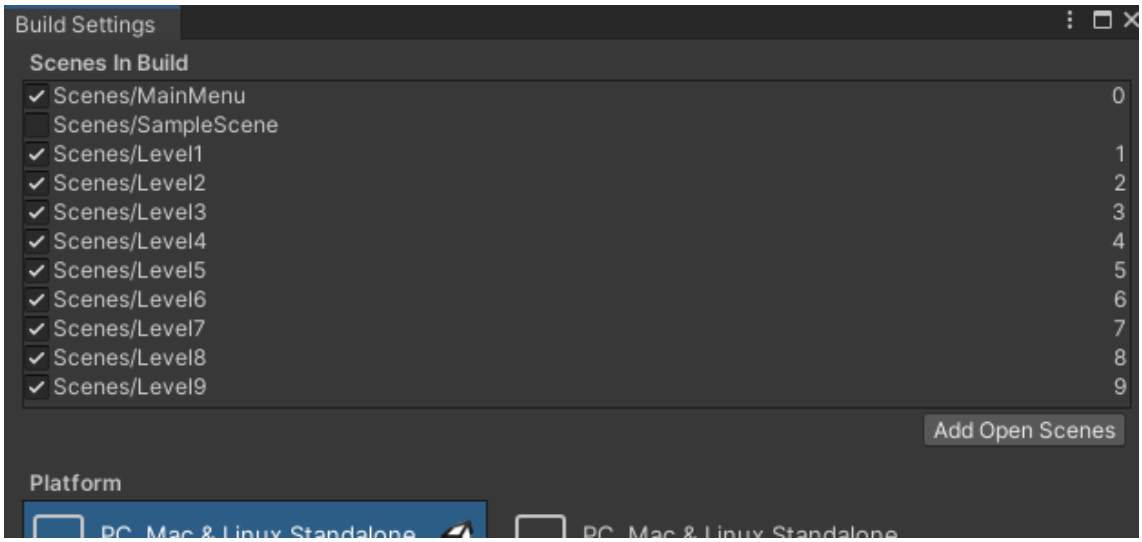

Figura 60 Menú per construir l'executable i on es porta el control de les escenes.

9.7 Problemes trobats durant el desenvolupament

Durant el desenvolupament ens hem trobat amb varis problemes els quals hem hagut de solucionar per poder avançar i aconseguir un producte final.

9.7.1 NavMesh i NavMeshAgents

Com hem mencionat al punt 7.5.2, vam fer ús d'un asset de NavMesh per tal que les intel·ligències artificials es moguessin per l'escenari. A part d'aquest problema, perquè els personatges es moguin per el NavMesh necessiten l'atribut de NavMeshAgent, que dona Unity. El NavMeshAgent és un atribut dissenyat per el NavMesh de Unity que treballa només amb 3D, per tant al realitzar moviments per l'espai. Aquest agent ens rota 90º els sprites en 2d sobre l'eix Z, de manera que es tornaven invisibles.

Per solucionar aquest problema, s'ha realitzat dues accions: fer que els enemics estiguin forçosament apuntant en una direcció, la majoria apunten al jugador, o porten un script per aplicar-los una rotació de 90º per a què es mostrin correctament.

9.7.2 Errors en la construcció de l'executable del joc

Aquest error ha provocat que no es pogués realitzar un executable del joc ja que estem utilitzant assets amb mètodes deprecated i, a l'hora de compilar tot el programa, ens dona error per culpa dels assets utilitzats.

Al ser problemes de compatibilitat en funcions del codi de tercers i tractar-se de elements més avançats de Unity, he preferit de moment no modificar l'asset, ja que es tracta de problemes massa complexes.

10 Resultats

10.1 Legislació i normativa vigent

El projecte desenvolupat no presenta cap problema des del punt de vista legislatiu.

La llei orgànica de protecció de dades de caràcter personal (LOPD) no s'ha tingut en compte, ja que el sistema en cap moment tracta cap tipus de dades relatives a l'usuari.

Per el tema de la seguretat, no hi ha cap requisit de control d'accés al programa, ja que es tracta d'una aplicació on els usuaris només tenen un rol.

En quant a la llei de serveis de la societat de la informació i comerç electrònic (LSSICE), el projecte no constitueix una activitat econòmica en cap dels sentits.

10.2 Captures de pantalla

En aquest apartat es mostren captures de pantalla realitzades amb l'execució del programa dins el motor Unity.

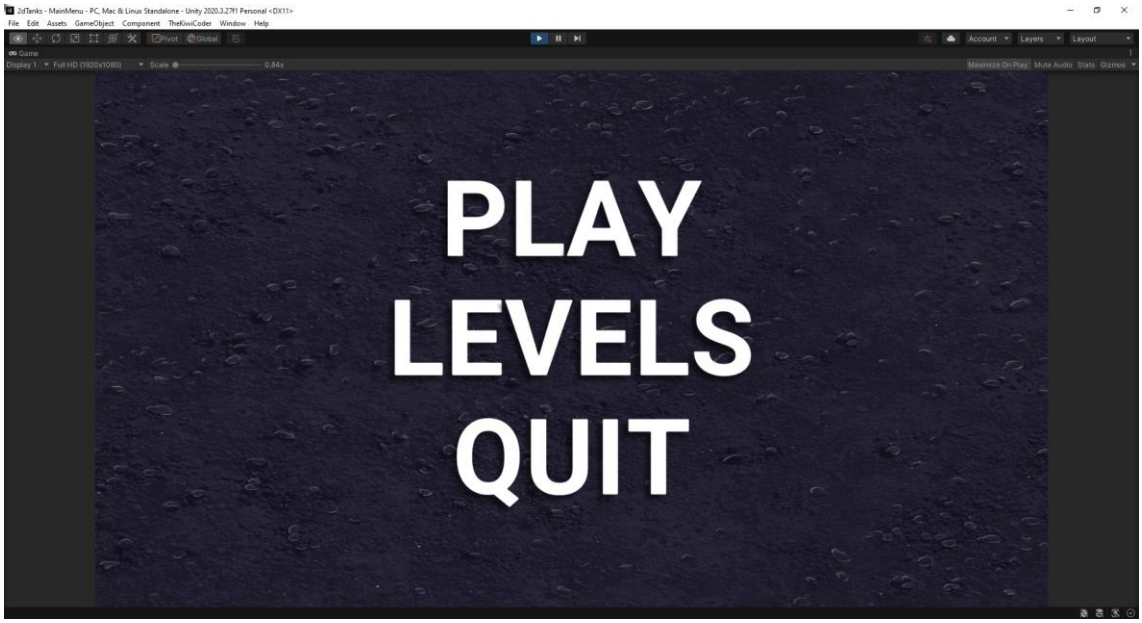

Figura 61 Captura de pantalla del menú principal

La Figura 61, correspon al menú principal, el primer menu el qual el jugador es trobaria al iniciar l'aplicació. A partir d'aquest menú pot començar una partida al nivell 1, seleccionar un nivell concret o sortir de l'aplicació.

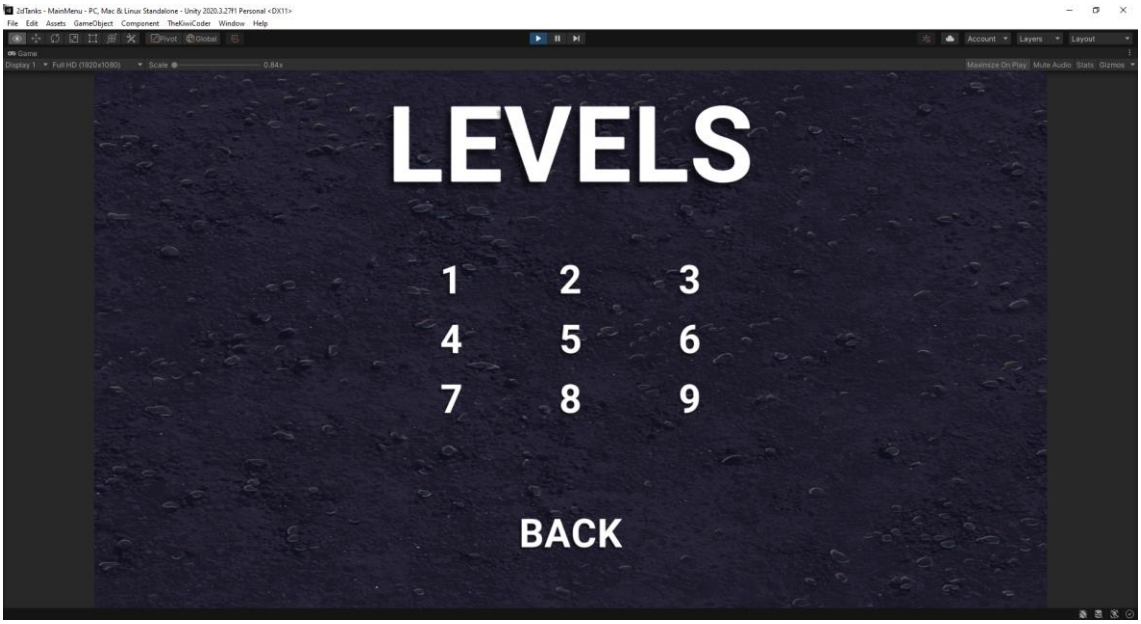

Figura 62 Captura de pantalla del menú de nivells

La Figura 62, correspon al menú on el jugador pot seleccionar un nivell per iniciar-lo o tornar al menú principal.

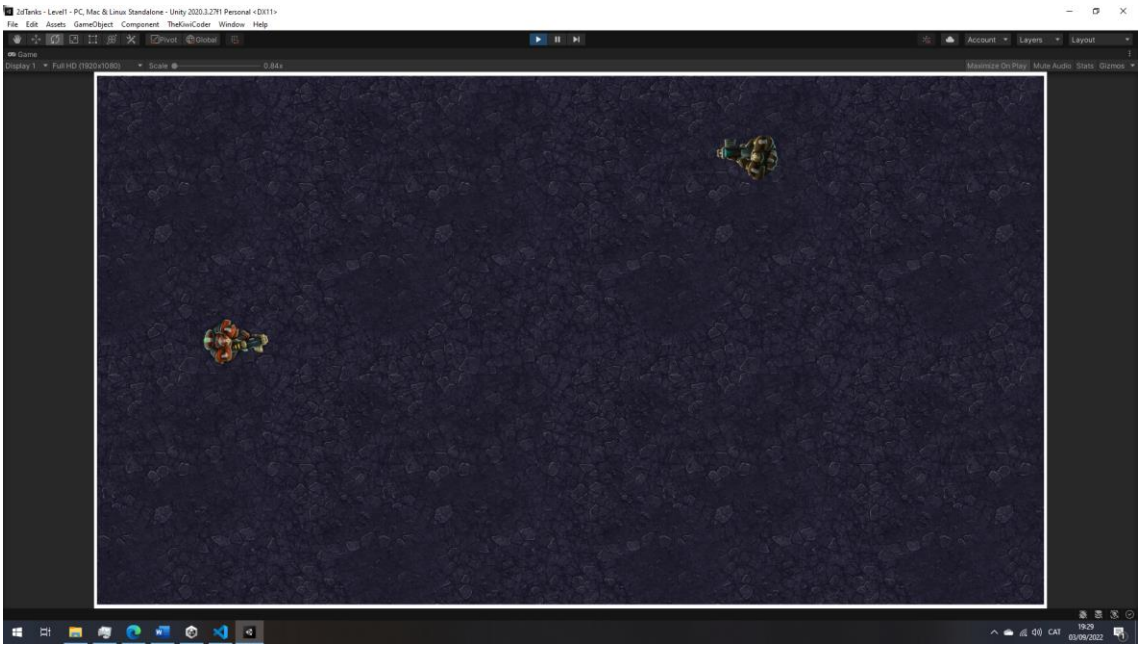

Figura 63 Nivell 1 iniciat

La Figura 63 mostra el nivell 1 iniciat on l'enemic ha començat a moure's.

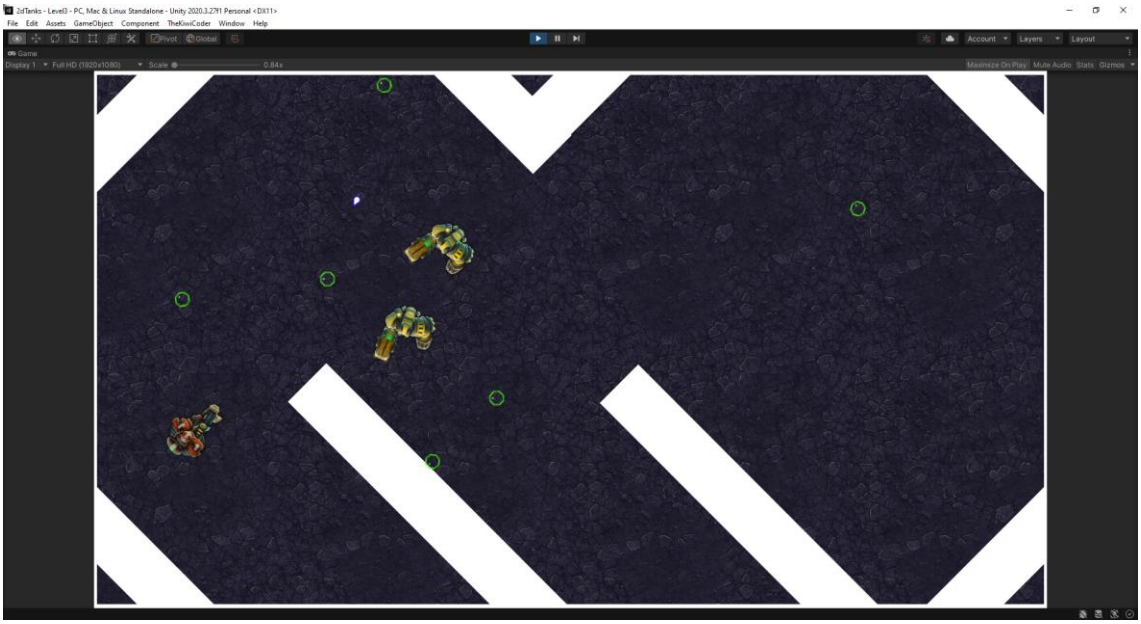

Figura 64 Nivell 3 iniciat

La Figura 64 ens mostra el nivell 3 que ja ha estat iniciat i els enemics estan atacant al jugador.

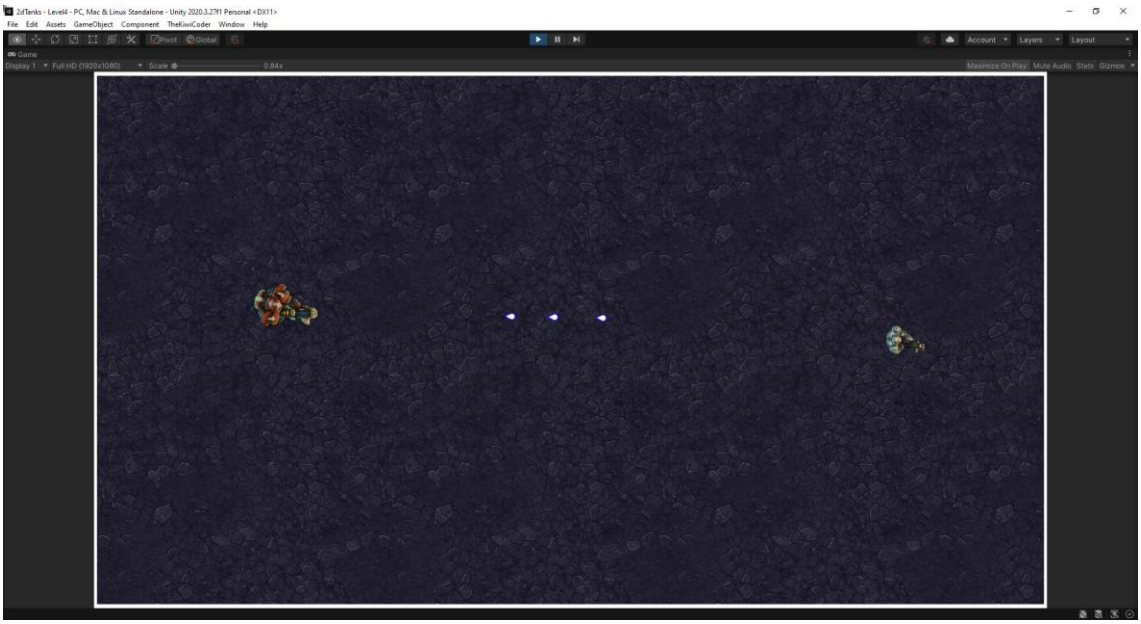

Figura 65 Nivell 4 iniciat

La Figura 65 ens mostra al jugador atacant a l'enemic del nivell 4.

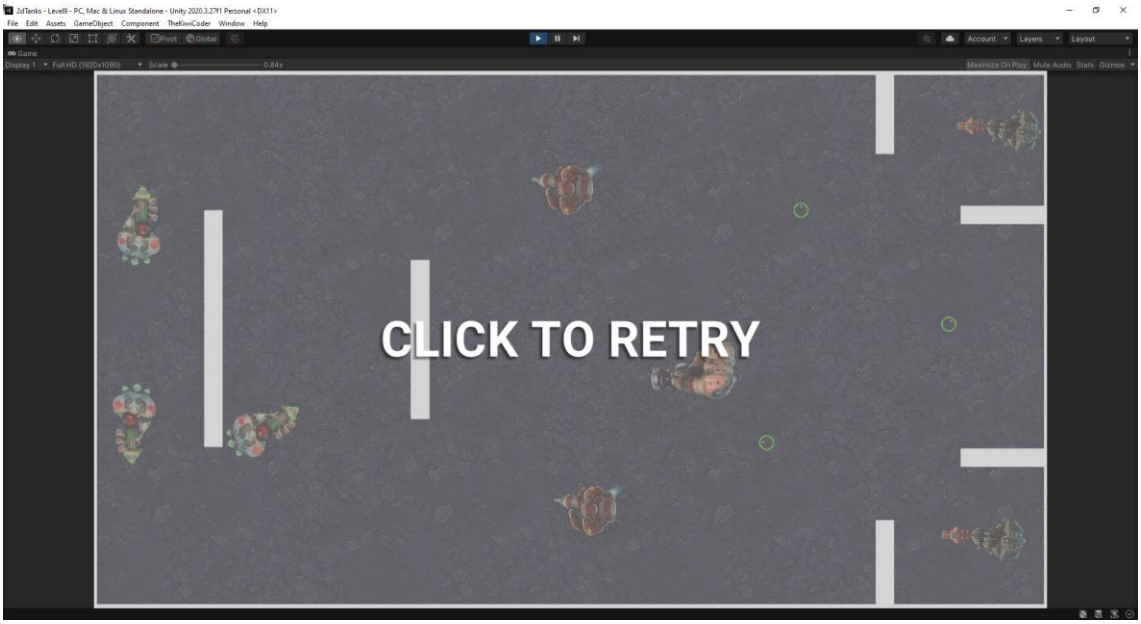

Figura 66 Nivell 9 amb jugador eliminat

A la Figura 66 podem veure la pantalla al ser el jugador eliminat. Aquesta consta d'una capa blanca semitransparent i en el fons podem veure l'estat del nivell quan el jugador és eliminat.

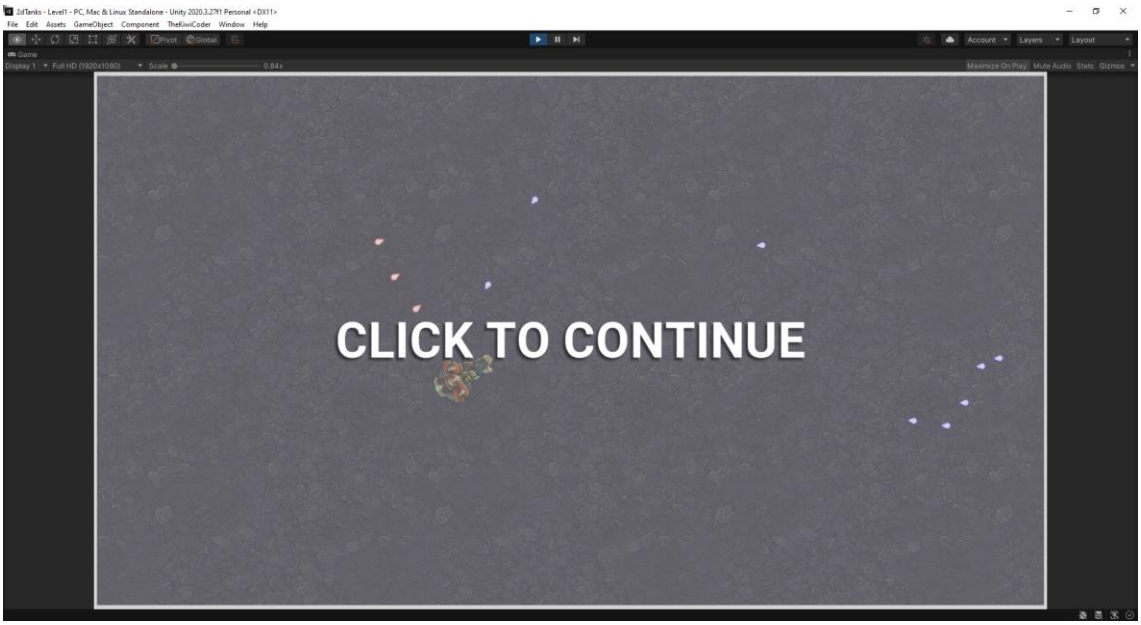

Figura 67 Nivell 1 on ha guanyat el jugador

A la Figura 67 podem veure la pantalla quan el jugador supera un nivell. És similar a la pantalla de derrota, però en lloc de reintentar permet seguir al següent nivell.

11 Conclusions

El desenvolupament d'aquest projecte ha estat la primera vegada que m'enfrontava amb el desenvolupament d'un videojoc i també el primer cop treballant amb intel·ligències artificials utilitzant arbres de comportament.

D'aquesta experiència m'emporto una sèrie de coses bones, coses dolentes i detalls importants si realitzo un altre projecte en aquest camp.

En quant a les coses bones, m'he iniciat en camp del desenvolupament de videojocs i alhora començat a treballar amb intel·ligències artificials dins d'aquests, que eren de les coses que més m'havien motivat a estudiar informàtica i amb aquest projecte he pogut aclarir que així és, i a més m'han descobert cap on enfocar la meva carrera d'informàtic.

També, he après la importància de la gestió del temps i la planificació en un projecte de gran volum de feina com seria un desenvolupament d'un videojoc, ja que una bona planificació i una bona fase disseny inicial pot evitar problemes en el futur desenvolupament.

En quant a coses dolentes, principalment m'emporto l'impossibilitat de crear executable del joc, que em va generar una mica d'insatisfacció al moment final, ja que vaig intentar solucionar l'error i no vaig poder. Tot i això, finalment, em quedo amb aquest detall important a l'hora del següent projecte ja que m'ha demostrat la importància d'utilitzar un bon motor que estigui preparat pel que s'ha de desenvolupar, ja que m'he trobat que Unity és un bon motor, fàcil d'entendre i fàcil per principiants. Dit això, mentre he desenvolupat, he sentit que tenia mancances en quant al seu apartat 2D, que potser amb un altre motor orientat específicament al 2D em podria haver servit més.

Finalment, sento que el desenvolupament d'un videojoc des de 0 ha estat un gran assoliment a la meva vida, i em sento amb ganes de començar un nou desenvolupament propi d'un videojoc, per endinsar-me més en aquest món.

12 Treball futur

Acabat el projecte es poden identificar varis aspectes a millorar o elements que no han acabat de quedar com ens agradaria. He realitzat un llistat de possibles tasques a realitzar amb aquest projecte:

- **Solucionar problemes de compatibilitat dels assets** per aconseguir crear l'executable del videojoc.

- **Creació de nou art propi**, per evitar usar sprites de tercers i aconseguir que fos un desenvolupament totalment propi.

- **Desenvolupament de nous enemics**, actualment hi ha un repertori petit d'enemics els quals ràpidament es fan repetitius, hi ha la possibilitat de augmentar la varietat per tal de fer el joc més atractiu.

- **Augmentar el nombre de nivells**, igual que la necessitat de nous enemics, es necessitaria un major nombre de nivells si volem obtenir un desenvolupament acabat i polit, com un joc que podria sortir al mercat.

- **Afegir so**, al tractar-se d'una demo per desenvolupar IA no s'ha treballat l'apartat sonor del joc ja que no era el propòsit del projecte.

- **Nous personatges per al jugador**, per tal de fer el joc possible de jugar varies vegades, una opció que tenim és la creació de diversos personatges de jugador amb diferents armes per tal que el jugador es pugui passar varies vegades el joc de diferents maneres.

13 Bibliografia

Epdata.(12 de maig de 2022). ¿Cómo evoluciona la industria del videojuego en España?

https://www.epdata.es/datos/videojuegos-espanoles-espana-datos-graficos/292

Wikipedia.(23 d'agost de 2022). Behavior tree (artificial intelligence, robotics and control)

https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control)

GitHub.(25 d'agost de 2022). Documentació NavMeshPlus

https://github.com/h8man/NavMeshPlus

Hmong.(23 d'agost de 2022). Árbol de comportamiento (inteligencia artificial, robótica y control)

https://hmong.es/wiki/Behavior_trees_(artificial_intelligence,_robotics_and_control)

Wikipedia.(21 d'agost de 2022). Unity (motor de videojuego)

https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

Wikipedia.(21 d'agost de 2022). Unreal Engine

https://es.wikipedia.org/wiki/Unreal_Engine

Wikipedia.(21 d'agost de 2022). Game Maker Studio

https://es.wikipedia.org/wiki/GameMaker_Studio

Wikipedia.(21 d'agost de 2022). Godot

https://es.wikipedia.org/wiki/Godot

OpenGameArt

https://opengameart.org/

Unity manual

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

14 Manual d'usuari

14.1 Instal·lació

Al no haver pogut generar un executable, per realitzar la instal·lació del joc es necessitarà el programari de Unity per importar-hi el projecte. Un cop obrim el projecte amb el motor, haurem d'obrir l'escena de Main Menu si no esta oberta, de dins la capeta de scenes des del navegador del projecte, veure Figura 68.

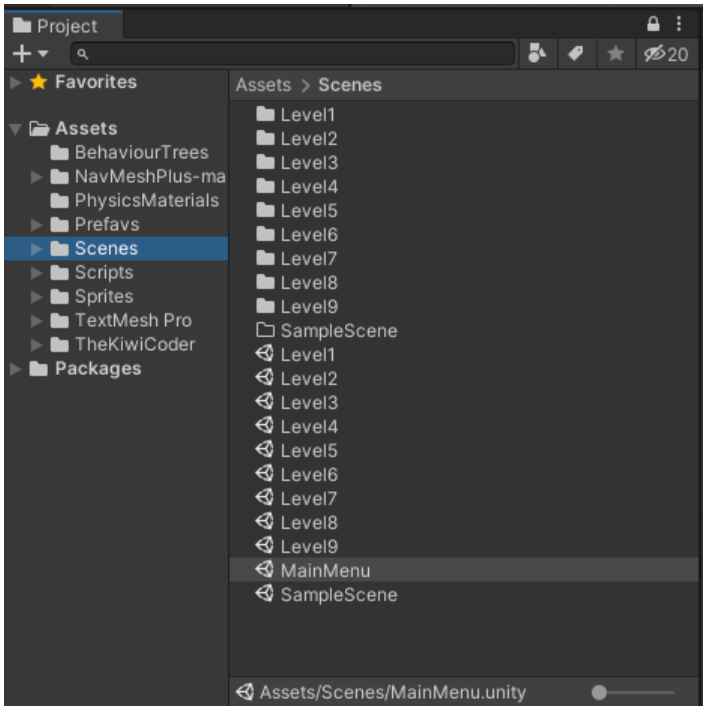

Figura 68 gestor de projecte de Unity

Un cop tinguem l'escena carregada, només necessitem fer click al play a la part superior de la interfície del motor per començar l'execució del joc. Es recomana tenir activat "Maximize on play" a la finestra del joc per augmentar la mida d'aquesta mentre s'executa. Veure Figura 69.

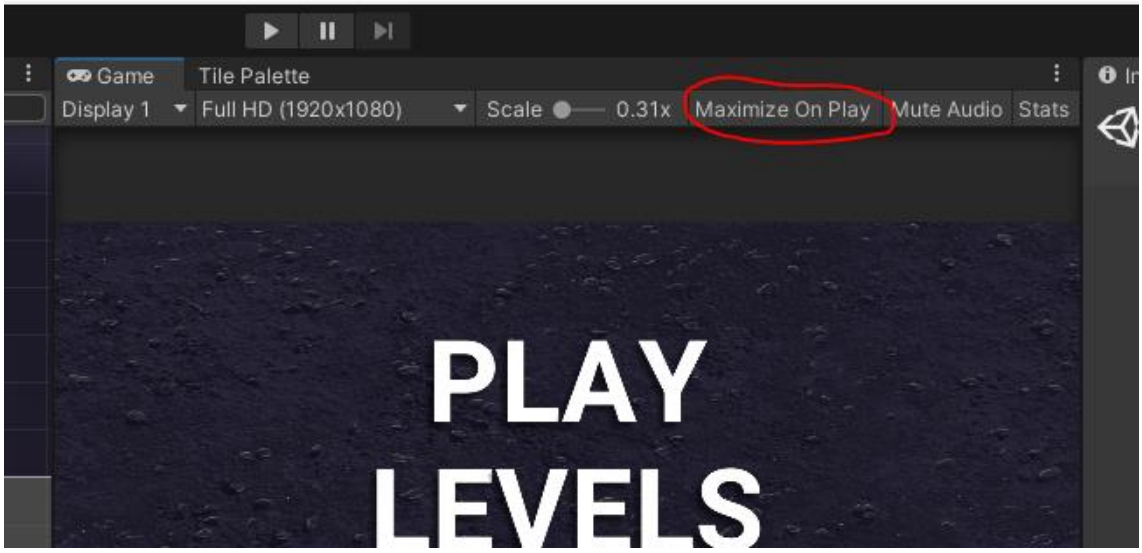

Figura 69 Finestra del joc on s'ha marcat "Maximize on play"

14.2 Objectiu del joc

L'objectiu d'aquest joc és superar els nivells eliminant a tots els enemics que hi ha en cadascun. Aquests nivells es van fent progressivament més difícils a mesura que es van superant.

14.3 Controls

Moviment: El moviment del joc es farà amb les tecles W,A,S,D que seran amunt , esquerra, avall i dreta respectivament. També es pot moure el jugador amb les fletxes del teclat. Aquest moviment serà respectivament a la càmera.

Apuntar i disparar: A l'hora d'apuntar, ho realitzarem amb el ratolí, el personatge sempre apuntarà a la posició del ratolí a la pantalla. Per disparar ho farem amb un clic esquerre del ratolí.