Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament d'un joc de supervivència en entorns reals

Document: Memòria Treball Final de Grau

Alumne: Marc Roger Sala

Tutor: Gustavo Patow
Departament: IMAE
Àrea: LSI

Convocatòria (mes/any): Gener 2022

ÍNDEX DE CONTINGUT

ÍNDEX DE FIGURES

1 INTRODUCCIÓ

Es coneix com a indústria dels videojocs, el sector econòmic involucrat en el desenvolupament, la distribució, màrqueting, venda de videojocs i també del maquinari necessari. Actualment és una de les indústries més importants del sector de l'entreteniment, tenint un gran impacte econòmic i social, havent superat altres indústries com la de la música i el cinema.

Un videojoc és un joc electrònic que és jugat en dispositius informàtics com pot ser un ordinador personal, una consola de videojocs o dispositius mòbils com poden ser les tauletes tàctils o els mòbils.

Es poden categoritzar a partir de la plataforma on es juguen però també es poden categoritzar, a causa de la gran varietat, a partir del gènere com podrien ser els jocs de plataformes, jocs de supervivència, els populars *First Person Shooters* (FPS), els *Massive Multiplayer Online* (MMO), jocs de rol, entre molts altres gèneres que la seva popularitat ha anat variant durant el temps i la tecnologia del moment. A la Figura 1 es pot veure com ha anat variant la popularitat de diferents gèneres al llarg del temps.

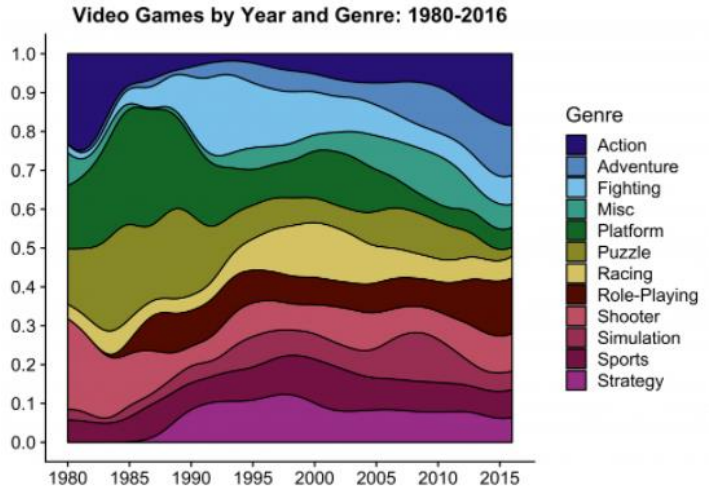

Figura 1. Generes més populars (1980-2016)

Tot i que diferents universitats i equips d'investigació van començar a desenvolupar videojocs senzills com el que és considerat el primer videojoc de la història *Spacewar!* fet l'any 1962, els primers jocs comercials van ser introduïts a la dècada dels 70. El primer exemple d'èxit comercial és quan Atari Computers va llançar *Computer Space* l'any 1971. El segon exemple i el més conegut, és el *Pong* que es va llançar el 1972.

El canvi i la millora constant de les tecnologies (com poden ser la creació de noves plataformes de joc, millora de les anteriors, millor capacitat de computació) fa que cada cop existeixin més opcions de cara al públic, i per tant, permeti a usuaris que anteriorment no estaven interessats en el sector, que hi entrin. La Figura 2 mostra el creixement dels usuaris des de l'any 2015 passant per l'actualitat i fent una predicció del nombre d'usuaris fins al 2023.

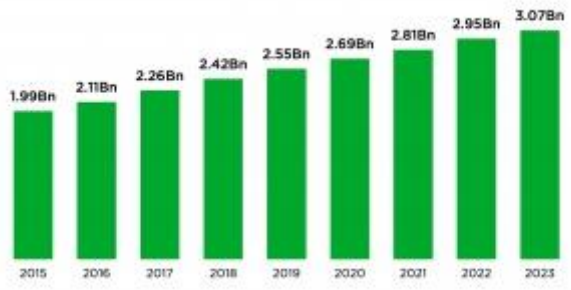

Figura 2. Nombre d'usuaris per anys

També s'ha de tenir en compte el fet de que a dia d'avui, cada cop hi ha més jugadors que juguen des de la seva consola portàtil, telèfon mòbil o tauleta ja que cada cop hi ha més gent que disposa al menys d'un d'aquests dipositius i cada vegada tenen més capacitat.

Després de veure aquest creixament i la varietat de plataformes que hi ha, la Figura 3 mostra el volum de negoci de l'indústria distingint els tres grans tipus de plataformes des del 2012 fins al 2021.

Figura 3. Volum de negoci entre el 2012 i 2021

Fins ara s'ha parlat de la situació global dels videojocs, si mirem la situació dels videojocs a Espanya, tal com es mostra a la Figura 4, es troba en el Top 10 de països amb més mercat l'any 2020.

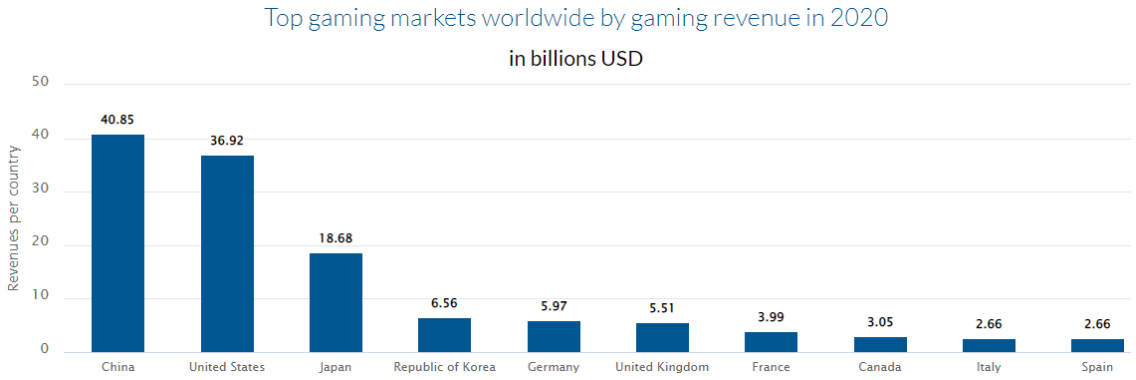

Figura 4. Mercats més grans l'any 2020

1.1 MOTIVACIÓ I OBJECTIUS PERSONALS

Personalment soc molt fanàtic dels jocs de supervivència, però sempre he tingut la curiositat de "com seria un joc en el que puc anar al meu carrer o visitar indrets coneguts?". A més a més des que era un nen, gràcies als meus pares, he estat envoltat de videojocs i això també ha fet que el meu interès cap als videojocs sigui molt gran. El primer joc que recordo veure jugar al meu pare i que m'ensenyés a jugar, va ser l'*Alex Kidd in Miracle World* un joc de la Sega. Veure Figura 5.

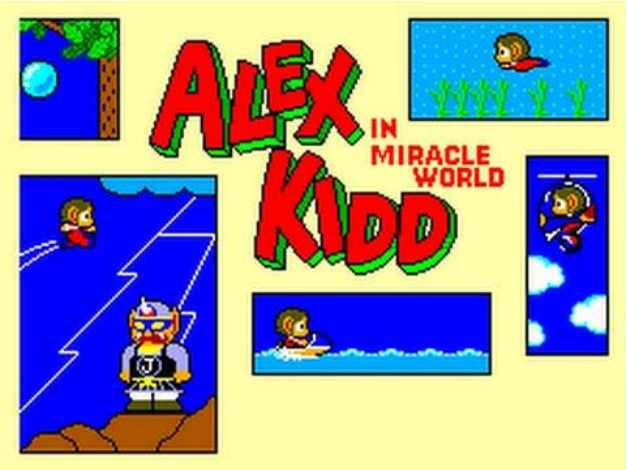

Figura 5. Inici Alex Kidd in Miracle World

De fet, a l'hora de triar carrera em vaig plantejar fer el Grau de Disseny i Desenvolupament de Videojocs, però al moment de decidir, em vaig decantar per la informàtica així veuria molt més del món de la informàtica que també és un àmbit que m'interessa molt, em permetria veure més parts de la informàtica i si a l'acabar GEINF encara m'interessa, buscar un master en videojocs o fer el grau de videojocs.

Pel TFG, al principi el motiu principal era posar-me a prova a mi mateix creant un programa que a partir de una imatge en vista aèria i d'una base de dades pública amb la densitat de població de diferents pobles i ciutats, crear-ne un model 3D per al final poder desenvolupar un videojoc. Al principi aquesta idea era un projecte personal ja que, com he dit, em volia posa a prova, però quan havia de pensar un tema per fer el TFG, vaig pensar que la part de crear el model 3D podria ser un projecte interessant. Després de parlar amb diferents professors, em van recomanar parlar amb el tutor i em va comentar que el que volia fer de crear el model 3D, en part ja existia una eina creada per Google, i em va proposar que, ja que el meu objectiu també era fer un videojoc, si el volia fer com a TFG. Tot i que el meu objectiu canviés, com que aprendre a desenvolupar videojocs també és un objectiu personal, vaig acceptar.

1.2 OBJECTIUS I PROPÒSITS DEL PROJECTE

Els objectius que ens hem plantejat aconseguir fent aquest projecte són:

- Aprendre i explorar les funcionalitats que disposa el motor de videojocs Unity.
- Aprendre i documentar-me sobre l'API de Unity
- Aprendre el funcionament de l'eina Google Maps Platform (GMP)
- Utilitzar Unity i GMP conjuntament
- Familiaritzar-me amb el llenguatge de programació C# utilitzat en les dues eines
- Modificació dels scripts de GMP per poder aprofitar-ne les dades
- Dissenyar i implementar un HUD (*Heads-Up Display*) per mostrar informació a l'usuari
- Dissenyar i implementar una GUI (*Graphical User Interface*) per a menús on l'usuari pot interaccionar

- Aprendre a crear i/o modificar animacions per utilitzar-les en el videojoc

1.3 BREU INTRODUCCIÓ ALS JOCS DE SUPERVIVÈNCIA

Als jocs de supervivència, com el nom indica, l'objectiu principal és sobreviure en un cert escenari. Però com que hi pot haver tanta varietat d'escenaris, la supervivència es tracta més com un subgènere complementari del gènere principal. Això fa que hi hagi una gran quantitat de jocs de supervivència de estils molt diferents com podrien ser:

- Valheim. Un joc de supervivència de món obert ambientat en època vikinga on el jugador lluita per entrar al Valhalla. Figura 6.

Figura 6. Captura de pantalla de Valheim

- Oxygen Not Included. Un supervivència ajuntat amb estratègia con el jugador ha d'aconseguir que els personatges del joc sobrevisquin en diferents asteroides. Veure Figura 7.

Figura 7. Captura de pantalla de Oxygen Not Included

- Minecraft. Un supervivència de món obert. El jugador té total llibertat per fer el que vulgui amb els recursos que ofereix el joc. Veure Figura 8.

Figura 8. Portada de Minecraft

- This War of Mine. Un supervivència 2D de scroll lateral on el jugador ha de aconseguir que un grup de civils sobrevisqui en un país en guerra. Veure Figura 9.

Figura 9. Captura de pantalla This War of Mine

- Subnautica. Un supervivència de món obert on la seva jugabilitat està ambientada al fons del mar. Veure Figura 10.

Figura 10. Captura de pantalla Subnautica

1.4 ESTRUCTURA DEL DOCUMENT

- **1. Introducció:**
  En aquest capítol s'explica el perquè del projecte, les motivacions que m'han portat a fer aquest projecte, quins són els objectius proposats i com s'ha organitzat el desenvolupament per portar-lo a terme i organització de la memòria.

- **2. Estudi de viabilitat**
  En aquest apartat es justifiquen els paràmetres que es necessiten per desenvolupar el projecte

- **3. Metodologia**
  Aquest apartat explica la raó de la metodologia que s'ha utilitzat durant el desenvolupament del projecte i el perquè s'ha utilitzat.

- **4. Planificació**
  Aquest capítol explica l'estratègia que s'ha seguit per assolir els objectius que s'han plantejat.

- **5. Marc de treball i conceptes previs**
  En aquest capítol es mostrarà una descripció dels aspectes del desenvolupament del projecte. També s'expliquen les decisions que s'han pres durant les diferents etapes del desenvolupament del projecte i els conceptes que s'han hagut d'aprendre.

- **6. Requisits del sistema**
  Aquí s'especificaran els requeriments funcionals i no funcionals del programari que recullen els objectius del projecte i les funcionalitats que es volen aconseguir.

- **7. Estudi i decisions**
  En aquest apartat s'especifiquen totes les eines utilitzades en el desenvolupament del projecte fent una explicació de cada una d'elles, característiques, ús en el projecte, importància en el desenvolupament i raó de perquè s'han triat.

- **8. Anàlisi i disseny del sistema**
  En aquesta secció s'explica la investigació realitzada per solucionar els problemes que s'han hagut de resoldre durant el desenvolupament del projecte i requeriments del projecte.
  A més de una explicació de les parts més importants del projecte, diagrames i fitxes de casos d'ús, diagrames d'activitats i el diagrama de classes del projecte.

- **9. Implementació i proves**
  En aquesta part es mostra el procés de com s'ha anat construint el videojoc, classes i mètodes implementats que siguin importants pel funcionament del videojoc.

- **10. Resultats**
  En aquest capítol es mostren les proves d'execució del videojoc, imatges dins del joc, interfícies, menús i entorns disponibles.

- **11. Conclusions**
  En aquesta part s'exposaran les conclusions que s'han tret del projecte en l'estat que es troba al moment de l'entrega

- **12. Treball futur**
  Aquí es descriuran totes les parts del projecte que es podrien millorar, que es podrien ampliar després de l'entrega del projecte.

- **13. Bibliografia**
  Apartat que conté les referències utilitzades durant el desenvolupament del projecte.

- **14. Manual d'usuari**
  Aquest últim capítol que conté les explicacions necessàries per a l'ús del videojoc.

2 ESTUDI DE VIABILITAT

Per al desenvolupament del projecte, no són necessaris una gran quantitat de recursos i amb ordinadors personals es podria dur a terme sense problemes.

El programari que s'ha utilitzat al llarg del projecte ha estat:

- Windows 10
- Motor de videojocs Unity
- Visual Studio Code
- Blender

I respecte al hardware, s'han utilitzat dos ordinadors diferents. Especificacions del primer (ordinador de sobretaula):

- Processador: AMD Ryzen 7 5800X 3.8GHz
- Memòria RAM: 32GB
- Targeta gràfica: GeForce GTX 970 4GB

Especificacions del segon (ordinador portàtil MSI Alpha 15):

- Processador: AMD Ryzen 7 3750H 2.3GHz
- Memòria RAM: 16GB
- Targeta gràfica: Raedon RX 5500M

Al principi del desenvolupament es disposava d'un altre ordinador de sobretaula amb un Intel i7-4790K, 16GB de RAM i la mateixa targeta gràfica.

També s'ha de tenir en compte que tot el programari que s'ha utilitzat és gratuït (menys el sistema operatiu que, tot i ser de pagament, ja se'n disposava).

2.1 PRESSUPOSTOS INICIALS

Per desenvolupar aquest projecte, no es necessita cap inversió inicial fora del hardware que es necessita però que ja se'n disposa. Llavors en el cas de que el joc es volgués publicar en alguna plataforma com podria ser Steam Direct en el que es paguen l'equivalent a 100USD, i si es publiqués el joc, no és necessària una llicència de Unity si els ingressos anuals no superen l'equivalent a 100000USD.

2.2 RECURSOS HUMANS

Tots els videojocs estan construïts sobre diferents bases:

- Gràfics
- So
- Programació
- Narrativa

Amb aquestes bases es forma un videojoc i l'ideal actualment és que un videojoc tingui totes parts (podríem posar casos com els jocs de texts on molts no tenen ni so ni gràfics, però és un tipus de joc que ja gairebé no es produeix). Per aquest motiu, si haguéssim de formar un equip per desenvolupar aquest videojocs, necessitaríem:

- Programador
- Dissenyador 3D
- Compositor
- Cap de projecte
- Tester
- Productor

En el cas d'aquest projecte, essent jo l'únic membre de l'equip i disposant de coneixements mínims o nuls en l'àmbit de disseny i composició, en comptes de crear aquests elements jo mateix, m'he decantat per obtenir aquests elements en pàgines web que els ofereixen de manera gratuïta i lliure.

2.3 AVALUACIÓ DE COSTOS I MITJANS

Per desenvolupar un videojoc, s'han de tenir en compte diferents tipus de viabilitat com la viabilitat econòmica, la viabilitat tècnica/tecnològica i la viabilitat legal.

2.3.1 Estudi de viabilitat tecnològica

L'estudi de la viabilitat tecnològica ens permet definir les necessitats de hardware per desenvolupar el projecte. També s'ha de tenir en compte que el hardware es desgasta i utilitzar hardware menys capaç pot resultar en un desgast i temps de desenvolupament superior que pugui acabar suposant el trencament del hardware.

Per el que respecte al nostre projecte, tant a l'hora de desenvolupar el videojoc com a l'hora de testejar-lo i jugar-lo, es necessiten uns requeriments de hardware mínims que són equivalents als requeriments mínims per utilitzar el motor Unity. Per aquest motiu i pel fet de que ja disposem de hardware capaç de dur a terme el projecte, el projecte és viable tecnològicament.

2.3.2 Estudi de viabilitat econòmica

La viabilitat econòmica es divideix en dos parts, la maquinaria i el cost dels recursos humans.

2.3.2.1 Cost de la maquinaria

Els costos de maquinària venen deguts al software i hardware que s'utilitzi. En el cas d'aquest projecte, de la manera que s'ha plantejat amb ubicacions estàtiques, es pot utilitzar la versió gratuïta però limitada de GMP. Unity, com s'ha comentat anteriorment, si es publica el joc i els ingressos són superiors a l'equivalent a 100000USD es requereix una llicència, però en aquest

cas no fa falta. Visual Studio Code i Blender també són gratuïts. Per tant, els costos que ens queden són:

Element	Quantitat	Cost unitat	Cost total
Llicència Windows 10	3	145,00€	435,00€
Ordinador sobretaula 1	1	1200,00€	1200,00€
Ordinador sobretaula 2	1	1100,00€	1100,00€
Portàtil MSI Alpha 15	1	1100,00€	1100,00€
Perifèrics varis	1	500,00€	500,00€
Total			4335,00€

Taula 1. Cost de la maquinaria

2.3.2.2 Cost recursos humans

Suposarem que a l'equip en comptes de només formar-ne part jo, pel que disposa de l'equip complert, amb un treballador per cada àmbit mencionat anteriorment, l'equip estaria format per 6 treballadors. A la següent llista hi ha cada treballador amb el sou mig de la seva professió a Espanya i treballant 40 hores setmanals. Al ser un equip petit, suposarem que el Dissenyador s'ocupa dels estudis i investigació de l'estil del joc, modelatge 3D i animacions.

- Programador videojocs: 16€/hora
- Dissenyador: 20€/hora
- Compositor: 14€/hora
- Cap de projecte: 25€/hora
- Tester: 10€/hora
- Productor: 15€/hora

Seguidament es mostra la taula amb les tasques que hauria de realitzar cada treballador amb les hores estimades que tardaria i el cost que suposaria.

Tasca	Treballador	Hores	Cost en €
Disseny del concepte del joc	Cap del Projecte	20	500
Investigació mercat dels videojocs	Productor	10	150
Estudis i investigació ambientació	Dissenyador	30	600
Estudi del motor Unity	Dissenyador	50	1000
Disseny IA	Dissenyador	20	400
Disseny sistemes de control	Dissenyador	15	300
Disseny interaccions	Dissenyador	5	100
Implementació IA	Programador	20	320
Implementació sistemes de control	Programador	10	160
Implementació interaccions	Programador	20	320
Estudi GMP	Programador	50	800
Implementació sistema de recollida de dades del GMP	Programador	40	640
Generar entorn a partir de les dades recollides	Programador	80	1280
Sistema canvi d'ubicacions	Dissenyador	20	400
Composició banda sonora	Compositor	40	560
Proves i testos	Tester	40	400

Creació personatges i animacions	Dissenyador	40	800
Publicitat i posada a la venda	Productor	60	900
Documentació	Cap del projecte	90	2250
Total		660	11880

Taula 2. Cost recursos humans

2.3.2.3 Cost total

Ajuntant els costos anteriors, ens surt que si es fes aquest projecte en un equip, el preu total seria de 16215,00€

3 METODOLOGIA

Com se'ns ha ensenyat a l'assignatura de Organització i Gestió de Sistemes d'Informació, la metodologia estàndard pel desenvolupament d'un projecte sol ser la Plan-Do-Check-Act: planejar el que es vol fer, dur-ho a terme, testejar i fer les modificacions necessàries en el cas de que el resultat no sigui l'esperat.

Tot i així, per aquest projecte s'ha utilitzat una metodologia personalitzada plantejada pel nostre tutor més adequada per aquest projecte. Aquesta metodologia està formada per aquests passos, veure Figura 11:

1. Triar el treball
2. Decidir eines a utilitzar
3. Aprendre el llenguatge de programació necessari i l'ús de les eines
4. Dividir el treball en diferents parts per tal de tenir més clars quins són els objectius concrets de cada una d'elles
5. Estructurar les parts de manera que no es comenci una sense tenir acabades les parts que es necessiten per començar-la.
6. Escollir la part que toca desenvolupar
7. Desenvolupar la part triada
8. Comprovar que la feina realitzada és satisfactòria
   8.1. Si el resultat no és satisfactori, tornar al punt 7 per arreglar els punts necessaris
   8.2. Altrament, si el resultat és l'esperat i queden parts a fer, tornarem al pas 6. En el cas de que no quedin parts, continuarem al punt 9
9. Ajuntar les parts que s'han desenvolupat i comprovar que el funcionament de totes les parts conjuntes és el correcta
   9.1. Si el resultat no és el correcte, tornar al punt 7 per arreglar les parts que no funcionen correctament
   9.2. Altrament es continuarà al pas 10
10. Generar models d'exemple per comprovar el bon funcionament de funcionalitats més concretes
    10.1. Si el resultat no és l'esperat, tornar al punt 7 per modificar les parts que no funcionen correctament
    10.2. Altrament, passar al pas final, el pas 11
11. Arrodonir la documentació desenvolupada al llarg del projecte.

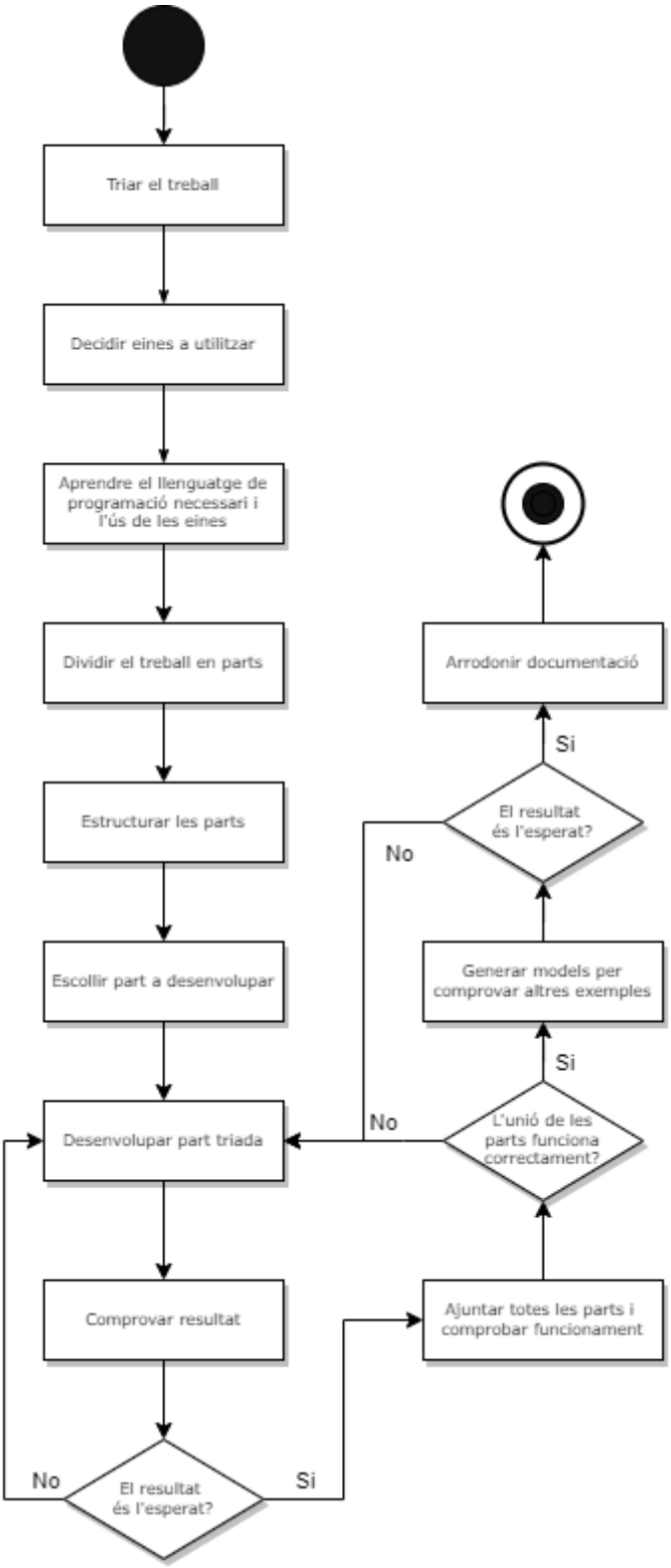

Figura 11. Diagrama de la metodologia utilitzada

4 PLANIFICACIÓ

Tot projecte necessita una planificació del temps dedicat o que es vol dedicar. En aquest capítol de la memòria es veurà la planificació utilitzada pel desenvolupament del projecte.

4.1 TASQUES PLANIFICADES

4.1.1 Planificació del joc

En aquesta primera tasca es defineix l'objectiu del joc, les característiques principals i els elements del videojoc.

4.1.2 Estudi dels diferents motors de videojocs

Aquesta tasca consisteix en buscar diferents motors de videojocs, buscar-ne les seves característiques per al final escollir-ne un.

4.1.3 Estudi del motor de videojocs Unity

En el cas d'aquest projecte, s'ha triat utilitzar el motor Unity, que en aquesta tasca s'estudia i s'aprèn a utilitzar.

4.1.4 Estudi de la plataforma GMP

Aquesta fase està dedicada a l'anàlisi i l'estudi de la plataforma GMP per veure si de veritat és viable utilitzar-la en el projecte i si ho és, estudiar com es pot fer per extreure'n les dades per utilitzar-les en el videojoc.

4.1.5 Estudi del llenguatge C# per Unity i GMP

Aquesta etapa està dedicada a aprendre el funcionament del llenguatge C# utilitzat en les dos plataformes, tant en Unity com GMP.

4.1.6 Disseny i implementació de la recollida de dades de poblacions reals

Aquesta tasca defineix com, a partir de l'anàlisi i estudi de la plataforma GMP i els coneixements en C#, podem extreure les dades de la plataforma i guardar-les de manera que les puguem utilitzar en el videojoc.

4.1.7 Disseny i implementació de la càrrega dels elements guardats

En aquesta etapa, a partir de les dades extretes de la plataforma GMP, es dissenya i s'implementa com es carreguen en el videojoc, és a dir, com es construeix el nostre mapa. Depenent del tipus i de l'ús que se'n vulgui donar en el projecte, cada element del joc pot

necessitar unes opcions diferents que sense elles no s'aconsegueix el resultat esperat, i com que carreguem el mapa de poblacions senceres, no es pot anar element per element posant les opcions manualment.

4.1.8 Assignació de textures als elements carregats

Aquesta tasca defineix com, a partir de les dades, es poden carregar textures en els elements. És una tasca important ja que no es pot aprofitar el sistema que utilitza GMP.

4.1.9 Disseny i implementació del sistema de moviment del personatge

En aquesta tasca es defineix el sistema de moviment i control del personatge del videojoc. També es defineixen les velocitats en que es mourà el personatge, les velocitats en que rotarà i les animacions corresponents a cada moviment.

4.1.10 Disseny i implementació de la IA dels enemics

Aquesta etapa defineix els elements necessaris per tal que els enemics apareguin, sàpiguen com i per on seguir al jugador i un cop estan a prop, ataquin.

4.1.11 Disseny dels moviments dels enemics

Aquest apartat s'ocupa dels moviments i les animacions que realitzaran els enemics al moment d'aparèixer, seguir el jugador i atacar-lo.

4.1.12 Disseny i implementació interaccions

En aquesta tasca s'implementen les diferents interaccions que poden ser amb el propi escenari com amb els enemics.

4.1.13 Implementació sistema de control de càmera

En aquesta part s'implementa el funcionament de les càmeres, ja sigui una càmera estàtica que segueixi el jugador o una càmera que pugui rotar el jugador per veure el seu voltant.

4.1.14 Disseny i implementació HUD

Aquesta tasca s'ocupa del disseny i implementació del HUD, que té la funció de mostrar informació al jugador com pot ser la vida actual.

4.1.15 Verificació i proves

Aquesta tasca s'ocupa de, amb tots els components i funcionalitats del joc junts, realitzar proves del funcionament de cada un dels components.

4.1.16 Documentació

Per últim, en aquesta tasca s'ajunta tota la documentació que s'ha anat realitzant durant tot el projecte

4.2 TEMPS ESTIMAT I DEFINITIU

El projecte en un principi estava pensat per començar a fer-lo a principis de juliol després de haver estat parlant amb el nostre tutor les últimes setmanes de juny i acabar-lo a finals d'any tal com mostra la Figura 12.

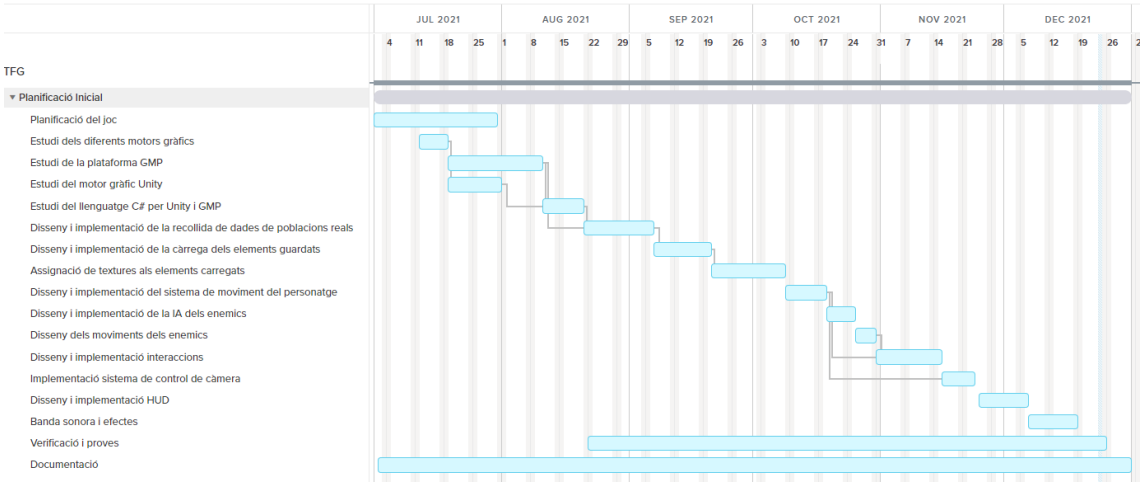

Figura 12. Diagrama de gantt, temps estimat

Tot i ser aquest el temps estimat inicialment, per diferents motius, el temps definitiu és com el de la Figura 13. Es pot veure que hi ha tasques que han ocupat molt de temps, la raó és que s'han hagut d'anar modificant a mesura que el projecte avançava i els requeriments de la tasca eren diferents.

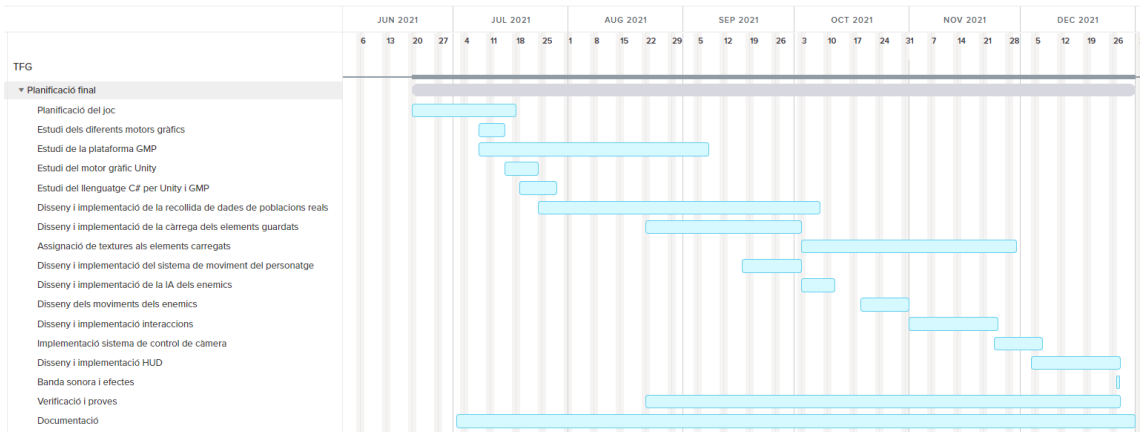

Figura 13. Diagrama de gantt, temps definitiu

4.3 RESULTATS ESPERATS

El resultat que s'espera del projecte és aconseguir desenvolupar un videojoc funcional però amb moltes àrees per ampliar.

S'espera que el jugador pugui moure's lliurament pel mapa mentre és perseguit per una sèrie d'enemics que apareixeran dins un radi del jugador, que el jugador pugui lluitar contra aquests enemics i que els pugui abatre ja sigui en combat cos a cos o amb armes de foc. El jugador també ha de poder ser atacat pels enemics i per tant, pugui ser abatut, el que farà que es perdi la partida.

A més de poder-se moure lliurament, el jugador, un cop s'acosta als extrems del mapa, el videojoc ha de fer que es pari de moure i obrir un menú on hi hagi diferents localitzacions disponibles a triar. Llavors el jugador, clicant en una d'aquestes localitzacions, podrà continuar la partida en aquesta nova localització.

Un últim resultat esperat és, aconseguir desenvolupar un videojoc que pugui servir com a base per futurs jocs ambientats en entorns reals.

5 MARC DE TREBALL I CONCEPTES PREVIS

En aquest apartat s'explica tota la feina realitzada durant els anàlisi i estudis per poder dur a terme el projecte

5.1 INTRODUCCIÓ ALS MOTORS DE VIDEOJOCS

Un motor de videojocs, *Game Engine* en anglès, és un software que permet crear i desenvolupar videojocs facilitant la feina als desenvolupadors que, utilitzant aquests software, poden no haver-se de preocupar de les capes més baixes del sistema com poden ser les llibreries gràfiques del sistema operatiu. Les funcionalitats típiques que ofereixen els motors de videojocs són renderitzar gràfics 2D o 3D, simulació de físiques, animacions, àudios, intel·ligència artificial, creació i modificació de scripts, entre altres. Veure Figura 14.

Figura 14. Diferents aspectes que formen un motor de videojocs

Els motors de videojocs es poden separar en dos categories, els motors gràfics i els motors físics. Els primers s'ocupen de l'aspecte visual del videojoc, creant les imatges, les textures. Els segons en canvi s'ocupen d'integrar les lleis físiques en, el nostre videojoc ocupant-se de les simulacions físiques que pugui tenir el nostre videojoc com la gravetat, la massa d'un objecte, friccions, forces...

Hi ha una gran varietat de motors de videojocs, alguns d'ells especialitzats en algun gènere o tipus de videojoc o pensats per desenvolupar videojocs en una plataforma concreta. Per aquestes raons, l'elecció del motor gràfic és important per qui vulgui desenvolupar un videojoc, ja que una mala elecció pot complicar molt el procés de desenvolupament. Per tant, aquestes són les característiques en les que ens basarem per triar un motor gràfic:

- Facilitat en el desenvolupament del videojoc. Un motor de videojocs ha d'aconseguir que amb uns coneixements mínims, un usuari pugui explorar i aprendre pel seu compte sense quedar-se bloquejat.
- Plataformes en que es pot utilitzar. El videojoc funcionarà en les plataformes que el motor gràfic estigui preparat per treballar.
- Documentació i informació de qualitat. Tot i que el motor faciliti el procés de desenvolupament, es necessitaran funcionalitats que no sabem que el motor disposa o

podem crear funcionalitats. Si el desenvolupador ha d'aprendre com funciona pot resultar una tasca impossible sense disposar de documentació.

- Organització dels projectes. Poder separar el contingut del joc en grups de treball.
- Barrera tècnica baixa. D'aquesta manera disminueix la càrrega de treball al desenvolupador

Nosaltres, com s'ha dit, ens basarem en buscar aquestes característiques ja que sóc un desenvolupador amb poca experiència, però desenvolupadors amb més experiència buscaran altres característiques com millor motor gràfic, millor motor de físiques, millor optimització de les intel·ligències artificials, etc.

5.2 EXEMPLES DE MOTORS DE VIDEOJOCS

5.2.1 Unity

Unity és un motor multi plataforma desenvolupat per Unity Tecnologies. És un motor que és molt utilitzat per empreses independents o aficionats gràcies a la gran quantitat de contingut gratuït, comunitat, documentació i l'oportunitat de publicar un joc sense haver de comprar cap llicència fins que el joc genera més de l'equivalent a 100000USD a l'any.

Figura 15. Logotip Unity

Com s'ha comentat, Unity és multi plataforma, podent ser utilitzat des de dispositius amb Windows o MacOS i permetent desenvolupar jocs per Windows, Mac, Xbox 360 i PlayStation 3 (si encara algú li interessa), PlayStation 4, PlayStation 5, Xbox One, Xbox Series S, Xbox Series X, Wii, iPad, iOS, Android i Windows Phone. A més, una part molt important a Unity, tot i que l'usuari pugui desenvolupar per moltes plataformes diferents, també es facilita molt canviar un joc de plataforma aconseguint el que abans podia costar mesos de feina en una empresa, com pot ser passar un joc d'Android a iPhone, ara es pugui fer modificant uns paràmetres.

Unity permet desenvolupar videojocs programant els scripts en C# i JavaScript.

Gràcies a Unity, s'han desenvolupat una gran quantitat de jocs amb milers de jugadors com pot ser el Cuphead, Cities: Skylines, Escape from Tarkov, Hollow Knight, Hearthstone, Pillars of Eternity entre molts altres grans títols. Veure Figures 16, 17 i 18

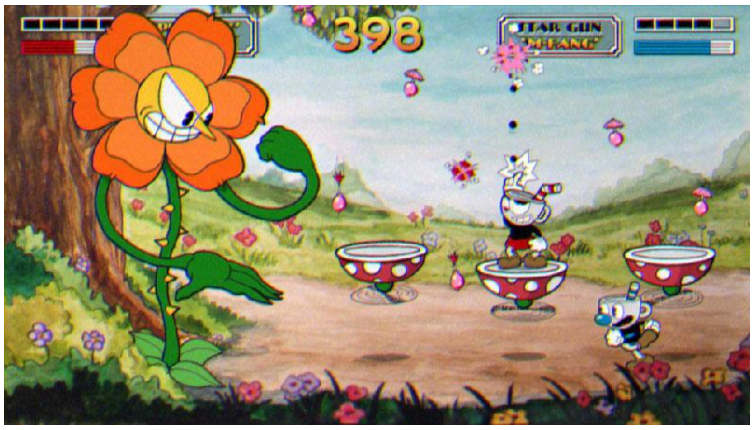

Figura 16. Captura de pantalla del videojoc Cuphead

Figura 17. Captura de pantalla del videojoc Cities: Skylines

Figura 18. Captura de pantalla del videojoc Hearthstone

5.2.2 Unreal Engine

Unreal Engine és un altre motor, que com el Unity, disposa d'una comunitat molt gran, documentació extensa, facilitat d'ús i dóna la capacitat, les eines i el suport als usuaris novells per tal que puguin desenvolupar un videojoc.

Unreal Engine, desenvolupat per Epic Games, va ser utilitzat per primer cop l'any 1998 en el joc Unreal. Des de llavors, la companyia ha continuat actualitzant-lo fins que s'ha convertit en el motor d'avui en dia que és una de les opcions preferides per molts desenvolupadors. Actualment hi ha disponible el que és conegut com Unreal Engine 5 però en accés anticipat. Es preveu que durant els primers mesos de 2022 es publiqui la versió final.

Figura 19. Logotiu Unreal Engine

En una de les últimes actualitzacions, Unreal Engine presentava la eina Blueprints, que a partir de diagrames d'estats i transicions, permet als usuaris crear un videojoc sense haver de programa ni una sola línia de codi.

A més, igual que Unity, permet desenvolupar jocs per diferents plataformes. L'última versió, Unreal Engine 5, permet desenvolupar jocs per PlayStation 5 i Xbox Series X i Series S sense pèrdre la capacitat de desenvolupar per Windows, Xbox 360, PlayStation 3, Android, macOS, iOS, Linux, PlayStation 4, Xbox One, entre moltes altres.

Un altre punt de Unreal Engine, és que, de la mateixa manera que Unity, permet publicar un joc sense la necessitat d'una llicència. El que canvia són les condicions, mentre que amb Unity no es paga a no se que el videojoc superi l'equivalent a 100000USD a l'any, Unreal Engine fa pagar un 5% dels ingresos que genera el videojoc si aquest genera més de 3000USD per quatrimestre.

Alguns dels jocs més coneguts desenvolupats amb Unreal Engine són: Fortnite, Sea of Thieves, Little Nightmares 1 i 2, Gears 5, Final Fantasy 7 Remake, Ark: Survival Evolved. Veure Figures 20, 21 i 22.

Figura 20. Captura de pantalla del videojoc Fortnite

Figura 21. Captura de pantalla del videojoc Final Fantasy 7 Remake

Figura 22. Captura de pantalla del videojoc Little Nightmares 2

5.2.3 CryEngine

CryEngine és un motor de videojocs desenvolupat per l'empresa Crytek. Al principi, estava pensat com un motor de demostració per Nvidia però al veure el potencial del que disposava, el van utilitzar per desenvolupar el primer Far Cry.

Figura 23. Logotip CryEngine

A diferència dels dos motors mencionats anteriorment, CryEngine és poc amigable pels desenvolupadors que no disposen d'experiència. Això és causa de les seves interfícies i la manca de facilitats per utilitzar-lo. Tot i així CryEngine destaca per la gran qualitat gràfica i el sistema de físiques, que són reconeguts com alguns dels millors del sector.

Alguns dels jocs destacats desenvolupats en el CryEngine són: Far Cry, State of Decay, Crysis 1, 2 i 3, Robinson: The Journey. Veure Figures 24 i 25.

Figura 24. Captura de pantalla del videojoc Far Cry

Figura 25. Captura de pantalla del videojoc State of Decay

5.2.4 Stencyl

Stencyl és un motor completament diferent a tots els anteriors. Està pensat per usuaris amb molt poca experiència tant desenvolupant videojocs com programant.

Pensat per desenvolupar jocs en plataformes com Windows, Mac, Linux, HTML5, Android i iOS, no requereix de codi. La lògica d'aquest motor s'implementa a partir de peces com si fos un puzle. També ofereix la possibilitat de que si l'usuari sap programar, pot crear noves peces del puzle personalitzades. Només ofereix desenvolupar jocs en 2D però et facilita moltes eines per crear el món que es desitgi, així com els personatges. Veure Figura 26.

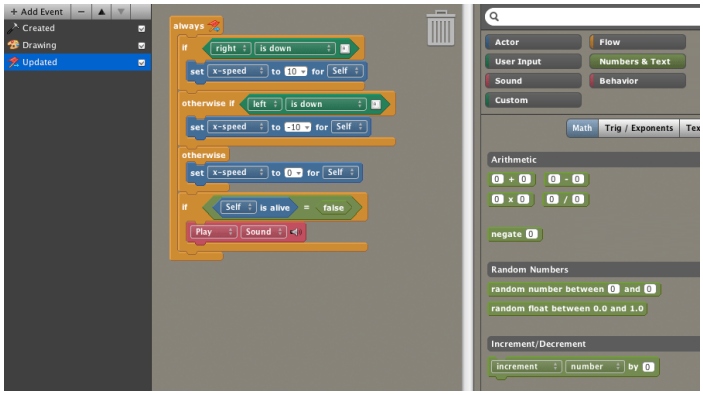

Figura 26. Com es programa a Stencyl

Stencyl és molt menys conegut comparat amb els altres motors mencionats i els jocs que s'han desenvolupat amb ell també són menys coneguts. Veure Figura 27.

Figura 27. Captura de pantalla del videojoc Reaching Finality

5.3 MOTOR TRIAT

Després d'haver mirat diferents motors gràfics, m'he decantat per Unity. Les raons per què m'he decantat per aquest motor gràfic són:

- No és el primer cop que utilitzo aquest motor
- És un motor amb una corba d'aprenentage ideal pel projecte
- Té una comunitat molt gran per donar suport a altres desenvolupadors
- A més de la comunitat disposa de molta documentació i de qualitat
- GMP està pensat per treballar-se amb Unity, una de les opcions era extreure les dades amb Unity i carregar-les en un altre motor gràfic, però al final també era millor opció utilitzar Unity també pel projecte
- Llenguatge C#. Després de les pràctiques a l'empresa ja tenia una mica d'experiència en aquest llenguatge i tant Unity com GMP l'utilitzen.

5.4 GOOGLE MAPS PLATFORM

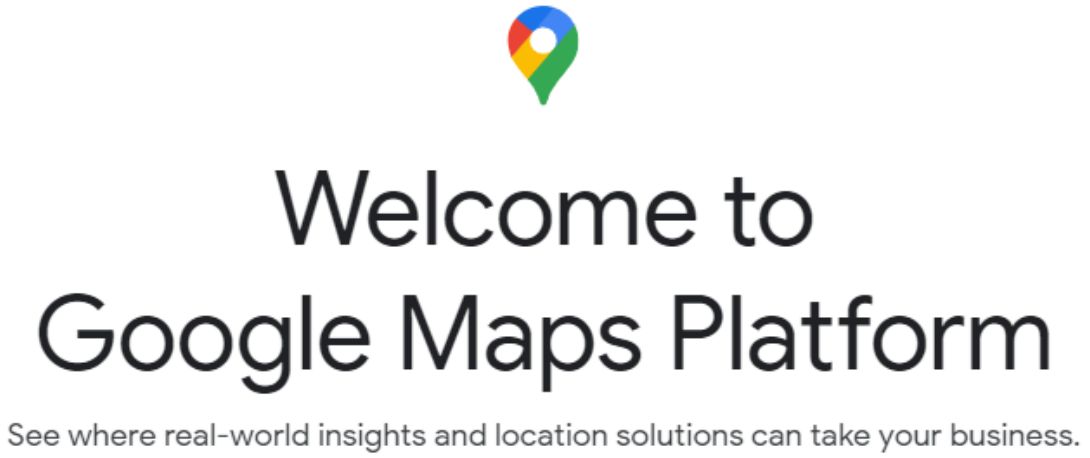

Figura 28. Pàgina principal Google Maps Platform

Google Maps Platform (GMP) és una plataforma de Google Cloud que ofereix deferents APIs i SDKs per tal que l'usuari pugui utilitzar dades del món real. Les eines que ofereix GMP es poden categoritzar en tres grups diferents (des del 18 de Octubre que la GMP s'ha actualitzat descontinuant algunes parts importants pel nostre projecte com GMP for gaming juntament amb la Maps SDK for Unity. També ha afegit API noves que, d'haver estat disponibles al principi del projecte, s'haguessin utilitzat. En vermell es marquen les eines que ja no estaran disponibles i en verd es marcaran les eines noves):

- Eines de Maps
  - API mapes dinàmics
  - Maps SDK for Unity
  - API street view
  - API elevació terreny
  - API mapes estàtics
  - API street view no interactiva
- Eines de Rutes
  - API de direccions
  - API de distàncies
  - API de carreteres
- Eines de Llocs
  - API comerços
  - API convertir coordinades en punts d'interès i viceversa.
  - API geolocalització sense necessitat de GPS
  - API detalls de locals
  - API fotos d'un lloc concret
  - API ubicació actual
  - API zones horàries

Com s'ha comentat hi ha eines que ja no estan disponibles com a tal, però que s'han implementat en altres o es presenten d'una altre manera. El problema que tenia anteriorment

la plataforma era que la unió entre diferents API podia ser confús. Per aquest motiu no s'ha implementat la API de comerços i no s'ha pogut implementar l'elevació del terreny.

Per tant, pel projecte, s'ha utilitzat GMP for gaming que oferia la SDK for Unity. Si el projecte es comencés a dia d'avui, s'aprofitaria també l'API d'elevació del terreny i l'API de convertir coordinades en punts d'interès.

6 REQUERIMENTS DEL SISTEMA

En aquest apartat s'especifiquen els requeriments de la nostra aplicació. Els requeriments els dividirem en dos parts, els requeriments funcionals i els no funcionals.

6.1 REQUERIMENTS FUNCIONALS

Els requeriments funcionals són els que fan referència a les funcionalitats que el sistema ha de dur a terme. Els requeriments funcionals de l'aplicació són:

- El jugador ha de ser capaç de moure's lliurament pel mapa, saltar i atacar a enemics
- El jugador ha de ser capaç de poder triar si vol la càmera fixe o una càmera que roti al personatge i que es pugui controlar a través del ratolí.
- El jugador ha de ser capaç de, quan s'acosta al final del mapa, poder escollir on anar de les localitzacions disponibles.
- El jugador veurà la informació del personatge, com la vida, a través d'un HUD
- El jugador ha de poder morir. En aquest cas el jugador triarà si tornar a aparèixer o acabar la partida
- Els enemics han d'aparèixer dintre un cert radi del personatge.
- Els enemics un cop apareguts, perseguiran al personatge.
- Els enemics apareixeran i abans de perseguir al personatge tindran una animació de aparèixer.
- Els enemics no apareixeran en cossos d'aigua.
- Els enemics no podran perseguir al personatge si han de passar per cossos d'aigua. En aquest cas s'eliminarà l'enemic per donar l'oportunitat a un pròxim enemic.
- Els enemics, un cop dins una certa distància del personatge, l'atacaran i li trauran vida
- Els enemics han de poder rebre mal i morir.
- El jugador ha de ser capaç de recollir objectes que apareixen aleatòriament pel mapa
- El jugador disposarà de dos atacs diferents, cos a cos i arma de foc

6.2 REQUERIMENTS NO FUNCIONALS

Els requeriments no funcionals són aquells que fan referència a restriccions de recursos, seguretat, optimització, interfícies, característiques que s'han de tenir en compte a l'hora de dissenyar.

Respecte al requeriment de seguretat, el projecte està pensat per a què sigui *single Player* i que, per tant, no es necessitin mesures de seguretat extres ja que no té la necessitat de connectar-se a la xarxa.

Pel que fa la optimització, no es necessitarà optimitzar el videojoc ja que no disposarà de escenaris suficientment grans com per què puguin resultar un problema per l'usuari, sempre i quan l'usuari disposi dels requeriments mínims del motor Unity, Taula 3, i restriccions de la plataforma. Aquestes són restriccions del propi motor.

Requeriments mínims	Windows	MacOS	Linux
Versió Sistema Operatiu	Windows 7 (SP1+) i Windows 10 només 64 bits	Sierra 10.12.6+	Ubuntu 16.04, Ubuntu 18.04 i CentOS 7
CPU	Arquitectura X64 amb SSE2 ISA	Arquitectura X64 amb SSE2 ISA	Arquitectura X64 amb SSE2 ISA
API gràfics	GPU compatibles amb DX10, DX11 i DX12	GPU de Intel o AMD compatibles amb Metal	GPU de Nvidia o AMD compatibles amb OpenGL 3.2+ o compatibles amb Vulkan
Requeriments adicionals	Els drivers han de ser els oficials del venedor del hardware	Només drivers oficicials de Apple	Drivers oficials de Nvidia o de AMD

Taula 3. Requeriments mínims Unity

Pel que respecta els requeriments tècnics, el videojoc s'ha testejat en tres dispositius diferents, dos ordinadors de sobretaula i un ordinador portàtil els mateixos dispositius que s'han utilitzat per desenvolupar el videojoc, descrits al Capítol 2 de la memòria.

7 ESTUDI I PRESA DE DECISIONS

En aquest apartat es veuran els programes i llibreries que s'han utilitzat pel desenvolupament del projecte. També s'inclouran eines que s'han utilitzat per fer la memòria.

7.1 SISTEMA OPERATIU

Com s'ha comentat al Capítol 2 de la memòria, el sistema operatiu utilitzant tant per desenvolupar com per testejar el videojoc, ha estat el Windows 10 Home x64.

7.2 UNITY

Com ja s'ha vist al Capítol 5 Apartat 3 de la memòria, Unity és el motor de videojocs que s'ha triat per desenvolupar el videojoc. Les explicacions de Unity més generals i conceptes prèvis es troben a l'apartat mencionat anteriorment, per tant en aquest apartat ens centrarem en elements més concrets que disposa el motor Unity.

Figura 29. Logotip Unity

7.2.1 Organització de la plataforma / Interfície de Unity

Unity està organitzat en una sèrie de finestres que es poden utilitzar. A continuació es mostren les finestres més utilitzades en el projecte. Veure Figura 30.

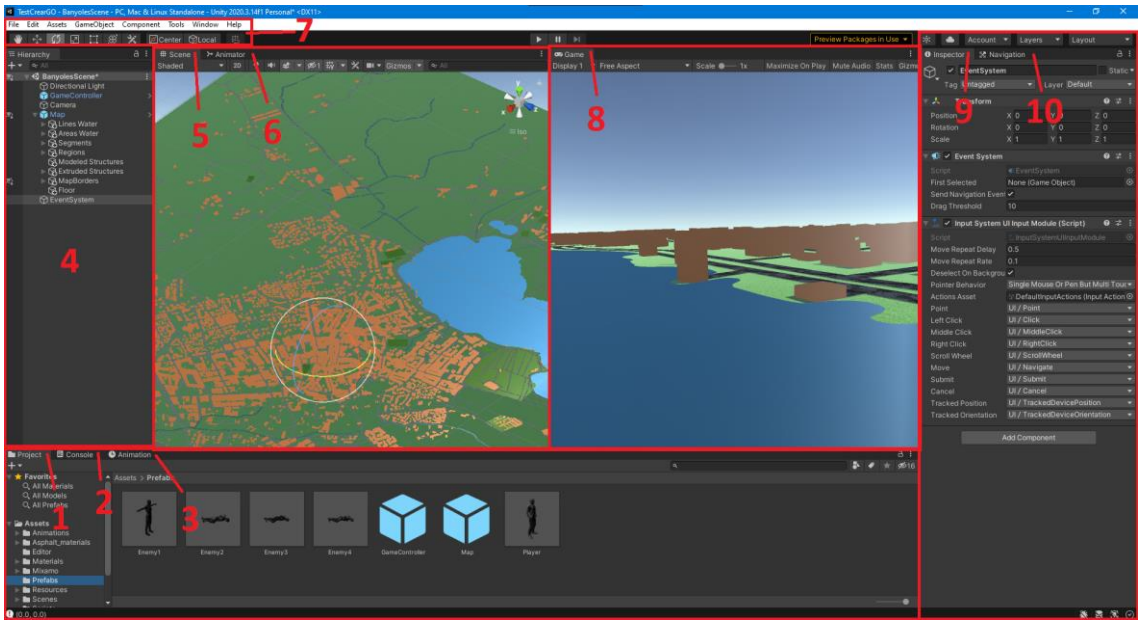

Figura 30. Interfície de Unity

1. Project: Pestanya que mostra els elements utilitzats en el projecte tenint a l'esquerra l'arbre de directoris i a la dreta el contingut del directori que necessitem.

2. Console: Pestanya que mostra els registres, errors i warnings de compilació i els errors d'execució que pot tenir el programa.

3. Animation: Mostra l'animació que es té seleccionada. No mostra res si no hi ha cap animació seleccionada. Des d'aquí es pot modificar l'animació.

4. Jerarquia de l'escena: Aquesta part de l'interfície mostra tots els GameObjects de l'escena que es té carregada i la relació que poden tenir els uns amb els altres. Per exemple, el GameObject "Map" és el pare de altres GameObjects que, a la vegada, són pare d'altres GameObjects.

5. Visió d'escena: Aquesta finestra ens permet veure i modificar manualment l'escenari que tenim seleccionat. A partir d'aquesta finestra, també, és des d'on es creen els menús que, encara que el videojoc sigui en 3D i els menús en 2D, hi ha una opció que permet canviar com es veu aquesta finestra.

6. Animator: Aquesta pestanya ens permet entrar a l'Animator, una eina que és la que permet ajuntar diverses animacions a partir de transicions. El resultat és una classe Animator que en podrem crear una instància als scripts, i a partir d'aquí modificar els valors necessaris per tal que les animacions que es mostren siguin les adequades.

7. Barra d'eines i menús: Àrea de la interfície que presenta diferents eines i menús des d'on l'usuari pot modificar parts del projecte.

8. Visió de joc: En aquesta finestra podem veure el que es veurà quan s'estigui jugant. Pot ser utilitzada encara que el joc no estigui en execució per comprovar parts que s'han modificat a la pestanya de l'escena, però és més utilitzada per fer les proves d'execució.

9. Inspector: En aquesta pestanya es visualitzen tots els components del GameObject que estigui seleccionat. Aquests components es poden modificar des d'aquesta pestanya.

10. Navigation: Des d'aquesta pestanya, Unity permet crear les parts de l'escena en que els agents podran passar.

7.2.2 Conceptes necessaris

De manera que el contingut que s'explica més endavant es pugui entendre, és important conèixer els següents conceptes de Unity

7.2.2.1 GameObject

Un GameObject és una classe que permet representar qualsevol cosa que pugui existir en una escena i per tant, que es pot modificar. Els GameObject estan formats per diferents Components que són els que donen les característiques desitjades als GameObject. Alguns exemples de GameObjects utilitzats al nostre projecte són els de les Figures 31, 32 i 33.

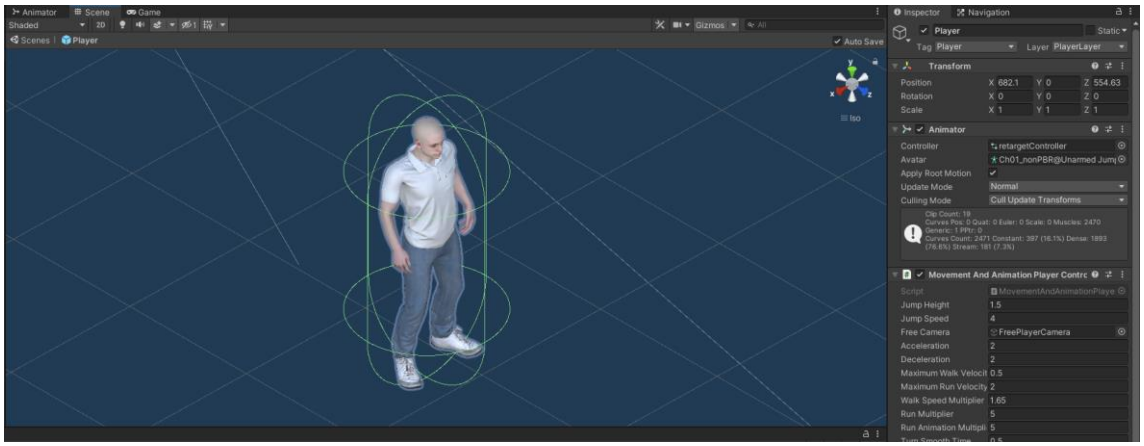

Figura 31. GameObject del personatge

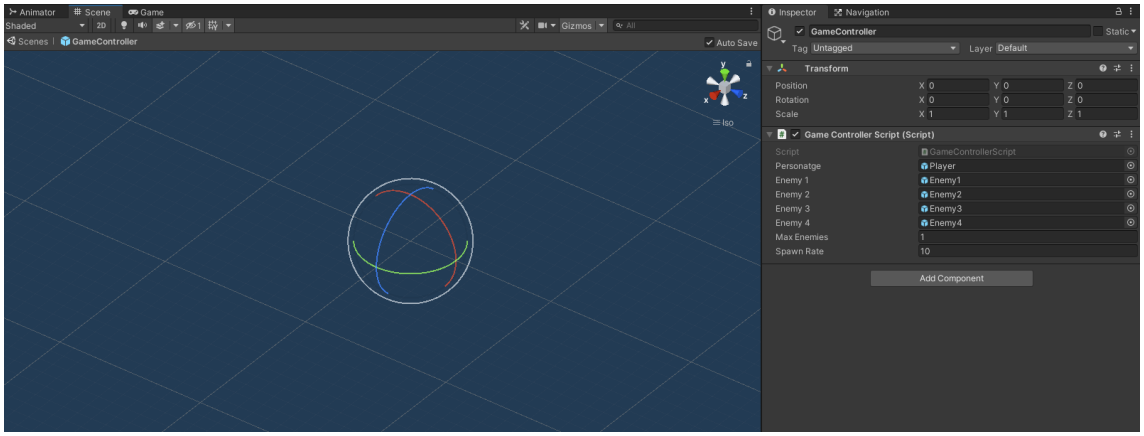

Figura 32. GameObject controllador del joc

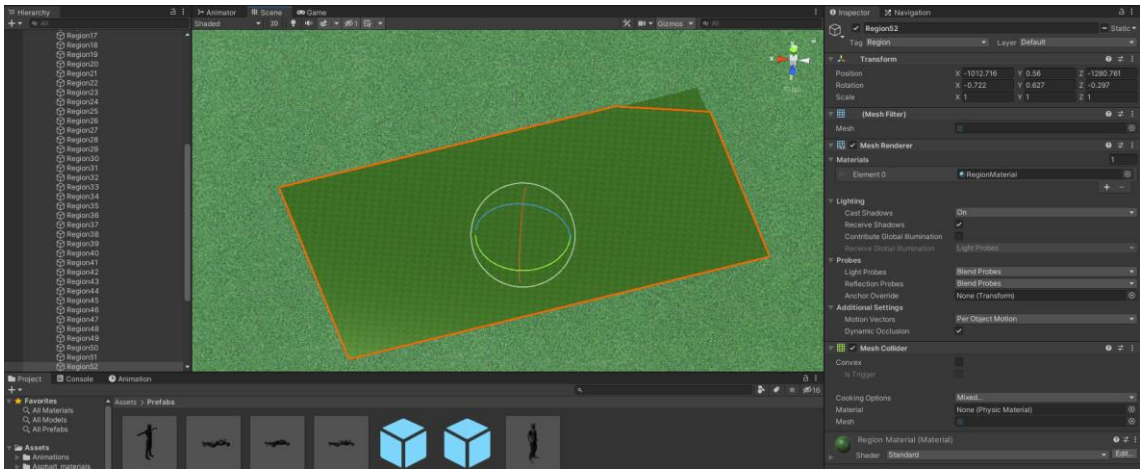

Figura 33. GameObject part de un terreny

7.2.2.2 Asset

Un asset és un element que es pot utilitzar en el projecte. Pot venir en forma de fitxer creat fora de Unity i després importat, com un model 3D, un àudio o una imatge, o pot ser un element creat dintre el propi Unity com poden ser models 3D primitius (cubs, esferes, piràmides, plans, etc).

7.2.2.3 Prefab

Un prefab és un tipus de asset que permet emmagatzemar un GameObject amb tots els components i propietats. Un prefab actua com una plantilla que, a partir d'ella, es poden crear noves instàncies de l'objecte a l'escena. Qualsevol modificació feta a un prefab es veurà, també, modificat en totes les instàncies del prefab. En el cas de que només es volgués modificar una de les instàncies, també és possible modificant la pròpia instància en comptes del prefab.

7.2.2.4 Component

Els components són tots els atributs que té un GameObject. Els components poden ser de tot tipus, la forma, si emeten llum, si es poden moure, si són afectats per la gravetat, si han de xocar amb altres objectes, sistemes d'animacions, etc. Tot i així, tots aquests components són opcionals, l'únic component que és obligatori en tots els GameObject, és el Transform, que és el que s'ocupa de mostrar la posició, rotació i escala del GameObject. Veure Figura 34.

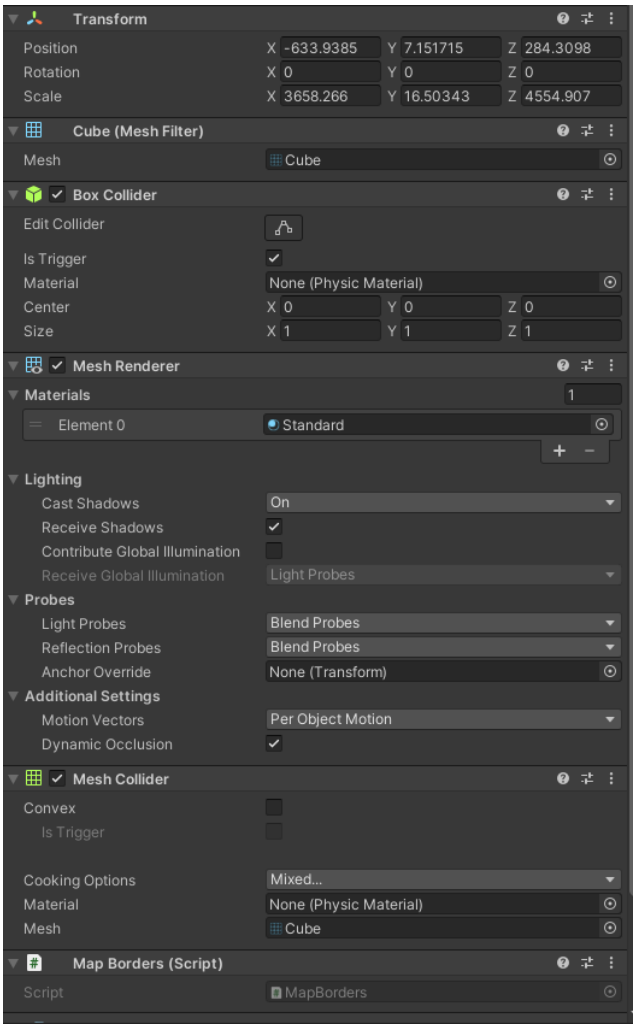

Figura 34. Components d'un GameObject

7.2.2.5 Scene

Les escenes són el lloc on el desenvolupador treballa. Són assets que contenen tota o part de l'aplicació. Per exemple, es pot fer tot un videojoc a la mateixa escena, o es pot partir en diferents escenes.

A partir de les opcions del projecte, es pot triar quina escena es mostrarà primer quan el joc s'executi.

7.2.2.6 Script

Els scripts són components dels GameObject que els hi diuen com s'han de comportar.

7.2.2.7 Editor Script

Els editor script són aquells scripts que no modifiquen el videojoc sinó que modifiquen el comportament del propi editor de Unity, permetent afegir noves funcionalitats o amagar funcionalitats que no ens interessen. Mirar la Figura 35 per un exemple de funcionalitat afegida.

Figura 35. Exemple funcionalitat afegida a l'editor

7.2.2.8 Material

Un material és l'element que controla com es veuran els GameObjects. Estan formats per una textura o un color i altres components que s'ocupen d'altres característiques, com de quina manera reacciona la llum, opacitat, etc. Veure Figura 36.

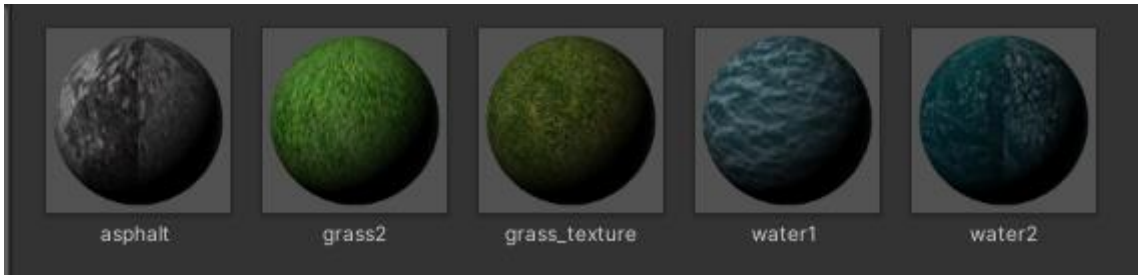

Figura 36. Materials utilitzats en parts del mapa

7.2.2.9 Textura

Una textura és una imatge bitmap estàndad que és aplicada sobre una superfície. Veure Figura 37.

Figura 37. Textures utilitzades en els materials

7.2.2.10 Collider

El collider és el component que permet a Unity i, a partir d'scripts, a l'usuari, saber quan dos objectes que tenen un component collider xoquen entre ells o ocupen el mateix espai. Podem dividir-los en dos, els IsTrigger Collider que permeten que es passi a través d'ells, i els Collider normals, que són els que no permeten que es passi a través d'ells.

7.2.2.11 RigidBody

Un Rigidbody és el component que permet que un GameObject sigui afectat per les físiques del joc, com la gravetat. El component de Rigidbody disposa de diferents atributs com poden ser la massa de l'objecte, la fricció, si és afectat per la gravetat, sobre quins eixos pot rotar o moure's.

Normalment, els rigidbody i els collider treballen junts per aconseguir una millor experiència de la física.

7.2.2.12 Camera

Les càmeres són els dispositius que capturen i ensenyen l'escenari al jugador. No només s'ocupen de l'aspecte visual del joc, sinó que també s'ocupen del so. Veure Figura 38.

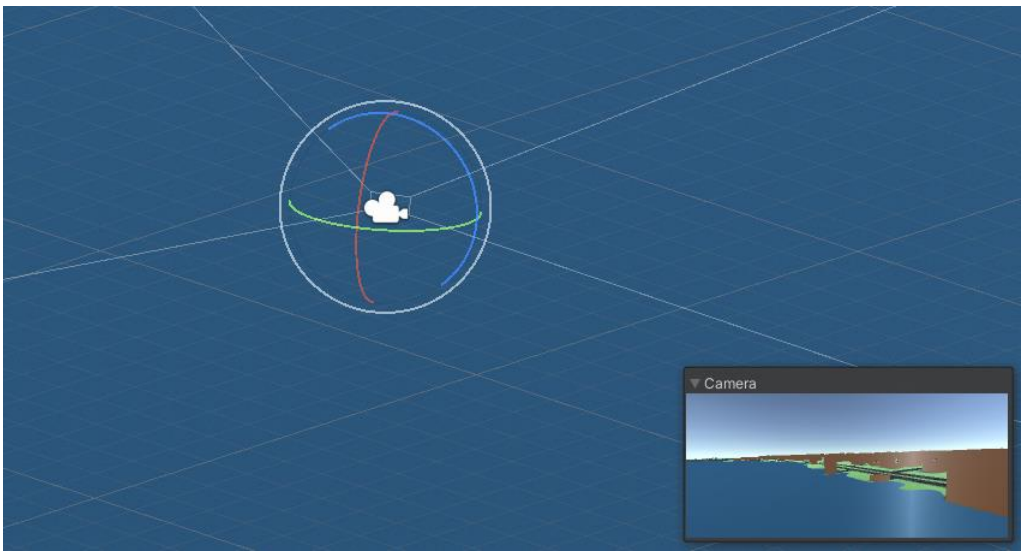

Figura 38. Càmera i image que captura la càmera

7.2.2.13 Mesh

Una mesh és el modelat 2D o 3D d'un GameObject. Les mesh estan formades per un conjunt de triganles. Les parts mínimes que han de formar una mesh són:

- Una llista de vectors3D que contenen la posició dels vèrtexs
- Una llista de int que són l'índex del vèrtex en el vector3D anterior. Aquesta llista ha de ser múltiple de tres, perquè cada triangle agafa tres índexs.

Llavors a partir d'aquí hi ha altres parts com els UV que és el que permet a Unity saber com organitzar les textures quan se li assigna una.

7.2.2.14 MeshFilter

És el component que té un model 2D o 3D que, a través d'una Mesh, dóna forma a un GameObject.

7.2.2.15 MeshRenderer
Aquest component agafa la mesh del GameObject i ho renderitza per poder visualitzar-ho a la pantalla. Un GameObject amb MeshFilter però no MeshRenderer continuarà existint i s'hi podran fer col·lisions, però el GameObject no es veurà per pantalla.

7.2.2.16 MeshCollider
Aquest és el component que permet al personatge i altres elements del mapa que hi hagi col·lisions entre ells.

7.2.2.17 Canvas
Canvas és un component que permet al desenvolupador crear interfícies d'usuari i menús pel videojoc.

7.2.2.18 Tag
Els GameObjects es poden dividir en diferents etiquetes, els Tags. Els Tags es poden fer servir per si el desenvolupador té molts tipus de GameObject del mateix tipus i els vol agrupar per així, si durant la partida els ha de buscar, pot buscar els GameObject que tinguin el tag que ha decidit.

7.2.2.19 Layer
Les Layers també són una altre manera d'agrupar GameObjects per tipus o en aquest cas, per capes. Les layers són molt útils quan volem que dos tipus diferents de GameObject ignorin les col·lisions entre ells i a l'hora de crear la NavMesh, quins tipus de GameObject volem que s'hi pugui generar la NavMesh.

7.2.2.20 Agent
Un agent és el GameObject que disposa d'un component NavMeshAgent. Aquest component permet al GameObject que es mogui per l'escena utilitzant la NavMesh.

7.2.2.21 NavMesh
Una NavMesh és una mesh que indica els llocs per on els agents poden caminar. La NavMesh és generada a partir de les mesh que hi han a l'escena i el GameObject que estan associades està marcat com a "Navigation Static". Mirar Figura 39.

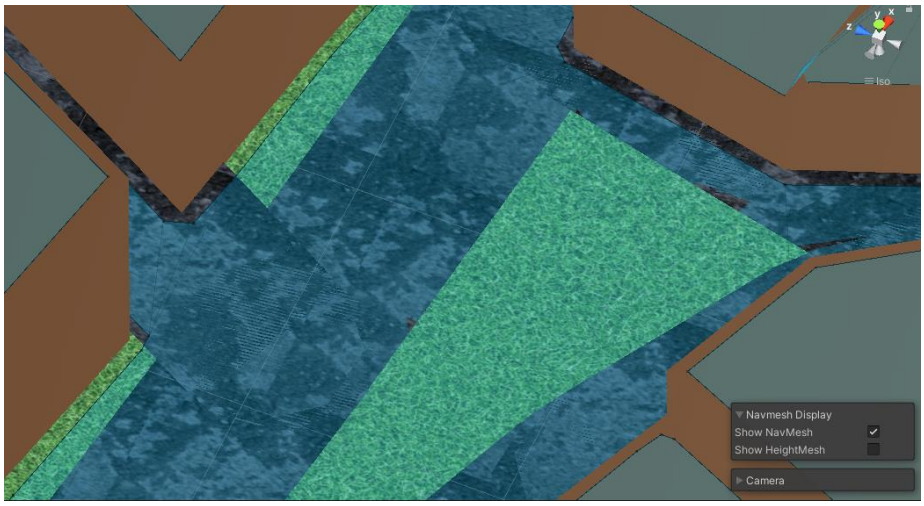

Figura 39. NavMesh en una part del mapa

7.2.2.22 Transform

El component transform és aquell que diu la posició, rotació i escala d'un GameObject.

7.2.2.23 Input System

Anteriorment, per capturar el inputs del jugador, s'utilitzaven comandes com GetKey(). Actualment Unity ha creat un nou input sytem. Aquest sistema permet agrupar tot tipus d'input en accions que crea el desenvolupador, mirar Figura 40. Amb les accions fetes, Unity genera una classe de C# per poder-ne crear una instància en el GameObject que ens interessa que disposi del input System.

Es poden crear diferents input Systems si al desenvolupador li interessa tenir una classe de C# diferent per cada tipus de GameObject que vol que rebi un input. Mirar Figura 40.

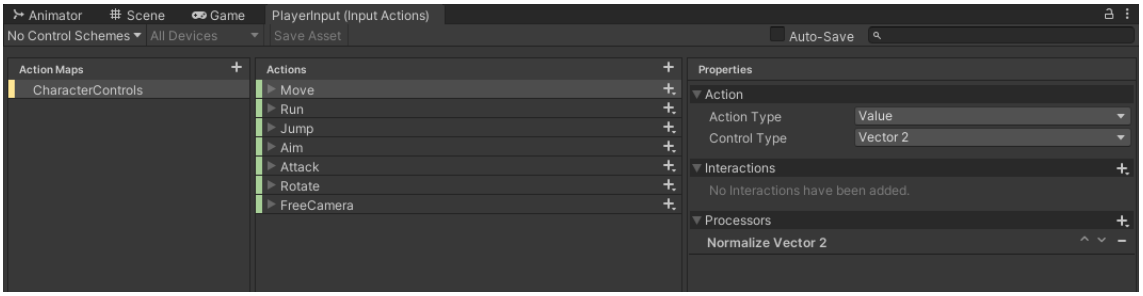

Figura 40. Input System per el jugador

7.2.2.24 Animator

Aquest component és utilitzat per tal de lligar un GameObject a un animationController. L'animationController és on hi ha les animacions, les transicions per passar d'una animació a una altra i les variables necessàries per a què es doni a terme la transició. Veure Figura 41.

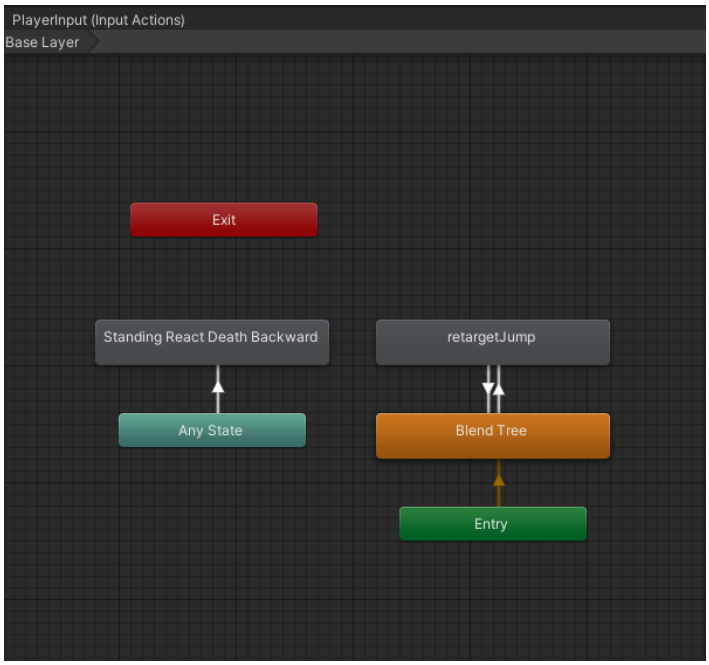

Figura 41. Exemple animator.

7.2.2.25 Blend Tree

Els blend trees són components de l'Animator. Estan formats per un conjunt d'animacions i el que permet és, com indica el seu nom, barrejar, combinar les animacions un cert grau depenent d'una variable que es controla des d'un script. Els blend trees són utilitzats per fer que les transicions siguin molt més suaus que si en comptes d'utilitzar-lo, es posessin les animacions a l'Animator i es lliguessin amb transicions. A més a més, el que també fa és que, com que es combinen dos animacions, permet al desenvolupador no necessitar tantes animacions concretes ja que el propi blend tree les aconsegueix. Mirar Figura 42.

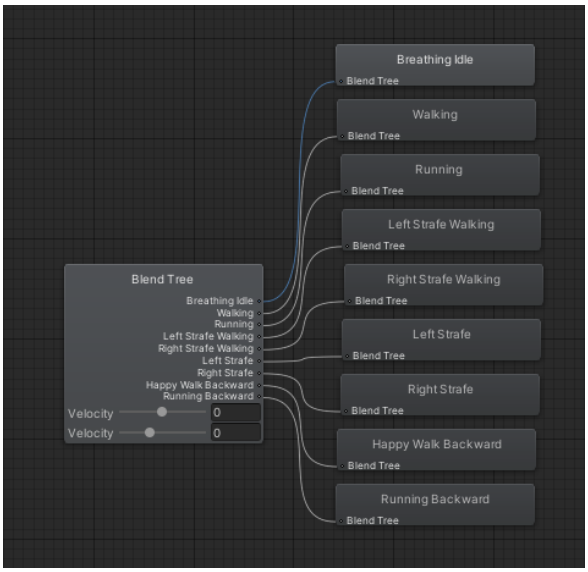

Figura 42. Exemple Blend Tree

7.2.2.26 Character Controller
El character controller és un component que permet el control d'un personatge sense necessitar rigidbody i haver de jugar amb foces per tal de fer el moviment. A diferència del rigidbody, algunes físiques com la gravetat se li han de posar a través d'un script.

7.2.3 Llibreries utilitzades

7.2.3.1 UnityEngine
Llibreria principal del motor Unity

7.2.3.2 UnityEditor
Llibreria necessària per realitzar modificacions a l'editor

7.2.3.3 System
Llibreria fonamental de C# que conté les classes més utilitzades que descriuen els valors i referències de tipus de dades més comuns.

7.2.3.4 System.Collections
Llibreria que conté classes que defineixen conjunts de dades com poden ser les llistes.

7.2.3.5 System.Collections.Generic
Llibreria que conté les classes de conjunts de dades més genèrics, per tant permeten crear col·leccions fortament tipades.

7.2.3.6 System.IO
Llibreria que conté els tipus necessaris que permeten llegir i escriure en fitxers.

7.2.3.7 System.Linq
Llibreria que conté les classes i interfícies que permeten fer consultes que utilitzen LINQ.

7.2.3.8 UnityEngine.InputSystem
Llibreria necessària per poder llegir els inputs del jugador.

7.2.3.9 UnityEngine.UI
Llibreria que conté els mètodes necessaris per tal que l'usuari pugui interactuar amb els elements de la interfície gràfica del videojoc.

7.2.3.10 UnityEngine.AI
Llibreria que permet que els agents tinguin les funcionalitats de *pathfinding* de Unity.

7.2.3.11 UnityEngine.SceneManagement
Llibreria que conté els mètodes necessaris per el control de les escenes del videojoc.

7.2.4 API de Unity
Unity ofereix una API, que a partir dels scripts que crea el desenvolupador, permet comunicar-se amb els GameObject i els components. La API està formada per molts de mètodes que no he necessitat, per tant en aquest apartat mostraré els mètodes i paràmetres que he utilitzat en els scripts.

7.2.4.1 GameObject
Classe principal de totes les entitats de Unity

Valors de Retorn	Variable	Descripció
Bool	isStatic	Get i Set de si el GameObject és un element estàtic de l'escena
Int	Layer	Retorna la layer en la que es troba el GameObject
String	Tag	Retorna el tag del Gameobject
Transform	Transform	Retorna el component transform del GameObject

Taula 4. Atributs de GameObject

Valors de Retorn	Mètode	Descripció
GameObject	GameObject()	Constructor d'un GameObject
Type	AddComponent(type t)	Afageix un component de tipus type al GameObject
Type	GetComponent(type t)	Retorna el primer component de tipus type del GameObject
List<Type>	GetComponents(type t)	Retorna tots els components de tipus type del GameObject
Void	SetActive(bool b)	Activa/desactiva el GameObject
GameObject	CreatePrimitive(PrimitiveType t)	Crea un GameObject amb un MeshRenderer primitiu
GameObject	Find(string name)	Troba un GameObject pel nom i el retorna, null si no n'hi ha cap
GameObject[]	FindGameObjectsWithTag(String tag)	Retorna una taula dels GameObjects actius que tinguin un cert tag
GameObject	FindWithTag(string tag)	Retorna un GameObject actiu amb un cert tag
GameObject	Instantiate(GameObject o, Vector3 position, quaternion rotation)	Clona un GameObject i el retorna

Taula 5. Mètodes de GameObject

7.2.4.2 GameObjectUtility

Funcions de utilitat de GameObject

Valors de Retorn	Mètode	Descripció
Void	SetStaticEditorFlags(GameObject o, staticEditorFlags f)	Set de les StaticEditorFlags que són els sistemes de Unity que consideren si un GameObject és static en diferents àmbits.

Taula 6. Mètodes de GameObjectUtility

7.2.4.3 Transform

Classe del component transform d'un GameObject que permet consultar i modificar el component.

Valors de Retorn	Variable	Descripció
Vector3	position	Retorna la posició del GameObject respecte l'eix del món
Quaternion	Rotation	Retorna la rotació del GameObject respecte l'eix del món
Transform	Parent	Retorna el pare del GameObject

Taula 7. Atributs de Transform

Valors de Retorn	Mètode	Descripció
Void	LookAt(Transform p)	Rota el GameObject per tal que apunti al GameObject que té transform p.
Void	RotateAround(Vector3 punt, Vector3.up, float angle)	Rota el GameObject amb el centre el GameObject del que es vol rotar, l'eix en que es vol rotar i l'angle

Taula 8. Mètodes de Transform

7.2.4.4 MonoBehaviour

La classe MonoBehaviour és la classe base del que tots els scripts de Unity en deriven. Aquesta té mètodes per controlar l'execució del programa.

Valors de Retorn	Mètode	Descripció
Void	Awake()	Mètode que es crida quan es carrega l'instància de l'script
Void	Start()	Mètode que es crida al primer frame de l'execució
Void	OnEnable()	Mètode que es crida quan el GameObject s'activa
Void	OnDisable()	Mètode que es crida quan el GameObject es desactiva
Void	Update()	Mètode que es crida una vegada per frame.
Void	OnTriggerExit()	Mètode que es crida quan un objecte deixa de col·lisionar amb un objecte que és trigger

Taula 9. Mètodes de MonoBehaviour

7.2.4.5 SceneManager

Classe que permet gestionar les escenes del videojoc.

Valors de Retorn	Mètode	Descripció
Void	LoadScene(int i)	Mètode que carrega l'escena amb índex "i"

Taula 10. Mètodes de SceneManager

7.2.4.6 Application

Classe que permet gestionar l'aplicació del videojoc

Valors de Retorn	Mètode	Descripció
Void	Quit()	Mètode que atura l'aplicació i la tenca

Taula 11. Mètode de Application

7.2.4.7 Animator

Classe associada a l'*animator* que permet accedir al component i canviar el comportament.

Valors de Retorn	Mètode	Descripció
Bool	GetBool(String name)	Retorna el bool associat al paràmetre name
Int	StringToHash(String s)	Retorna el hash de l'string s
Void	SetBool(int param, Bool b)	Assigna el valor b al paràmetre param
Void	SetFloat(int param, Float f)	Assigna el valor f al paràmetre param

Void	SetLayerWeight(int layer, float weight)	Assigna el pes weight a la Layer layer
Int	GetLayerIndex(string name)	Retorna l'index de la layer amb nom name

Taula 12. Mètodes de Animator

7.2.4.8 NavMeshAgent

Classe del component que permet al GameObject en el que està navegar pel mapa utilitzant la NavMesh.

Valors de Retorn	Variable	Descripció
Vector3	Destination	Paràmetre que conté on ha d'anar l'agent
Bool	isStopped	Atribut que té la condició de parar o continuar moguent-se del NavMeshAgent
Vector3	Velocity	Accés a la velocitat del component NavMeshAgent.

Taula 13. Variables de NavMeshAgent

7.2.4.9 CharacterController

Classe que s'ocupa del component CharacterController encarregat del moviment del personatge.

Valors de Retorn	Variable	Descripció
Transform	transform	Paràmetre que conté el transform del GameObject

Taula 14. Variables de CharacterController

Valors de Retorn	Mètode	Descripció
CollisionFlags	Move(Vector3 v)	Mou el GameObject a la direcció v

Taula 15. Mètodes de CharacterController

7.2.4.10 InputSystem

Aquesta classe és el nou sistema de Unity per llegir els inputs de l'usuari. Està fet de tal manera que, després d'haver preparat un InputSystem, es genera una classe en C# automàticament pel que fa que cada classe generada amb el nou InputSystem sigui diferent a l'anterior.

7.2.4.11 Resources

Aquesta classe és utilitzada per carregar diferents elements que hi ha a la carpeta Resources. Aquesta carpeta és creada pel desenvolupador dins la carpeta de Assets que crea Unity al començar un projecte.

Valors de Retorn	Mètode	Descripció
Type	Load(string s)	Retorna l'objecte amb nom s de la carpeta Resources.

16

7.2.4.12 Editor

Aquesta classe és l'equivalent a MonoBehaviour però en l'entorn de l'editor i no del videojoc executant-se, és la classe base pels scripts d'editor, que s'explicaran a la secció 8.7.

Valors de Retorn	Mètode	Descripció
void	OnInspectorGUI()	Implementa el contingut del mètode a l'editor.

Taula 17. Mètode de Editor

7.2.4.13 Bounds

Aquesta classe és la representació dels límits d'un GameObject.

Valors de Retorn	Variable	Descripció
Vector3	Center	Centre dels límits

Taula 18. Variables de CharacterController

Valors de Retorn	Mètode	Descripció
Void	Encapsulate(Vector3 v)	Augmenta els límits del GameObject fins al punt v

Taula 19. Mètodes de CharacterController

7.2.4.14 Time

Classe que proveeix d'una interfície de temps de Unity.

Valors de Retorn	Variable	Descripció
Float	deltaTime	Interval en segons del temps que ha passat entre l'últim frame i l'actual
Float	timeScale	Escala en la que passa el temps.

Taula 20. Variables de Time

7.3 GOOGLE MAPS PLATFORM

Google Maps Platform (GMP) és una plataforma de Google Cloud que permet a desenvolupadors utilitzar les API de Google per extreure dades de Google Maps i aprofitar funcions de Google Maps com calcular rutes, veure punts d'interès, després de la nova actualització l'elevació del terreny, etc.

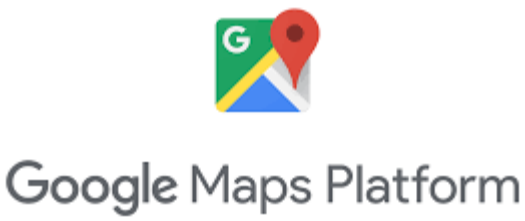

Figura 43. Google Maps Platform

En aquest projecte s'ha utilitzat GMP, més concretament l'apartat de GMP for gaming. El que ofereix aquest apartat és un Software Development Kit (SDK) per Unity. El que ofereix aquest SDK és l'opció de carregar les dades més fàcilment a Unity i poder-se moure amb dades carregant-se al moment.

Tot i la utilitat de GMP for gaming en el nostre projecte, aquesta eina està enfocada en la creació de jocs per dispositius mòbils on la ubicació del jugador és una part important del joc. D'aquesta manera, gràcies al SDK, es pot investigar com funciona ja que Google deixa alguns scripts visibles a l'usuari, a diferència d'alguns altres que l'usuari no hi pot accedir. A partir d'aquests scripts es pot aprendre com es carreguen les dades a l'entorn i aprofitar aquest moment per guardar la forma i la posició de l'objecte que s'està carregant. Un dels scripts que Google no deixa veure ni modificar és l'script d'elevació del terreny.

GMP funciona de manera que es necessita una llicència de pagament. Els pagaments estan basats en un crèdit mensual del que, a partir d'aquí, l'usuari pot triar en quines API destinar els seus recursos. Tot i així, Google ofereix un període de prova de 3 mesos amb 300USD de crèdit per utilitzar en les API o SDK que es necessiti. Al final d'aquest període de prova Google no cobra automàticament sinó que pregunta a l'usuari què vol fer. Si l'usuari decideix continuar, se li cobrarà l'import que gastarà el mes següent (no el que ha gastat en el període de prova) i si l'usuari no decideix continuar, Google bloquejarà les claus de les API per què no puguin ser utilitzades però permetrà a l'usuari guardar el progrés dels mesos de prova.

A partir dels punts anteriors, es va escollir aquesta plataforma ja que tot i que no estigui pensada per funcionar amb el nostre projecte, després del seu estudi continua sent una opció viable ja que com capaços d'aprofitar-la.

7.4 MIXAMO

Figura 44. Logotip Mixamo

Mixamo és una empresa que forma part de Adobe. Està enfocada en la tecnologia gràfica en 3D per ordinador i disposa als usuaris de serveis basats en web per a l'animació de personatges en 3D.

La raó per la que s'ha triat aquesta plataforma és per la gran varietat de models 3D i animacions totalment gratuïts dels que disposa. A més a més, a l'hora de descarregar el contingut, en el

cas de que no es vulgui modificar més, es pot descarregar en un format optimitzat per treballar amb Unity.

7.5 C#

C# (C Sharp) és un llenguatge de programació orientat a objectes desenvolupat per Microsoft com a part del framework .NET i deriva dels llenguatges C i C++ (ambdós treballats a la carrera). C# és similar a Java per exemple, però conté millores inspirades en altres llenguatges de programació.

Figura 45. Logotip C#

Aquest és el llenguatge que s'ha utilitzant en el nostre projecte tant en Unity com GMP. Tot i que Unity també es pot programar en JavaScript, personalment ja disposava de certa experiència treballant amb C#, a més de ser el llenguatge més similar als que s'han estudiat durant la carrera.

7.6 Visual Studio Code

Visual Studio Code és un editor de codi desenvolupat per a Windows, Linux i MacOS. Inclou suport per a la depuració de codi, control de versions integrat de Git, ressaltat de sintaxi, finalització intel·ligent de codi, fragments i refactorització de codi.

Figura 46. Logotip Visual Studio Code

La raó de perquè s'ha utilitzat com a entorn de programació és per la facilitat d'ús i perquè no necessita compilador ja que, un cop es guarda l'script, el propi motor Unity compila els scripts on hi ha hagut canvis. També perquè és totalment gratuït i a partir de extensions es pot aconseguir que, com s'ha comentat, trobi errors de compilació, auto completi, ressaltar la sintaxi.

7.7 Blender

Blender és una eina lliure i gratuïta dedicada a l'edició de models 3D i animacions. Compatible amb Windows, Linux i MacOS.

Figura 47. Logotip Blender

La raó pel que s'ha utilitzat Blender és per què per diferents animacions dels personatges del projecte no acabaven d'estar lligades

entre elles i, com que l'eina de Unity no permetia solucionar aquest error, Blender ha permès modificar les animacions per tal de solucionar els erros que es van presentar.

7.8 Microsoft Word

Microsoft word és un software de processador de textos desenvolupat i comercialitzat per Microsoft.

Figura 48. Logotip Word

S'ha escollit Microsoft Word per fer la memòria ja que la universitat disposava als estudiants el paquet Office 365 gratuït a més de ser l'editor en el que hi tinc més experiència treballant havent-hi realitzat altres memòries i treballs.

7.9 Team Gantt

TeamGantt és software per la planificació i col·laboració de projectes des d'una pàgina web. Disposa de tot el que es necessita per planificar i dur a terme projectes, entre ells disposa de diagrames de Gantt, que són cronogrames d'un projecte.

Figura 49. Logotip TeamGantt

Per aquesta raó i perquè ja s'havia utilitzat aquesta eina a l'assignatura de Organització i Gestió de Sistemes d'Informació, s'ha fet servir aquesta eina per fer el cronograma del projecte.

7.10 Draw.io

Draw.io és un programa online que permet fer qualsevol tipus de diagrama relacionat amb la informàtica i més (UML, diagrames de flux, models relacionals, disseny d'arquitectures, etc). És fàcil i intuïtiu d'utilitzar a més que és un programa que s'ha utilitzat en diverses assignatures, per tant és una eina en la que no fa falta familiaritzar-se.

Figura 50. Logotip Draw.io

8 ANÀLISI I DISSENY DEL SISTEMA

En aquest apartat s'explicarà com és el videojoc desenvolupat de manera conceptual sense entrar en la implementació. A més, també es mostraran les classes i diagrames per entendre millor el funcionament del videojoc.

8.1 DESCRIPCIÓ GENERAL

El videojoc que s'ha desenvolupat, com s'ha comentant en els apartats inicials d'aquesta memòria, és un joc de supervivència de món obert en tercera persona disposant de càmera fixa a l'esquena del personatge o càmera lliure quan el jugador ho decideixi. Quan el jugador comenci la partida, apareixerà en una carretera de la població que hagi escollit i, a partir d'aquí, es podrà moure lliurament mentre apareixen enemics al voltant seu amb l'objectiu d'atacar al personatge.

Un cop el jugador s'acosti a un extrem del mapa on esta jugant, se li donarà l'oportunitat de triar la pròxima població que vulgui visitar de les disponibles en el moment.

En el cas que els enemics acabin amb la vida del jugador, apareixerà un menú on el jugador podrà triar si començar una altre partida o sortir del joc.

Un factor comú dels videojocs de supervivència és que solen tenir un objectiu final, la raó per sobreviure, com pot ser una història sobre el personatge o l'entorn o acabar amb un enemic final. Algun d'aquests exemples pot ser el videojoc The Forest, Figura 51, on el personatge té la missió de trobar el seu fill després d'un accident d'avió. En el cas del nostre videojoc, no disposa d'un objectiu final ja que la finalitat és crear un precedent de videojoc ambientat en entorns reals on, a partir d'aquest, se'n puguin crear de tot tipus. D'aquesta manera, amb aquest videojoc l'usuari podrà jugar indefinidament fins on vulgui.

Figura 51. Captura de pantalla de The Forest

8.2 DISSENY DELS ATRIBUTS

El personatge disposa de diferents atributs. Aquests són:

- **Vida:** Atribut que marca la quantitat de dany que pot rebre el personatge. Si arriba a 0 el personatge morirà.
- **Dany:** Atribut que marca la quantitat de dany que el personatge farà al atacar als enemics.
- **Rang:** Distància partint del cos del personatge on aquest podrà atacar els enemics.
- **Interval entre atacs:** Temps entre atac i atac del personatge

Els enemics disposen dels mateixos atributs que el personatge, però inicialitzats amb valors diferents.

8.3 DISSENY DEL FUNCIONAMENT

Com s'ha comentat a la descripció general [8.1], el personatge es pot moure pel mapa lliurament mentre és perseguit per una sèrie d'enemics. El jugador es trobarà amb els següents elements a l'hora de jugar.

8.3.1 Elements del mapa

Aquests són els elements funcionals i visuals que formen part del mapa.

8.3.1.1 Segments

Els Segment són la representació de les carreteres en el món real, extretes de GMP. El jugador podrà caminar i els enemics podran perseguir el jugador, sobre d'elles. Veure Figura 52.

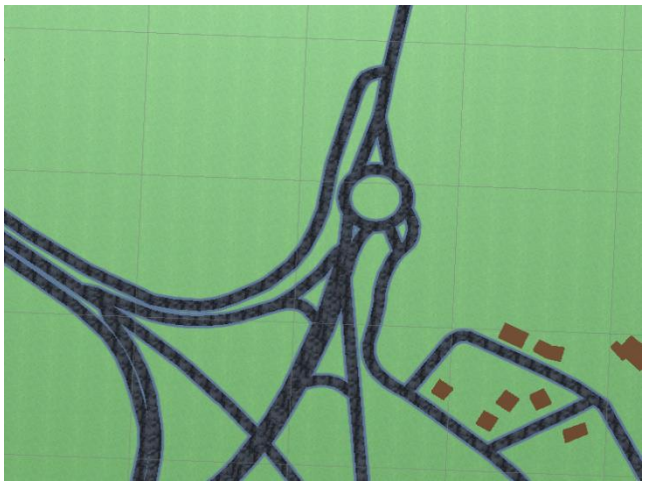

Figura 52. Exemple conjunt de segments formant una intersecció

8.3.1.2 Regions

Les regions són la representació d'entorns públics o privats, com poden ser parcs, camps de futbol, pistes d'atletisme, etc. Són extretes de GMP i els enemics podran perseguir el jugador sobre les diferents regions. Veure Figura 53.

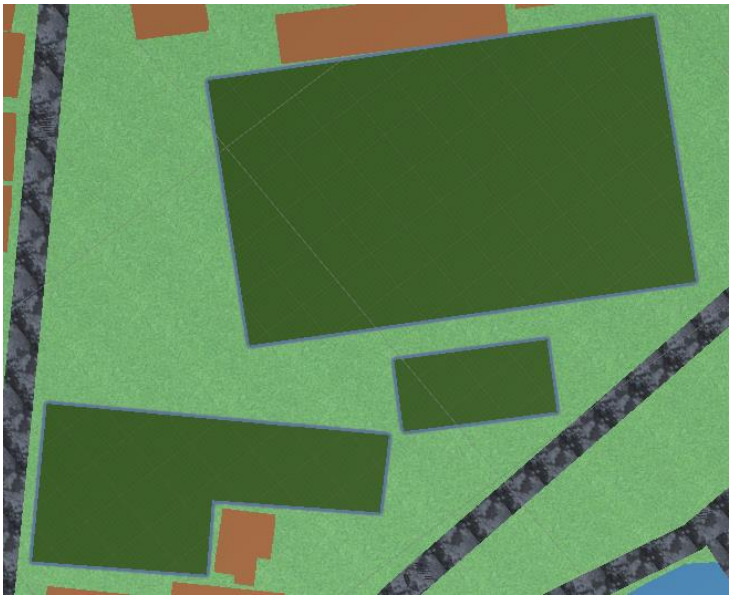

Figura 53. Exemple Regions

8.3.1.3 Structures

Les structures són la representació de les edificacions que es troben a la població, extretes de GMP. Els enemics no podran passar pel mig de les sctructures i gràcies a la NavMesh, tampoc s'hi acostaran més d'una certa distància. Aquests elements es divideixen en dos subcategories:

8.3.1.3.1 Modeled Structures

Les modeled structures són els tipus de edificacions o elements de la població que tenen una forma complexa i no poden ser representats amb una estructura més simple. Per exemple l'estàtua de la llibertat, la torre Eiffel, la Sagrada Família. Veure Figura 54.

Figura 54. Exemple Modeled Structure

8.3.1.3.2 Extruded Structures

Les extruded structures són aquells tipus de edificacions que es poden representar a partir d'una planta i, a partir d'aquí, extrudir-les fins a l'altura adequada. Les extruded structures formen la major part de totes les poblacions. Veure Figura 55.

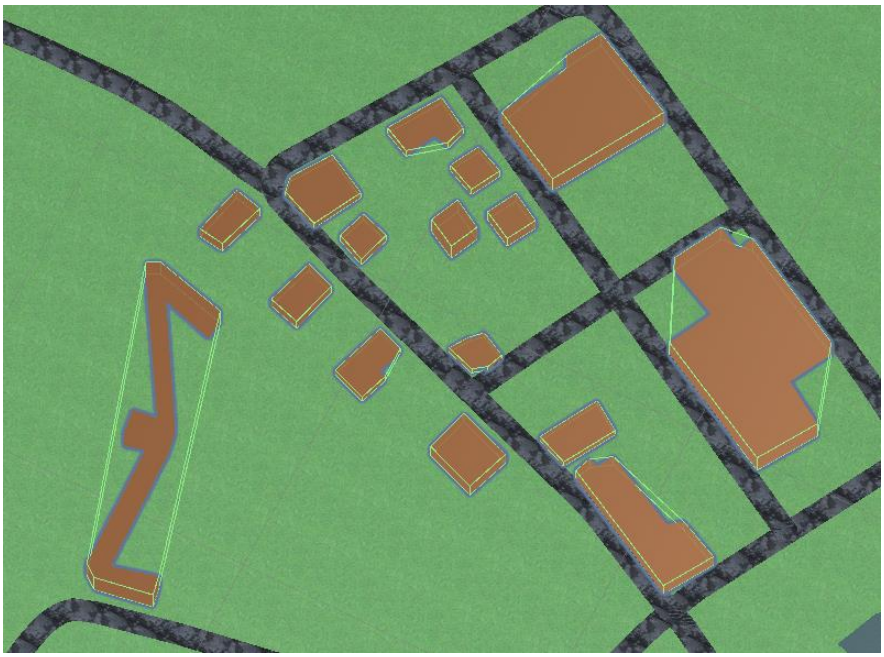

Figura 55. Exemple d'un veïnat de Extruded Structures

8.3.1.4 Area Water

Les àrees d'aigua són extretes de GMP. Són aquelles zones on l'aigua és predominant i el conjunt de d'aquests elements formen cossos d'aigua, com poden ser estanys, llacs, mars i oceans. Ni el personatge ni els enemics poden moure's per aquests elements. Veure Figura 56.

Figura 56. Exemple conjunt d'Area Water

8.3.1.5 Line Water

Les line wàter són cossos d'aigua més petits com poden ser rius i recs. Tant el personatge com els enemics es poden moure a través d'aquests elements. Veure Figura 57.

Figura 57. Exemple de dos conjunts de Line Water

8.3.1.6 Terra

El terra és creat a partir de la mida del mapa, ja que no és extret de GMP. Aquest, com és d'esperar, està pensat per cobrir les zones que no cobreixen els altres elements extrets de GMP.

El personatge i els enemics es podran moure per aquest element. Veure Figura 58.

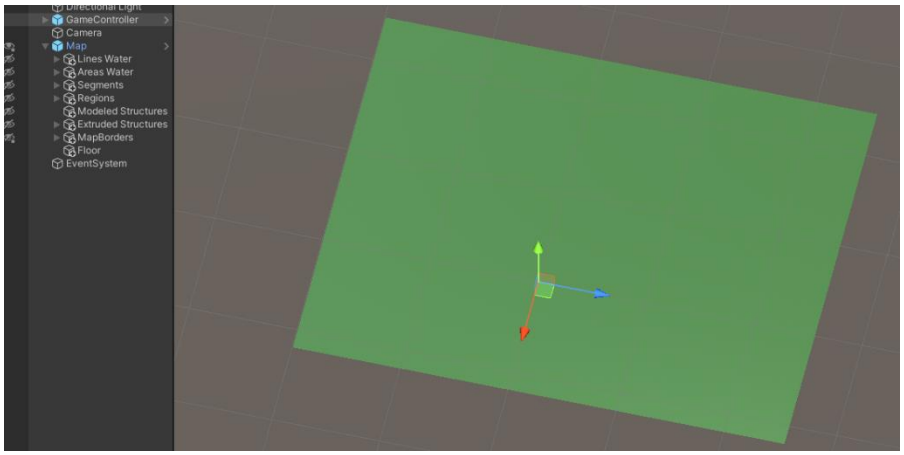

Figura 58. GameObject del terra amb la resta del mapa ocult

8.3.1.7 Frontera del mapa

Aquest element també és creat a partir de la mida del mapa ja que, igual que el terra, tampoc és extret de GMP. La seva raó d'existir és mantenir el mapa i el personatge dins un espai limitat i quan el jugador en surti, es mostri el mapa per canviar de població. Veure Figura 59.

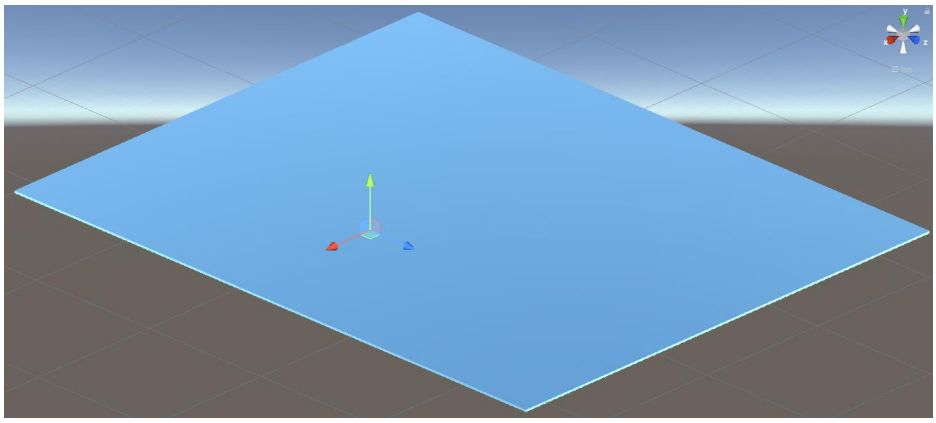

Figura 59. GameObject de la frontera del mapa

8.3.2 Enemics

Els enemics apareixeran entre una distància mínima i màxima del personatge per així assegurar que no apareixen enemics massa a prop i que, per tant, sigui injust o pel contrari, que no apareguin al costat oposat del mapa i hagin de recórrer tot el mapa per arribar al personatge.

Els enemics, a l'hora d'aparèixer, utilitzen segments, però pot ser possible que, per la forma del segment, apareguin a àrees d'aigua o a l'interior de estructures. En aquests casos l'enemic s'esborrarà donant la oportunitat a un altre enemic a aparèixer.

Tots els tipus d'enemic tenen el mateix objectiu, el jugador, i el persegueixen de la mateixa manera. Tot i així, en el videojoc poden aparèixer 4 tipus d'enemics diferents. Veure Figures 60, 61, 62 i 62.

Figura 60. Model Enemic 1

Figura 61. Model Enemic 2

Figura 62. Model Enemic 3

Figura 63. Model Enemic 4

8.3.3 Interfícies d'usuari

En el nostre videojoc es poden trobar diferents interfícies d'usuari. Les interfícies d'usuari són elements que serveixen per mostrar informació a l'usuari. En el videojoc hi ha les següents:

8.3.3.1 Menú principal

El menú principal és el primer menú que el jugador es trobarà a l'obrir el joc. En aquests menú s'hi troben dos botons. Veure Figura 64.

El primer, "Play", és el botó que farà que comenci la partida. El jugador començarà a la població de Banyoles des d'on podrà jugar.

El segon, "Quit", és el botó que quan es clica, es sortirà de l'aplicació.

Figura 64. Captura de pantalla menú principal

8.3.3.2 Menú canvi de població

Aquest menú és obert quan el jugador s'acosta a un extrem del mapa. Es bloquejaran els controls per tal de el jugador no caigui o es pugui moure pel mapa amb el menú obert, però es desbloquejarà el cursor per a què el jugador pugui triar on anar.

En aquest menú apareix un mapa de Catalunya amb diferents punts d'exclamació, veure Figura 65. Quan es cliqui un d'aquests botons es portarà al jugador a una nova escena, es tornarà a bloquejar el cursor i es tornarà a habilitar el moviment.

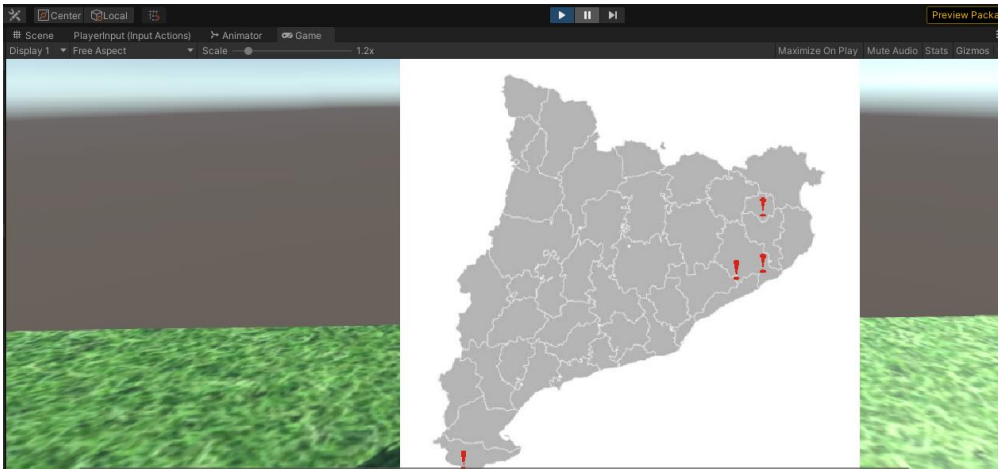

Figura 65. Captura de pantalla del menú de canvi de població

8.3.3.3 Menú usuari mort

Un cop el jugador arribi a 0 o menys punts de vida, començarà una animació del personatge morint. Un cop acabada l'animació començarà un *fade out* i s'obrirà aquest menú. Veure Figura 66.

El menú disposa de un text i dos botons. El text avisa a l'usuari que el personatge ha mort i pregunta si vol continuar. El primer dels dos botons, quan es clica, comença una nova partida carregant de nou l'escena. El segon botó, quan es clica, surt del joc. Mirar Figura 66.

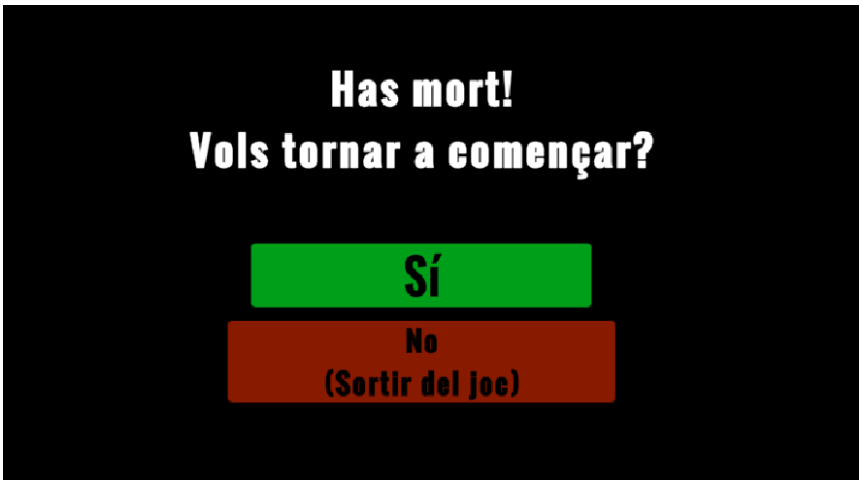

Figura 66. Menú personatge mort

8.4 DIAGRAMES I FITXES DE CAS D'ÚS

En aquest apartat es defineixen les activitats i comportaments per tal d'entendre com funciona el sistema. Un cas d'ús és una descripció dels passos que es realitzen entre l'actor i el sistema a l'hora de dur a terme una funcionalitat del sistema.

Els casos d'ús sempre són portats a terme per un actor i serveixen per especificar el comportament del sistema a través de les interaccions amb els actors.

8.4.1 Actors

Un actor és una entitat externa al propi videojoc que interactua amb ell assolint un rol o estat.

En aquest projecte, com que s'està desenvolupant un videojoc *singleplayer* sense connexió online i amb els mapes estàtics, només existeixen dos actors, el jugador i el sistema.

8.4.2 Menú principal

En aquest apartat es veuran els casos d'ús del menú principal.

8.4.2.1 Diagrama de casos d'ús menú principal

Diagrama dels casos d'ús del menú principal. Veure Figura 67.

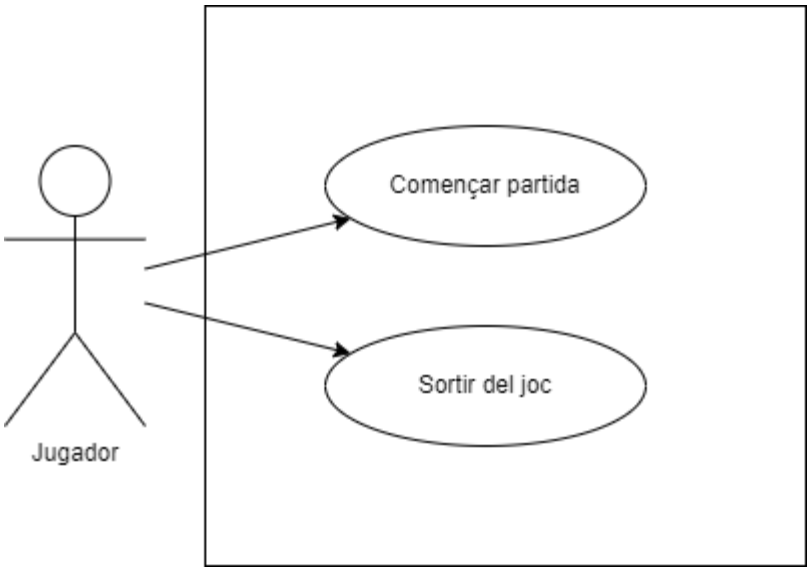

Figura 67. Diagrama de casos d'ús menú principal

8.4.2.2 Fitxes de casos d'ús menú principal

Fixa de cas d'ús	Començar partida
Descripció	El jugador selecciona el botó de començar partida
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar botó "PLAY"
Flux alternatiu	Sense flux alternatiu
Post-condició	Es canvia l'escena a l'escena d'una població

Taula 18. Fitxa cas d'ús començar partida

Fixa de cas d'ús	Sortir del joc
Descripció	El jugador selecciona el botó de sortir
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar botó "QUIT"
Flux alternatiu	Sense flux alternatiu
Post-condició	El videojoc para d'executar-se

Taula 19. Fitxa cas d'ús sortir del joc

8.4.3 Dins de la partida

Casos d'ús de les funcionalitats del videojoc durant la partida. Es divideix en dos parts, els casos d'ús amb l'actor sent el jugador i amb l'actor sent el sistema.

8.4.3.1 Diagrames de casos d'ús

8.4.3.1.1 Actor jugador

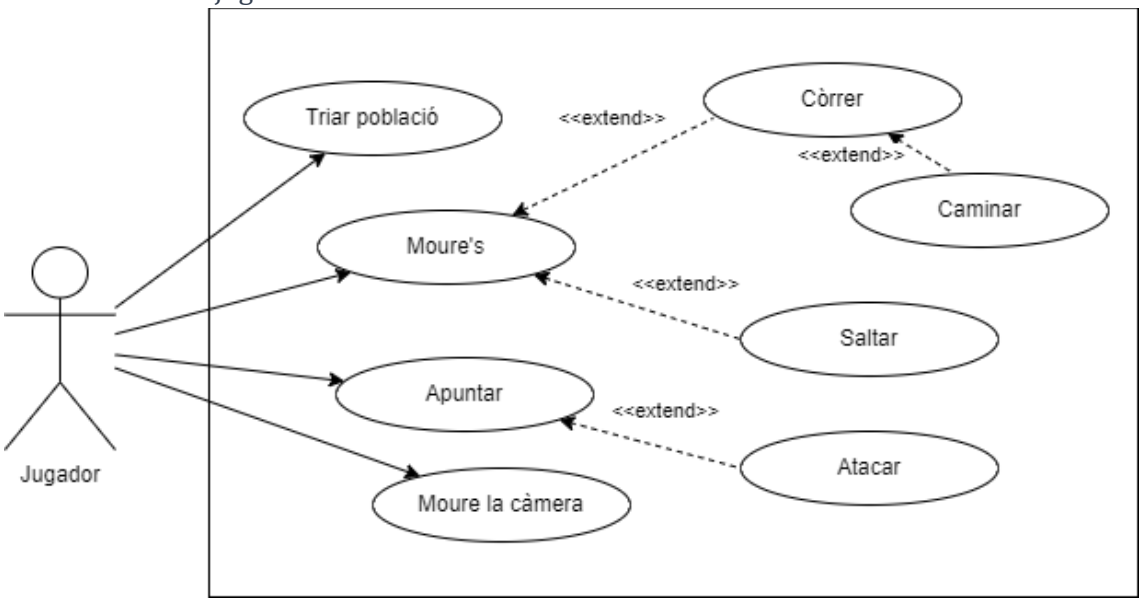

Figura 68. Diagrama de casos d'ús dins la partida del jugador

8.4.3.1.2 Actor sistema

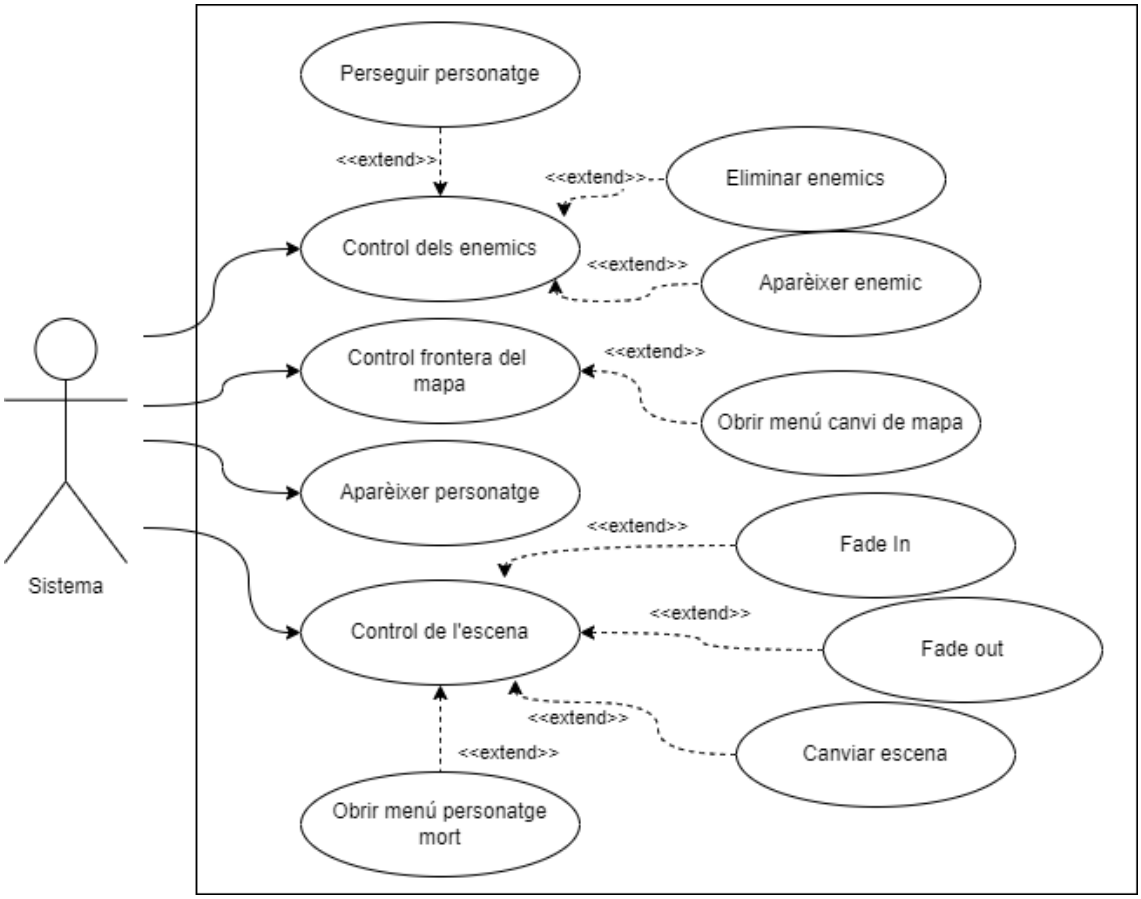

Figura 69. Diagrama casos d'ús dins la partida del sistema

8.4.3.2 Fitxes de casos d'ús

8.4.3.2.1 Actor jugador

Fixa de cas d'ús	Triar població
Descripció	El jugador tria la població on vol anar
Actors	Jugador
Pre-condició	El jugador ha creuat la frontera del mapa, s'ha obert el menú i s'ha desbloquejat el cursor
Flux principal	1. Clicar un dels botons disponibles
Flux alternatiu	Sense flux alternatiu
Post-condició	Es porta el jugador a la població que ell ha triat.

Taula 20. Fitxa cas d'ús tria població

Fixa de cas d'ús	Moure's
Descripció	El jugador es mou pel mapa
Actors	Jugador
Pre-condició	L'usuari es troba en una població
Flux principal	1. Clicar qualsevol botó de moviment
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge es mou pel mapa

Taula 21. Fitxa cas d'ús moure's

Fixa de cas d'ús	Caminar
Descripció	El personatge camina pel mapa
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar qualsevol botó de moviment
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge s'ha posat a caminar

Taula 22. Fitxa cas d'ús caminar

Fixa de cas d'ús	Córrer
Descripció	El personatge corre pel mapa
Actors	Jugador
Pre-condició	El personatge està caminant
Flux principal	1. Clicar el botó "shift" del teclat.
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge s'ha posat a córrer

Taula 23. Fitxa cas d'ús córrer

Fixa de cas d'ús	Saltar
Descripció	El personatge salta
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar botó "espai" del teclat
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge ha saltat

Taula 24. Fitxa cas d'ús saltar

Fixa de cas d'ús	Apuntar
Descripció	El personatge es prepara per atacar
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar botó dret del ratolí
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge s'ha preparat per atacar

Taula 25. Fitxa cas d'ús apuntar

Fixa de cas d'ús	Atacar
Descripció	El personatge ataca
Actors	Jugador
Pre-condició	El personatge està apuntant
Flux principal	1. Clicar botó esquerra del ratolí
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge ha atacat

Taula 26. Fitxa cas d'ús atacar

Fixa de cas d'ús	Moure la càmera
Descripció	El jugador pot moure la càmera lliurament apuntant al personatge
Actors	Jugador
Pre-condició	L'usuari es troba a l'escena del menú principal
Flux principal	1. Clicar botó central del ratolí

	2. Si la càmera està bloquejada, es desbloqueja
Flux alternatiu	3. Si la càmera està desbloquejada, es bloqueja
Post-condició	El jugador ha bloquejat o desbloquejat la càmera

Taula 27. Fitxa cas d'ús moure la càmera

8.4.3.2.2 Actor sistema

Fixa de cas d'ús	**Perseguir el personatge**
Descripció	Els enemics persegueixen el personatge
Actors	Sistema
Pre-condició	Existeixen enemics a l'escenari.
Flux principal	1. Buscar posició del personatge a l'escenari. 2. Si el personatge està viu, moure's cap a la seva direcció
Flux alternatiu	3. Si el personatge està mort, l'enemic es queda quiet
Post-condició	Els enemics es mouen cap on es troba el personatge

Taula 28. Fitxa cas d'ús perseguir el personatge

Fixa de cas d'ús	**Eliminar enemics**
Descripció	Els enemics que apareixen en llocs on no ho poden fer, són eliminats
Actors	Sistema
Pre-condició	L'enemic apareix en una part del mapa on no pot aparèixer
Flux principal	1. Es desactiva l'animador de l'enemic 2. Es desactiva el component navMeshAgent 3. Es destrueix l'enemic
Flux alternatiu	
Post-condició	S'ha esborrat l'enemic

Taula 29. Fitxa cas d'ús eliminar enemics

Fixa de cas d'ús	**Aparèixer enemics**
Descripció	Apareixen enemics a prop del personatge
Actors	Sistema
Pre-condició	S'està en una població i el personatge està viu
Flux principal	1. Cada cert nombre de segons es busquen les carreteres que es troben entre dos distàncies del personatge. 2. Es comprova que no s'hagi arribat al nombre màxim d'enemics a la partida 3. Si no s'ha arribat al màxim d'enemics, es selecciona quin enemic apareixerà 4. Es selecciona aleatòriament un número entre 0 i el nombre de carreteres menys 1 on poden aparèixer enemics 5. S'instancia l'enemic a la carretera escollida
Flux alternatiu	6. Si s'ha arribat al màxim, no es continua
Post-condició	Ha aparegut un enemic a l'escena

Taula 30. Fitxa cas d'ús aparèixer enemics

Fixa de cas d'ús	**Obrir menú canvi de mapa**
Descripció	S'obra un menú per triar la pròxima població
Actors	Sistema
Pre-condició	El personatge ha sortit de la frontera del mapa
Flux principal	1. Es bloqueja el moviment del personatge 2. Es desbloqueja el cursor

	3. Es mostra el mapa
Flux alternatiu	
Post-condició	S'ha obert el menú per canviar de població

Taula 31. Fitxa cas d'ús obrir menú canvi de mapa

Fixa de cas d'ús	**Aparèixer pesonatge**
Descripció	Es fa aparèixer el personatge al mapa
Actors	Sistema
Pre-condició	La població s'acaba de carregar
Flux principal	1. Es busquen totes les carreteres del mapa i es guarden en una llista 2. Es selecciona un número entre 0 i el nombre de carreteres menys 1 aleatòriament 3. S'instancia el personatge a la posició de la carretera seleccionada
Flux alternatiu	
Post-condició	El personatge ha aparegut en el mapa

Taula 32. Fitxa cas d'ús aparèixer personatge

Fixa de cas d'ús	**Fade in**
Descripció	Es fa un fade in
Actors	Sistema
Pre-condició	S'acaba de carregar una població
Flux principal	1. S'activa l'animació del fade in
Flux alternatiu	
Post-condició	S'ha completat el fade in i el joc és totalment visible

Taula 103. Fitxa cas d'ús fade in

Fixa de cas d'ús	**Fade out**
Descripció	Es fa un fade out
Actors	Sistema
Pre-condició	L'animació de l'usuari morint s'ha completat
Flux principal	1. Comença l'animació del fade out
Flux alternatiu	
Post-condició	S'ha completat el fade out i el joc no es pot veure.

Taula 34. Fitxa cas d'ús fade out

Fixa de cas d'ús	**Canviar escena**
Descripció	Es canvia de població
Actors	Sistema
Pre-condició	L'usuari ha clicat el botó de la pròxima població on vol anar
Flux principal	1. Es comprova que l'escena on es vol anar no sigui la que ja s'està 2. Si la població és diferent, es canvia d'escena a la nova població
Flux alternatiu	3. Si la població és la mateixa, es deixa tornar a triar a l'usuari
Post-condició	S'ha carregat una nova escena amb una població diferent

Taula 35. Fitxa cas d'ús canviar d'escena

Fixa de cas d'ús	**Obrir menú personatge mort**
Descripció	S'obra un menú informant de que el personatge ha mort, mostrant les opcions que es poden escollir

Actors	Sistema
Pre-condició	L'animació del fade out s'ha acabat
Flux principal	1. Es desbloqueja el cursor 2. S'activa el menú per tal que sigui visible i s'hi pugui interactuar
Flux alternatiu	
Post-condició	S'ha obert el menú de personatge mort

Taula 36. Fitxa cas d'ús obrir menú personatge mort

8.4.4 Final de la partida

8.4.4.1 Diagrama de casos d'ús

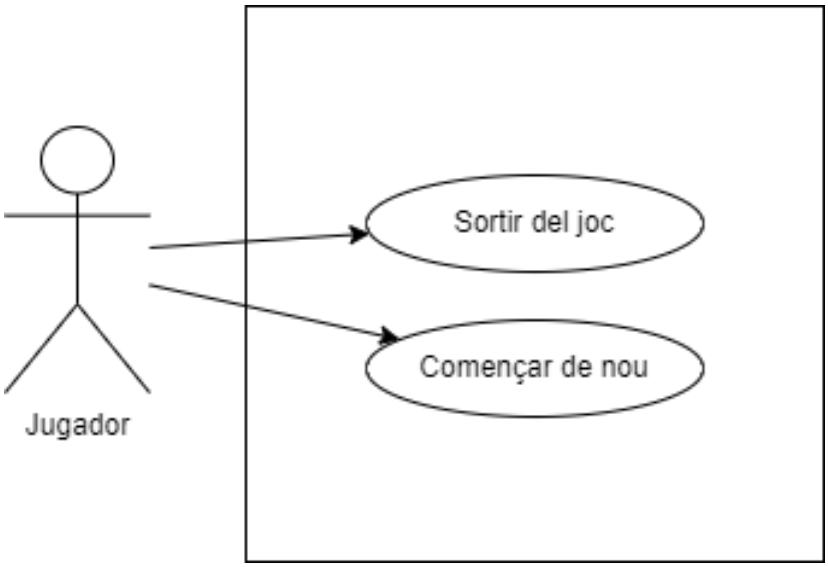

Figura 70. Diagrama de cas d'ús del final de la partida

8.4.4.2 Fitxes de casos d'ús

Fixa de cas d'ús	**Sortir del joc**
Descripció	El jugador surt del joc
Actors	Jugador
Pre-condició	El personatge s'ha quedat sense punts de vida
Flux principal	1. Clicar botó "QUIT"
Flux alternatiu	Sense flux alternatiu
Post-condició	El videojoc para d'executar-se

Taula 37. Fitxa cas d'ús sortir del joc

Fixa de cas d'ús	**Començar de nou**
Descripció	El jugador selecciona el botó de sortir
Actors	Jugador
Pre-condició	El personatge s'ha quedat sense punts de vida
Flux principal	1. Clicar el botó de continuar
Flux alternatiu	Sense flux alternatiu
Post-condició	El personatge torna a aparèixer a l'escena

Taula 38. Fitxa cas d'ús comneçar de nou

8.5 DIAGRAMES D'ACTIVITAT

Els diagrames d'activitat són la representació del flux d'accions que es realitzen en un procés, generalment dins del marc d'un o varis casos d'ús. Mostren en quin ordre s'executen les accions, com depenen les unes de les altres i qui les du a terme.

8.5.1 Menú principal

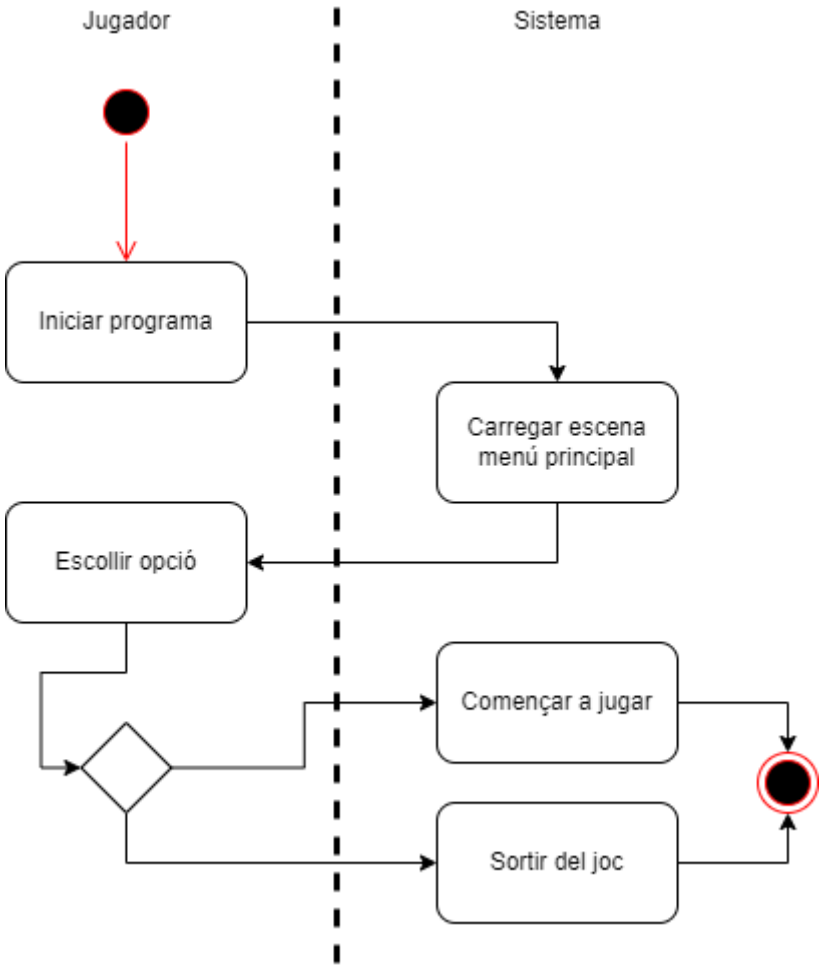

Figura 71. Diagrama d'activitats del menú principal

8.5.2 Partida

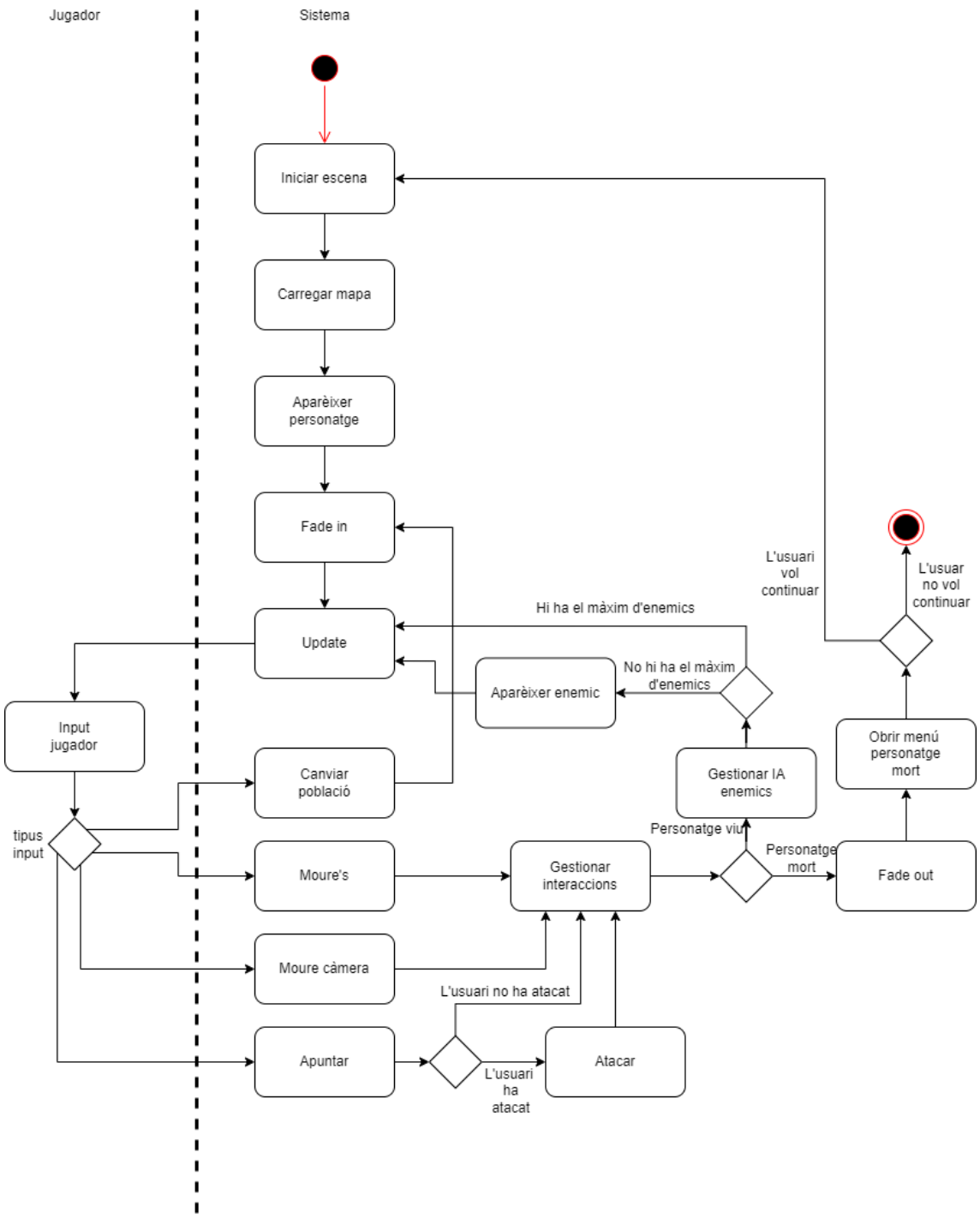

Figura 72. Diagrama d'activitats de la partida

8.6 CLASSES I MÈTODES

A continuació es mostrarà el diagrama de classes representats de manera senzilla seguit d'una explicació més detallada de cada una de elles.

8.6.1 Diagrama de classes

Primer de tot el diagrama de classes. Les classes són una plantilla per crear objectes. Una classe està formada per diferents atributs i mètodes (funcionalitats de la classe). Aspectes importants de les classes en C#, és que poden ser polimòrfiques i poden heretar mètodes d'altres classes.

L'herència en C# és la capacitat que classes filles d'una altra classe, puguin utilitzar mètodes declarats a la classe pare. Un clar exemple d'herència en aquest projecte és la classe MonoBehaviour, classe base de Unity de la que no es pot prescindir a l'hora de crear scripts que tinguin una funcionalitat en el videojoc.

El polimorfisme és la capacitat que un mètode declarat en una classe, en les classes que hereten d'aquesta primera, es pot sobreescriure el comportament utilitzant el terme *override* a l'hora de la declaració del mètode. En el cas d'aquest projecte, s'ha utilitzat el polimorfisme pels scripts d'editor, explicats en un pròxim apartat.

A continuació podem veure el diagrama de classes fet de manera general, Figura 73, és a dir, sense mètodes ni atributs. En apartats següents es desglossaran les classes per aquests elements on també s'explicaran amb detall. Només es mostraran les classes implementades durant el desenvolupament, no es mostraran classes bases ni classes generades per Unity. Els mètodes utilitzats en aquestes classes de Unity, són explicats a la secció 7.2.4 d'aquest document.

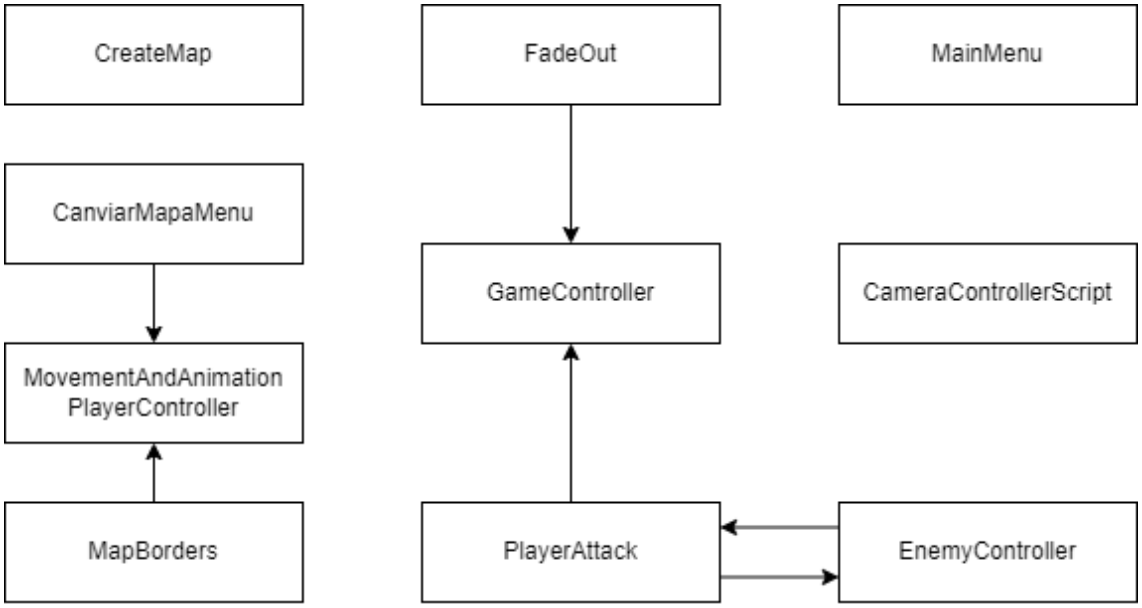

Figura 73. Diagrama de classes

8.6.2 Classe GameController

Aquesta classe s'ocupa del comportament del videojoc. És un component del GameObject amb el mateix nom.

8.6.2.1 Atributs

- **public GameObject personatge:** Prefab del personatge per poder-lo fer aparèixer.
- **public GameObject enemy1:** Prefab del primer tipus d'enemic.
- **public GameObject enemy2:** Prefab del segon tipus d'enemic.
- **public GameObject enemy3:** Prefab del tercer tipus d'enemic.
- **public GameObject enemy4:** Prefab del quart tipus d'enemic.
- **GameObject player:** Variable on es guarda l'instancia del personatge quan apareix en el mapa.
- **private List<Collider> spawnableRoads:** Llista de les carreteres on poden aparèixer els enemics.
- **public float maxEnemies:** Nombre màxim d'enemics que hi pot haver en el mapa a la vegada.
- **public float spawnRate:** Nombre de segons en el que es mirarà si hi ha suficients enemics en el mapa i si no s'ha arribat al màxim, fer-ne aparèixer.
- **private int currentEnemies:** Nombre d'enemics que hi ha en un moment en el mapa.
- **private bool isPlayerDead:** Variable de control per saber si el personatge està viu o mort.
- **public GameObject fadeOutPanel:** Variable on es guarda el GameObject que s'ocupa dels fade in i fade out.

8.6.2.2 Mètodes

- **void SpawnPlayer():** Mètode per fer aparèixer el personatge en el mapa.
- **void LookPlayersNearRoads():** Busca les carreteres que estan entre dos distàncies del personatge.
- **void CheckSpawnEnemies():** Mira si s'ha arribat al màxim d'enemics en el mapa, en cas que no s'hi hagi arribat, s'ocupa de fer-ne aparèixer un altre.
- **public void playerDead():** Mètode que s'ocupa de la lògica del programa quan el personatge s'ha mort.
- **public void RestartGamePlay():** Mètode que s'ocupa de tornar a començar la partida en cas de que el jugador així ho hagi decidit.
- **public void CallFadeOut():** Mètode que crida el fade out.

8.6.3 Classe MainMenu

Classe que s'ocupa del comportament del menú principal.

8.6.3.1 Atributs

Aquesta classe no disposa d'atributs.

8.6.3.2 Mètodes

- **public void PlayGame():** Aquest mètode s'ocupa de carregar una escena.
- **public void QuitGame():** Mètode que s'ocupa de tancar l'aplicació.

8.6.4 Classe CameraControllerScript

Aquesta classe s'ocupa del moviment de la càmera lliure de la que disposa el personatge.

8.6.4.1 Atributs

- **public float sensitivity:** Sensitivitat del moviment del ratolí.
- **public float smoothFactor:** Factor suavitzador del moviment del ratolí.
- **CameraInput cameraInput:** Instància de la classe CameraInput creada pel nou sistema de inputs de Unity. S'ocupa de llegir els inputs de l'usuari.
- **Vector2 currentMovementInput:** Valors de moviment del ratolí.
- **Vector2 currentZoomInput:** Valors del zoom del ratolí
- **bool isMoving:** Indica si el ratolí s'està movent.
- **private float rotationX:** Rotació en l'eix de les X.
- **private float rotationY:** Rotació en l'eix de les Y.
- **private Transform target:** Objectiu que apuntarà la càmera, al personatge.
- **public float maxFOV:** Màxim Field Of View permés.
- **public float minFOV:** Mínim Field Of View permés.

8.6.4.2 Mètodes

- **void onZoomInput(InputAction.CallbackContext context):** Mètode que és executat cada cop que comença, es fa i s'acaba el moviment del zoom del ratolí. S'ocupa de llegir el valor i aplicar-lo al zoom de la càmera.
- **void onMovementInput(InputAction.CallbackContext context):** Mètode que és executat cada cop que el ratolí fa un moviment. Llegeix el valor dels moviments del ratolí, assigna l'atribut isMoving, rotationX, rotationY i s'ocupa de la rotació de la càmera.

8.6.5 Classe PlayerAttack

Classe que s'ocupa de l'estat del personatge i dels atacs.

8.6.5.1 Atributs

- **Animator animator:** Instància del component Animator del personatge
- **float aiming:** Valor que indica com de completa està l'animació d'apuntar. 0 no s'apunta, 1 s'apunta.
- **bool aimPressed:** Indica si el botó d'apuntar està clicat en un moment.
- **public float aimAcceleration:** Velocitat en que es farà l'animació d'apuntar.
- **bool attackPressed:** Indica si el botó d'atacar està clicat en un moment.
- **int aimHash:** Hash del paràmetre aim de l'animator.
- **int attackHash:** Hash del paràmetre attack de l'animator.

- **int isDeadHash:** Hash del paràmetre isDead de l'animator.
- **public Transform attackPoint:** Variable on es té la posició del GameObject que serveix per saber quina zona tindrà efecte l'atac.
- **pulbic float attackRange:** Distància del personatge on l'atac tindrà efecte.
- **public LayerMask enemyLayers:** Layer dels enemics.
- **public float attackInterval:** Variable que indica el temps que ha de passar entre atac i atac.
- **public int meleeDamage:** Valor que indica la quantitat de mal que farà el jugador al atacar.
- **float nextAttack:** Variable on es guarda l'instant de temps on es podrà tornar a atacar.
- **public int maxHealth:** Valor que indica els punts de vida màxims que té el personatge.
- **int currentHealth:** Valor que indica els punts de vida que té el personatge en un cert moment.
- **PlayerInput playerInput:** Instància de la classe PlayerInput creada pel nou sistema de inputs de Unity. S'ocupa de llegir els inputs de l'usuari.
- **private bool isDead:** Indica si el personatge s'ha mort: té 0 punts de vida.

8.6.5.2 Mètodes

- **void onPauseInput(InputAction.CallbackContext context):** S'executa cada cop que es clica el botó de Pausa.
- **void onAttackInput(InputAction.CallbackContext context):** S'executa cada cop que el jugador clica o deixa de clicar el botó d'atacar.
- **void onAimInput(InputAction.CallbackContext context):** S'executa cada vegada que el jugador clica o deixa de clicar el botó d'apuntar.
- **void changeAiming(float attackLayerWeight):** S'ocupa d'aumentar el valor de aiming en el cas necessari.
- **void lockOrResetAiming(float attackLayerWeight):** Disminueix el valor de aiming quan sigui necessari.
- **void checkAttack(bool attacking):** S'avisa a l'animator que executi l'animació d'atacar.
- **void lockOrResetAttacking(bool attacking):** S'avisa a l'animador que no es vol atacar més
- **void Attack():** S'ocupa de mirar si el personatge pot atacar.
- **public void TakeDamage(int damage):** Mètode que s'ocupa de rebre el mal causat pels enemics.
- **void Die():** Mètode que executa la lògica de quan el personatge mor.
- **public void CallFadeOut():** Crida l'animació del fade out.

8.6.6 Classe MovementAndAnimationPlayerController

Aquesta classe s'ocupa del moviment del personatge i la sincronització amb les animacions que pertoquen a l'acció que el personatge realitza.

8.6.6.1 Atributs

- **PlayerInput playerInput:** Instància de la classe PlayerInput creada pel nou sistema de inputs de Unity. S'ocupa de llegir els inputs de l'usuari.
- **CharacterController characterController:** Instància del component CharacterController del personatge.

- **bool isGrounded:** Indica si el personatge està tocant el terra.
- **float gravityValue:** Valor de la gravetat.
- **public float jumpSpeed:** Valor de la velocitat vertical que se li donarà al personatge al saltar.
- **public GameObject freeCamera:** GameObject de la càmera lliure del personatge.
- **public GameObject staticCamera:** GameObject de la càmera estàtica del personatge.
- **Vector2 currentMovementInput:** Valors de l'input de l'usuari.
- **Vector3 currentMovement:** Valors de currentMovementInput adaptats a moviment caminant.
- **Vector3 currentRunMovement:** Valors de currentMovementInput adaptats a moviment corrent.
- **bool isMovementPressed:** Indica si hi ha un input de qualsevol moviment del personatge.
- **Animator animator:** Instància de l'animator.
- **public float acceleration:** Velocitat en la que es faran les animacions.
- **public float deceleration:** Velocitat en la que es deixaran de fer les animacions.
- **public float maximumWalkVelocity:** Velocitat màxima que pot tenir el personatge caminant.
- **public float maximumRunVelocity:** Velocitat màxima que pot tenir el personatge corrent.
- **public float walkSpeedMultiplier:** Valor que s'utilitza per passar d'un valor d'input a valor de moviment caminant.
- **public float runMultiplier:** Valor que s'utilitza per passar d'un valor d'input a un valor de moviment corrent.
- **public float runAnimationMultiplier:** Valor per controlar la velocitat de les animacions al córrer.
- **public float turnSmoothTime:** Valor per controlar la suavitat amb la que el personatge rotarà.
- **float velocityZ:** Velocitat en l'eix de les Z del personatge.
- **float velocityX:** Velocitat en l'eix de les X del personatge.
- **bool jumpPressed:** Indica si el botó de saltar està clicat.
- **bool runPressed:** Indica si el botó de córrer està clicat.
- **bool activeCamera:** Indica si s'està utilitzant la càmera lliure o la estàtica.
- **float turnSmoothVelocity:** Velocitat de rotació del personatge.
- **int VelocityZHash:** Hash del paràmetre VelocitatZ de l'animator.
- **int VelocityXHash:** Hash del paràmetre VelocitatX de l'animator.
- **int jumpHash:** Hash del paràmetre jump de l'animator.

8.6.6.2 Mètodes

- **void onSetActiveStart(InputAction.CallbackContext context):** S'executa cada cop que es clica el botó del mig del ratolí. S'ocupa de la lògica d'alternar la càmera lliure amb la càmera estàtica.
- **void onRunInput(InputAction.CallbackContext context):** Llegeix l'input del botó de córrer.
- **void onJumpInput(InputAction.CallbackContext context):** Llegeix l'input del botó de saltar.

- **void onMovementInput(InputAction.CallbackContext context):** Llegeix l'input dels moviments.
- **void changeVelocity(bool forwardPressed, bool backPressed, bool leftPressed, bool rightPressed, bool runPressed, float currentMaxVelocity):** S'ocupa de donar pes a les animacions de moure's.
- **void lockOrResetVelocity(bool forwardPressed, bool backPressed, bool leftPressed, bool rightPressed, bool runPressed, float currentMaxVelocity):** S'ocupa de treure o mantenir pes a les animacions de moure's.
- **void checkJumping():** Mètode que s'ocupa de l'animació de saltar.
- **void handleRotation():** Mètode utilitzat per la rotació del personatge segons els input.
- **void handleJumpsAndGravity():** Mètode que s'ocupa del moviment del personatge a l'hora de saltar.
- **public void EnableMovement():** Habilita el component characterController.
- **public void DisableMovement():** Deshabilita el component characterController.
- **public void UnlockCursor():** Desbloqueja el cursor.

8.6.7 Classe EnemyController

Aquesta classe s'ocupa del comportament dels enemics.

8.6.7.1 Atributs

- **Animator animator:** Instància de l'animator.
- **int standUpKindHash:** Hash del paràmetre standUpKing de l'animator.
- **int walkKindHash:** Hash del paràmetre walkKind de l'animator.
- **int isCloseEnoughHash:** Hash del paràmetre isCloseEnough de l'animator.
- **int enemyDeadHash:** Hash del paràmetre enemyDead de l'animator.
- **int playerDeadHash:** Hash del paràmetre playerDead de l'animator.
- **int randomStandUp:** Animació que farà l'enemic a l'aparèixer.
- **int randomWalk:** Animació que utilitzarà l'enemic al caminar.
- **private NavMeshAgent navMeshAgent:** Atribut que fa de l'enemic un agent de la NavMesh.
- **GameObject movePositionTransform:** GameObject del personatge
- **public int maxHealth:** Punts de vida màxims que tindran els enemics.
- **int currentHealth:** Punts de vida que té l'enemic en un moment.
- **public Transform attackPoint:** Variable on es té la posició del GameObject que serveix per saber quina zona tindrà efecte l'atac.
- **public float attackDistance:** Distància del personatge que l'atac tindrà efecte.
- **public LayerMask enemyLayers:** Layer del personatge.
- **public float attackInterval:** Variable que indica el temps que ha de passar entre atac i atac.
- **public int attackDamage:** Valor que indica la quantitat de mal que farà el jugador al atacar.
- **float nextAttack:** Variable on es guarda l'instant de temps on es podrà tornar a atacar.
- **private bool isPlayerDead:** Indica si el personatge s'ha mort: té 0 punts de vida.

8.6.7.2 Mètodes

- **public void enableFollowing():** Mètode que s'ocupa d'habilitar el seguiment de l'agent

- **public void TakeDamage(int damage):** Mètode que s'ocupa de rebre mal.
- **public void playerDead():** S'ocupa de la lògica necessària quan el personatge es mor.
- **void Die():** Lògica de l'enemic quan arriba a 0 punts de vida.
- **void Attack():** Mètode que s'ocupa de la lògica de quan l'enemic va a atacar.

8.6.8 Classe FadeOut

Aquesta classe s'ocupa del comportament del panel responsable dels fade in i fade out, a més del menú del final de la partida.

8.6.8.1 Atributs
- **public Animator animator:** Instància de l'animator.
- **public GameObject restartMessage:** GameObject on es troba el missatge de final de partida i els botons necessaris.
- **public GameObject gameController:** GameObject on hi ha una instància del gameController.

8.6.8.2 Mètodes
- **public void FadeOutAnimation():** Mètode encarregat de l'animació del fade out.
- **public void OpenDeadMenu():** Mètode que s'ocupa del menú de quan el personatge s'ha mort.
- **public void RestartGame():** Mètode on es troba la lògica per tornar a començar la partida.
- **Public void QuitGame():** Mètode que s'ocupa de sortir de l'aplicació.
- **Public void FadeIn():** Mètode encarregat de l'animació del fade in

8.6.9 Classe CanviarMapaMenu

Aquesta classe és l'encarregada de els canvis de població que vol fer l'usuari.

8.6.9.1 Atributs
En aquesta classe no s'han necessitat atributs.

8.6.9.2 Mètodes
- **public void CanviarBanyoles():** Mètode encarregat de canivar l'escena a la de Banyoles
- **public void CanviarBreda():** Mètode encarregat de canviar l'escena a la de Breda
- **public void CanviarUlldecona():** Mètode encarregat de canviar l'escena a la de Ullecona.
- **public void CanviarVidreres():** Mètode encarregat de canviar l'escena a la de Vidreres.
- **private bool ChechCurrentScene(string nameScene):** Mètode que s'ocupa de mirar si l'escena a la que es vol canviar és la que es troba el personatge.

8.6.10 Classe MapBorders

En aquesta classe hi ha la lògica per quan el personatge surt de les fronteres del mapa.

8.6.10.1 Atributs
No s'han utilitzat atributs per aquesta classe.

8.6.10.2 Mètodes

Tampoc s'han utilitzat mètodes que no siguin els heretats per la classe MonoBehaviour.

8.7 EDITORSCRIPTS

Els scripts d'editor, o editor scripts, són aquells que permeten modificar el comportament i els aspectes visuals de Unity. Aquests scripts són molt útils quan, per exemple, només es volen executar mètodes sense haver d'executar el videojoc o es necessiten funcionalitats de les que Unity no disposa o de les que si disposa, però se'n necessita una modificació per mostrar informació rellevant al desenvolupador.

A diferència dels scripts que donen funcionalitat al joc, els scripts d'editor no hereten de la classe MonoBehaviour, sinó que hereten de la classe Editor, part de UnityEditor.

En el cas d'aquest projecte, només s'ha necessitat un script d'editor, l'script per tenir les opcions de crear els mapes en l'editor, i el mètode utilitzat és un de la classe Editor que hem modificat per tal que mostri la informació que ens interessa.

Una opció diferent per executar aquestes parts de codi era utilitzar l'etiqueta "ExecuteInEditMode" que té Unity. Aquesta etiqueta permet que el codi que s'executa quan es té el joc executant, com poden ser logs, també s'executin quan no s'està jugant i s'està a l'editor.

Tot i així, aquesta etiqueta no era la més adequada comparat amb un script d'editor. Un dels motius és que, cada cop que es volgués crear una localització diferent, s'hauria de modificar el codi. En els scripts d'editor es poden crear diferents botons on cada un serveixi per una població diferent. Per això, al final es hem decantat per aquestes modificacions de l'editor de Unity.

8.8 CREAR MAPA

Per crear el mapa, s'ha creat un script, CreateMap. Aquest script s'ocupa de llegir les dades que s'han guardat de GMP, les construeix i les instancia en el mapa. També s'ocupa d'assignar un material o textura a cada GameObject per a què es puguin diferenciar els uns dels altres.

A més, aquest script també s'ocupa de crear un terra, ja que amb només els elements de GMP, el jugador cauria ja que no disposa de terra, a més de crear les fronteres del mapa.

Per assignar textures als elements, aquests són extrets de la carpeta Resources (classe Resources explicada a l'apartat 7.2.4.12 d'aquest document) que s'ha creat al projecte de Unity.

El mètode d'aquest script és:

- **public void CrearMapa(int indexLloc)**

On el paràmetre indexLloc indica la localització que es vol crear de les disponibles.

8.9 GOOGLE MAPS PLATFORM

Figura 74. Videojoc Pokemon GO

A diferència de Unity, la plataforma de GMP, més concretament el SDK for Unity que s'ha utilitzat per aquest projecte, no ha estat pensat per ser utilitzat de la manera que s'ha fet en aquest projecte. L'SDK for Unity està pensat per desenvolupar videojocs mòbils, és a dir, on la localització de l'usuari no és fixe, com per exemple el Pokemon GO, veure Figura 74, de manera que a mesura que l'usuari es mou, es carrega més mapa. El fet de que aquesta plataforma no estigui pensada per desenvolupar videojocs com el d'aquest projecte, va suposar diferents problemes, explicats a la Secció 9.1.

L'SDK for Unity posa a disposició dels desenvolupadors una sèrie d'exemples que es poden executar en Unity. Aquests exemples ensenyen al desenvolupador com funcionen les bases de l'SDK. A partir de coordenades reals, es pot provar l'SDK a la localització que es vulgui.

En un d'aquests exemples es creava el mapa a mesura que el desenvolupador mou el "personatge" dels exemples. Vist aquest exemple, es va arribar a la conclusió que les dades havien de ser extretes d'una base de dades de Google i llavors ser transformades per posar elements en el mapa.

Tenint a disposició alguns dels scripts utilitzats en aquests exemples, alguns d'ells no són visibles pel desenvolupador, s'havia d'estudiar el funcionament de l'eina traçant les crides que realitza l'exemple, fins trobar una crida que prové d'un script que no es pot veure, o fins trobar les crides d'on s'extreuen les dades o es transformen. Una altra part que també era interessant investigar, era la part de l'elevació del terreny. Resulta que no ha estat possible extreure les dades necessàries ja que les crides i la lògica que transforma les dades en el terreny, es troben en scripts que no són visibles. Un exemple del mapa de Banyoles amb elevació, el podem veure a la Figura 75.

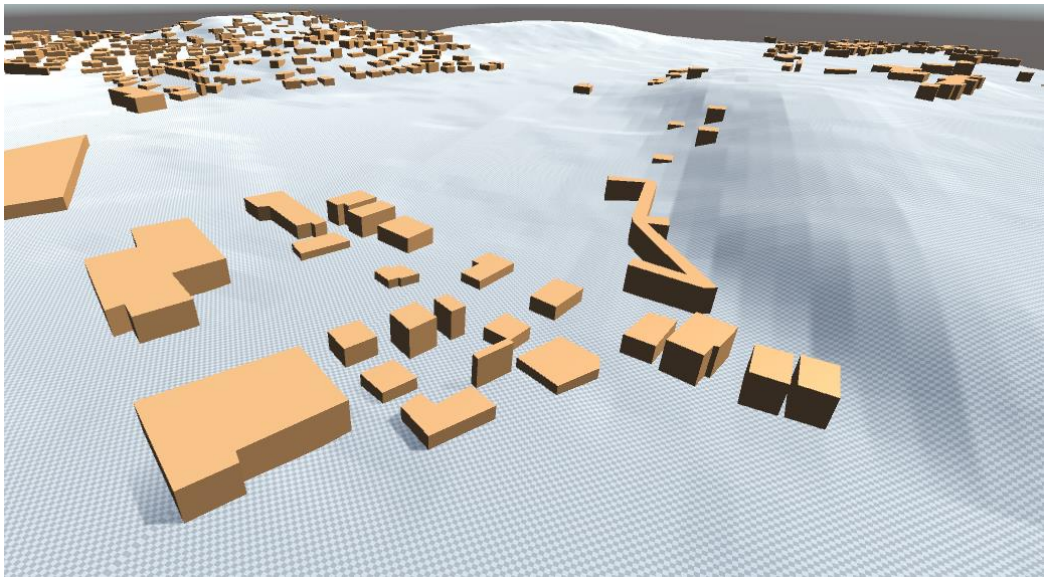

Figura 75. Mapa Banyoles amb elevació de terreny

El mètode utilitzat per extreure les dades dels GameObject és:

- **private void GetTransformAndMeshData(GameObject go, string type):** Aquest mètode és l'encarregat de extreure la posició del Transform i els vèrtexs i triangles de la Mesh. Cada una d'aquestes dades és guardada en un fitxer per separat en una carpeta diferent. El nom dels fitxers és el mateix i és el número de GameObjects que s'han guardat fins en aquest moment.

9 IMPLEMENTACIÓ I PROVES

En aquest apartat s'explicarà detalladament com s'han implementat totes i cada una de les parts del projecte explicades amb menys detall al Capítol 8 d'aquesta memòria. A més de la implementació detallada, s'explicaran els problemes que s'han trobat durant el desenvolupament i també es veuran proves que s'han realitzat per assegurar-se del correcte funcionament de la part implementada.

9.1 GOOGLE MAPS PLATFROM I DADES PEL MAPA

Com s'ha explicat a l'apartat 8.9 d'aquest document, s'han utilitzat eines de Google Maps Platform com l'SDK for Unity. Com s'ha explicat, l'SDK for Unity està pensat per desenvolupar jocs amb localitzacions dinàmiques, pel que suposa un problema pel desenvolupament d'aquest projecte ja que uns dels seus objectius és crear un videojoc per PC amb mapes estàtics. Un dels altres problemes que es presentava al utilitzar aquesta eina, és que degut a que els dispositius mòbils tenen menys recursos, no es carreguen mapes tant grans el que fa que no es carregui una població al complert. Veure la Figura 76 per un exemple de l'Estany de Banyoles sense estar carregat per complert.

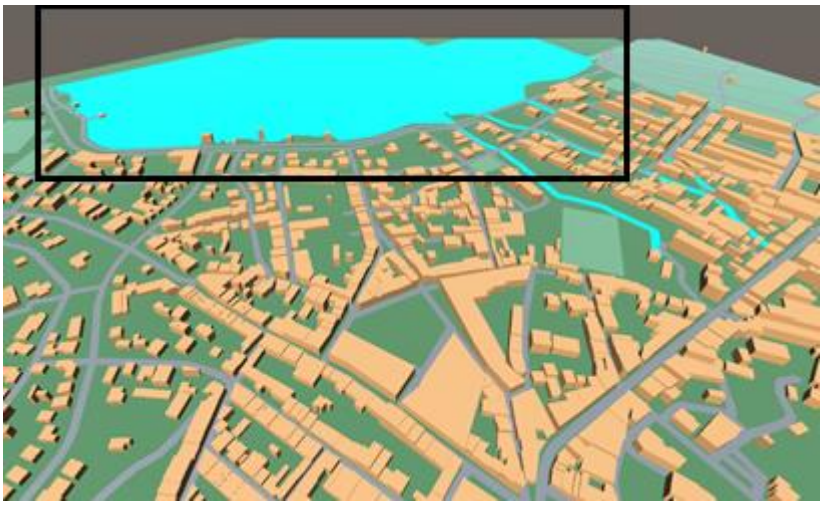

Figura 76. Mapa dinàmic amb parts sense carregar

Un altre problema de la plataforma és que només utilitzant les API, sense utilitzar l'SDK for Unity no es poden extreure les dades dels diferents elements com la forma o, en el cas de les edificacions, tampoc de la altura, l'altura del terreny, cossos d'aigua, etc, ja que al estar pensat per jugar-se en el món real no és necessari diferenciar els diferents elements i, sense l'SDK, el mapa era un pla amb els elements dibuixats per sobre.

El nou problema que es va trobar, va ser que no tots els scripts es podien visualitzar, entre ells, les crides que es feien per extreure les dades. Seguint estudiant els scripts i l'eina, es va trobar l'script on hi ha els triggers que s'executen un cop s'han creat els elements en el mapa i en els mètodes cridats dins d'aquests triggers, es veu com l'SDK accedeix al GameObject creat. Gràcies a aquest script, i el fet que es pot modificar, soluciona els problemes que s'han explicat anteriorment: com podem accedir als GameObjects que es carreguen, significa que podem accedir a l'estructura interna, com pot ser el vector de vèrtexs, el vector de triangles i el vector de posició. A més a més, cada tipus de GameObject que es carrega té el un trigger propi, pel que

també resulta fàcil saber amb quin tipus de GameObject s'està tractant. Igualment trobar aquest script soluciona el fet que el mapa que carrega la SDK pot no ser suficientment gran com per mostrar tota la població. Com que el mapa es carrega dinàmicament, tots els elements que es carreguen de nou executaran els triggers mencionat. Per tant, mentre ens movem pel mapa de l'SDK, a mesura que es van carregant zones noves, es van guardant les dades.

El fet de moure's suposa tres problemes. El primer, que la posició no fos la correcte en el cas que la posició central del mapa es modifiqués cada cop que es carrega una zona nova. Aquesta part al final no ha resultat ser un problema ja que la posició central del mapa no varia respecte es van carregant noves zones.

El segon problema és que les zones que es carreguen són una fracció de la mida del mapa, pel que hi ha parts del mapa que queden buides ja que no s'ha arribat a carregar la zona, veure Figura 77.

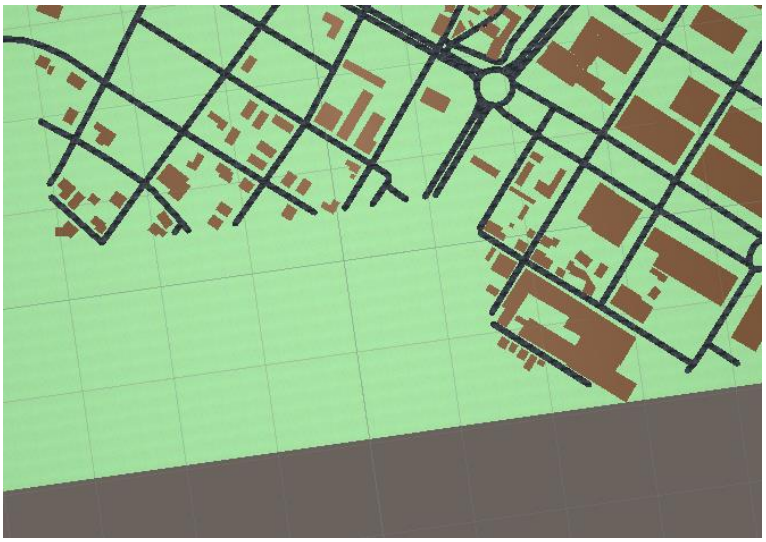

Figura 77. Extrem del mapa que no s'ha arribat a carregar

El tercer problema que suposa és, de la mateixa manera que es carrega una nova part, per estalviar recursos, altres parts es descarregaran. Això pot suposar un problema ja que si descarreguem una zona per després tornar-la a carregar, es tornen a guardar aquestes dades i no ens interessa. Tot i que no és ideal, a l'hora d'implementar el sistema d'aquesta manera, per assegurar-nos que no es guarden dades repetides, el temps per guardar una població augmentava molt ja que per cada element que es volia guardar, es mirava la posició de la resta d'elements del mateix tipus. Això feia que, com que cada cop hi havia més elements a mirar, el procés fos molt llarg i poc eficient. Després d'analitzar altres opcions, com que els mapes són estàtics i de poblacions (en comptes de comarques o províncies, que són molt més extenses), es va veure que els recursos i memòria utilitzats pels elements duplicats no són suficientment grans com per suposar un problema de rendiment. Llavors es va deixar aquest problema fins poder trobar una solució més eficient a l'hora d'extreure les dades de GMP.

Finalment, tenint en compte els problemes mencionats, aprofitant els triggers de l'script que l'SDK for Unity aporta, s'ha desenvolupat el mètode per guardar les dades. El mètode funciona tal que primer de tot guardarem el component MeshFilter en una variable. Seguidament, en el directori que volem guardar les dades, guardarem el nombre de elements que s'han guardat. D'aquesta manera, tindrem els elements ordenats per número i tots diferents.

Seguidament, es crea el fitxer on hi haurà guardada la posició del GameObject, i com que és el que haurà de guardar menys informació, també es guardarà el tipus de GameObject que és per,

a l'hora de crear el mapa, poder-lo identificar. Llavors es crearan els fitxers i es guardaran les dades corresponents als vèrtexs i triangles.

```
private void GetTransformAndMeshData(GameObject go, string type){
    MeshFilter meshFilter = go.GetComponent<MeshFilter>();

    string p = @"D:\UnityProjects\TestScripts2\Assets\TextFiles\Transforms";
    int fCount = Directory.GetFiles(p, "*", SearchOption.AllDirectories).Length;
    string pathTransforms =
Path.Combine(@"D:\UnityProjects\TestScripts2\Assets\TextFiles\Transforms",
fCount.ToString() + ".txt");
    string pathVertices =
Path.Combine(@"D:\UnityProjects\TestScripts2\Assets\TextFiles\Vertices", fCount.ToString() +
".txt");
    string pathTriangles =
Path.Combine(@"D:\UnityProjects\TestScripts2\Assets\TextFiles\Triangles", fCount.ToString()
+ ".txt");

    if(!File.Exists(pathTransforms)){
     using(StreamWriter sw = File.CreateText(pathTransforms)){
      sw.WriteLine(type);
      sw.WriteLine(go.transform.position.x.ToString());
      sw.WriteLine(go.transform.position.y.ToString());
      sw.WriteLine(go.transform.position.z.ToString());
     }
    }
    else{
     Debug.Log("Transforms Fail");
    }

    if(!File.Exists(pathVertices)){
     using (StreamWriter sw = File.CreateText(pathVertices)){
      for(var i = 0; i<meshFilter.mesh.vertices.Length; i++){
       sw.WriteLine(meshFilter.mesh.vertices[i].x.ToString());
       sw.WriteLine(meshFilter.mesh.vertices[i].y.ToString());
       sw.WriteLine(meshFilter.mesh.vertices[i].z.ToString());
      }
     }
    }
    else{
     Debug.Log("Vertices Fail");
    }
```

```
if(!File.Exists(pathTriangles)){
      using (StreamWriter sw = File.CreateText(pathTriangles)){
        for(var i = 0; i<meshFilter.mesh.triangles.Length; i++){
          sw.WriteLine(meshFilter.mesh.triangles[i].ToString());
        }
       }
    }
    else{
      Debug.Log("Triangles Fail");
    }
 }
```

El mètode anterior, com s'ha explicat, és cridat des dels mètodes que utilitza l'SDK for Unity un cop s'han creat els GameObject. Abans de executar l'SDK for Unity, es pot veure com a la jerarquia hi ha GameObjects que tenen de nom el tipus i, el que fan els mètodes que són cridats per l'SDK és assignar com a pare aquest GameObject que ja estava creat. Per aquest projecte, s'han aprofitat aquest mètodes per just després, cridar el nostre mètode GetTransformAndMeshData on, com que ja sabem que cada tipus de GameObject té un mètode diferent, li passem una string amb el tipus de GameObject que es guarda.

```
void OnExtrudedStructureCreated(DidCreateExtrudedStructureArgs args) {
  if (ExtrudedStructuresContainer != null) {
          args.GameObject.transform.SetParent(ExtrudedStructuresContainer.transform, true);
    GetTransformAndMeshData(args.GameObject, "es");
  }
}

void OnModeledStructureCreated(DidCreateExtrudedStructureArgs args) {
  if (ModeledStructuresContainer != null) {
          args.GameObject.transform.SetParent(ModeledStructuresContainer.transform, true);
    GetTransformAndMeshData(args.GameObject, "ms");
  }
}

void OnRegionCreated(DidCreateExtrudedStructureArgs args) {
  if (RegionContainer != null) {
          args.GameObject.transform.SetParent(RegionContainer.transform, true);
    GetTransformAndMeshData(args.GameObject, "r");
  }
}

void OnSegmentCreated(DidCreateExtrudedStructureArgs args) {
  if (SegmentCContainer != null) {
          args.GameObject.transform.SetParent(SegmentCContainer.transform, true);
    GetTransformAndMeshData(args.GameObject, "s");
  }
}
```

```
void OnAreaWaterCreated(DidCreateExtrudedStructureArgs args) {
  if (AreaWaterContainer != null) {
        args.GameObject.transform.SetParent(AreaWaterContainer .transform, true);
    GetTransformAndMeshData(args.GameObject, "aw");
  }
}

void OnLineWaterCreated(DidCreateExtrudedStructureArgs args) {
  if (LineWaterContainer != null) {
        args.GameObject.transform.SetParent(LineWaterContainer .transform, true);
    GetTransformAndMeshData(args.GameObject, "lw");
  }
}
```

9.2 SCRIPTS D'EDITOR

Els scripts d'editor, explicats a l'apartat 8.7, han estat utilitzats en aquest projecte per poder executar codi des del mode editor de Unity, en comptes de executant el videojoc. La raó és que, quan s'executa un script en el videojoc, encara que es realitzin canvis en el mapa, o a la resta dels GameObjects o components, aquests canvis no es mantenen. En el cas d'aquest projecte, és interessant tenir el mapa disponible en el mode editor, per exemple per poder realitzar modificacions manualment que, en general són més fàcils d'aplicar manualment que a partir d'un script. El fet de que sigui més fàcil fer canvis en el mapa és molt útil ja que es poden fer canvis per veure com queden visualment per veure si el que es vol aplicar és factible, o per el contrari, no serveix. En el cas de que els canvis fets es volen aplicar a la resta del mapa, llavors, en comptes d'anar GameObject en GameObject aplicant els canvis, s'apliquen els canvis a l'script que crea el mapa i es crea de nou.

En el cas d'aquest projecte s'ha modificat el component CreateMap que es troba en el prefab Map. El que s'ha modificat de l'editor ha estat que, com que per defecte els components script, si no existeixen variables públiques, estan buits, s'han afegit 4 botons, un per cada població que es podrà jugar, veure Figura 78.

Figura 78. Botons afegits al component

Per aconseguir aquests botons s'ha modificat el mètode OnInspecotrGUI(), explicat a l'Apartat 7.2.4.13, que forma part de la classe Editor de Unity. Primer de tot es necessita una etiqueta per a què Unity sàpiga quin component, o en el nostre cas, quin script estem modificant. Es segueix el codi creant la classe que hereta de Editor i dins d'aquesta classe es modifica el mètode utilitzant el polimorfisme que ens ofereix C#.

Tot i que l'script ja sap quin component es vol modificar, és un tipus genèric. Per tant, per fer les modificacions es necessita guardar el component en una variable del tipus corresponent utilitzant la referència de Unity, target, amb un cast.

Seguidament es creen els botons que es necessiten passant el nom que volem que tingui el botó. Quan es clica un botó, el mètode Button retorna un bool, per tant encapsularem la creació del botó en un if per, quan el botó sigui clicat, s'executi el codi pertinent.

```
[CustomEditor(typeof(CreateMap))]
public class CreateMapEditor : Editor
{
   public override void OnInspectorGUI(){
      CreateMap scriptCrearMapa = (CreateMap) target;
      if(GUILayout.Button("Crear Mapa Banyoles")){
         scriptCrearMapa.CrearMapa(1);
      }
      if(GUILayout.Button("Crear Mapa Breda")){
         scriptCrearMapa.CrearMapa(2);
      }
      if(GUILayout.Button("Crear Mapa Ulldecona")){
         scriptCrearMapa.CrearMapa(3);
      }
      if(GUILayout.Button("Crear Mapa Vidreres")){
         scriptCrearMapa.CrearMapa(4);
      }

   }
}
```

9.3 MAPA

El mapa és un dels elements més importants del videojoc i en el cas d'aquest projecte, al ser ambientat en entorns reals, encara cobra més importància. En el mapa existeixen 8 tipus de components explicats a la Secció 8.3.1. Tots aquests elements són creats i configurats des del mateix script.

El mètode que crea el mapa és el CrearMapa, part de la classe CreateMap explicada anteriorment. El mètode és creat quan es clica qualsevol dels botons afegits a través de l'editor script i s'hi passa un paràmetre per saber quina població es vol carregar. Aquest mètode esta format per tres parts: la primera, el codi a continuació, on es troba el path dels fitxers de la població escollida, el nombre d'elements a carregar, la definició de variables per tenir el recompte del número de GameObjects que s'han creat de cada tipus, la creació dels GameObject pares i l'assignació de les editor flags per posteriorment poder crear la NavMesh.

```
string txtFilesPath = Application.dataPath;
txtFilesPath = txtFilesPath.Remove(txtFilesPath.IndexOf("Assets"));
txtFilesPath = string.Concat(txtFilesPath, "txtFiles");
switch(indexLloc){
   case 1:
      txtFilesPath = string.Concat(txtFilesPath, "\\TxtFiles_Banyoles");
      break;
   case 2:
      txtFilesPath = string.Concat(txtFilesPath, "\\TxtFiles_Breda");
      break;
   case 3:
      txtFilesPath = string.Concat(txtFilesPath, "\\TxtFiles_Ulldecona");
      break;
   case 4:
      txtFilesPath = string.Concat(txtFilesPath, "\\TxtFiles_Vidreres");
      break;
   default:
      Debug.Log("error index");
      break;
}
//Check the number of txt files for the loop
//string p = @"C:\UnityProjects\TestCrearGO\Assets\TextFiles\Transforms";
string p = Path.Combine(txtFilesPath, "Transforms");
int fCount = (Directory.GetFiles(p, "*", SearchOption.AllDirectories).Length);// /2;

int numLineWater = 0;
int numAreaWater = 0;
int numSegment = 0;
int numRegion = 0;
int numModeledStructure = 0;
int numExtrudedStructure = 0;

GameObject parentLineWater = new GameObject("Lines Water");
parentLineWater.transform.parent = gameObject.transform;
GameObject parentAreaWater = new GameObject("Areas Water");
parentAreaWater.transform.parent = gameObject.transform;
GameObject parentSegment = new GameObject("Segments");
parentSegment.transform.parent = gameObject.transform;
GameObject parentRegion = new GameObject("Regions");
parentRegion.transform.parent = gameObject.transform;
GameObject parentModeledStructure = new GameObject("Modeled Structures");
parentModeledStructure.transform.parent = gameObject.transform;
GameObject parentExtrudedStructure = new GameObject("Extruded Structures");
parentExtrudedStructure.transform.parent = gameObject.transform;
var flags = StaticEditorFlags.NavigationStatic;
```

```
GameObjectUtility.SetStaticEditorFlags(parentLineWater, flags);
GameObjectUtility.SetStaticEditorFlags(parentAreaWater, flags);
GameObjectUtility.SetStaticEditorFlags(parentSegment, flags);
GameObjectUtility.SetStaticEditorFlags(parentRegion, flags);
GameObjectUtility.SetStaticEditorFlags(parentModeledStructure, flags);
GameObjectUtility.SetStaticEditorFlags(parentExtrudedStructure, flags);
```

La segona part és el bucle on es llegeixen els fitxers del GameObject que es volen afegir en el mapa. Per tots els elements que hi hagi guardats, s'obriran els tres fitxers, el del transform, el dels triangles i el dels vèrtexs. Seguidament es crearà un GameObject buit i se li assignaran els components obligatoris en tots els GameObject del mapa: MeshFilter, MeshRenderer i MeshCollider.

Llavors, el primer que es fa és guardar el tipus de element que s'està creat, per després, en el codi, configurar-lo, seguit de la lectura de la posició on es troba i, un cop llegida, es tancarà el fitxer.

Una vegada tancat el fitxer de la posició del transform, es llegirà la primera línia del fitxer on es troben els vèrtexs i es crearà una llista de Vector3, ja que sabem que els vèrtexs es troben en una posició a l'espai. Llavors, mentre quedin línies a llegir del fitxer, es crearà un Vector3 amb la línia llegida i juntament amb les dos següents, s'afegirà a la llista creada. Amb el fitxer buit, la llista es passarà a array i s'assignarà al component vertices de la mesh. El mateix que s'ha fet amb els vèrtexs es farà amb els triangles, canviat la llista de Vector3 per una llista de int on cada int és un índex de la llista de vèrtexs. També s'ha creat una array de Vector2 on s'han assignat els UV necessaris per poder assignar textures als elements.

```
string color = transformFile.ReadLine();

//Read the tranformFile and assign the position to the new gameObject's transform.
After close the file
Vector3 v3aux = new Vector3(float.Parse(transformFile.ReadLine()),
                  float.Parse(transformFile.ReadLine()),
                  float.Parse(transformFile.ReadLine()));
go.transform.position = v3aux;
transformFile.Close();
string auxFiles = verticesFile.ReadLine();
List<Vector3> meshVertices = new List<Vector3>();
while(auxFiles != null){
      Vector3 aux = new Vector3(float.Parse(auxFiles),
      float.Parse(verticesFile.ReadLine()),
      float.Parse(verticesFile.ReadLine()));
      meshVertices.Add(aux);
      auxFiles = verticesFile.ReadLine();
}
```

```
            //Once the Vector3 array is completed, assign it to the
vertices of the mesh from the new game object. Then close the file as
it's no longer needed
            mesh.vertices = meshVertices.ToArray();
            verticesFile.Close();


            //Declare a int list for the triangles. Then read the first
line of the trianglesFile
            List<int> meshTriangles = new List<int>();
            auxFiles = trianglesFile.ReadLine();
            //While there'se still lines left in the trianglesFile, add
the read line to the int array created previously
            while(auxFiles != null){
                //meshTriangles.Add(int.Parse(auxFiles));
                meshTriangles.Add(int.Parse(auxFiles));
                auxFiles = trianglesFile.ReadLine();
            }
            //Once the int array is completed, assign it to the triangles
of the mesh from the new game object. Then close the file as it's no
longer needed
            mesh.triangles = meshTriangles.ToArray();
            trianglesFile.Close();

            Vector3[] vertices = mesh.vertices;
            Vector2[] uvs = new Vector2[vertices.Length];
            for(int j = 0; j < uvs.Length; j++){
                uvs[j] = new Vector2(vertices[j].x, vertices[j].z);
            }
            mesh.uv = uvs;
```

A continuació es configurarà el GameObject. El primer que es fa és assignar-li un nom a partir del tipus i el número d'element del mateix tipus que s'ha creat, a més d'assignar-li un tag. A continuació, per cada tipus menys per les structures, es carregarà un material amb una textura provinent de la carpeta Resources. En el cas de les structures se'ls hi assignarà un color pla, ja que el sistema de UV que s'ha fet anteriorment no serveix. Seguidament se li assigna al GameObject la posició i, depenent del tipus, es modifica el component Y per evitar problemes a l'hora de renderitzar les textures. Per últim, se'ls hi assignen els flags necessaris.

```
        if(color.Equals("lw")){
            //print("ok lw");
            go.name = "LineWater" + numLineWater.ToString();
            numLineWater++;
            go.tag = "LineWater";
            newMaterial.material = Resources.Load("Materials/LineWaterMaterial") as Material;
            //newMaterial.color = new Color(37/255f, 116/255f, 169/255f); //deep blue
            Vector3 auxV = go.transform.position;
            auxV.y -= 0.001f;
            go.transform.position = auxV;
            go.transform.parent = parentLineWater.transform;
            goColl.convex = true;
            goColl.isTrigger = true;
        }
        else if(color.Equals("aw")){
            //print("ok aw");
            go.name = "AreaWater" + numAreaWater.ToString();
            numAreaWater++;
            go.tag = "AreaWater";
            newMaterial.material = Resources.Load("Materials/AreaWaterMaterial") as Material;
            //newMaterial.color = new Color(25/255f, 181/255f, 254/255f); //light blue
            Vector3 auxV = go.transform.position;
            auxV.y -= 0.002f;
            go.transform.position = auxV;
            go.transform.parent = parentAreaWater.transform;
            goColl.convex = true;
            goColl.isTrigger = true;
        }
        else if(color.Equals("s")){
            go.name = "Segment" + numSegment.ToString();
            numSegment++;
            go.tag = "Segment";
            newMaterial.material = Resources.Load("Materials/asphalt") as Material;
            //newMaterial.color = new Color(191/255f, 191/255f, 191/255f); //gray
            Vector3 auxV = go.transform.position;
            auxV.y += 0.002f;
            go.transform.position = auxV;
            go.transform.parent = parentSegment.transform;
            GameObjectUtility.SetStaticEditorFlags(go, flags);
        }
        else if(color.Equals("r")){
            go.name = "Region" + numRegion.ToString();
            numRegion++;
            go.tag = "Region";
            newMaterial.material = Resources.Load("Materials/RegionMaterial") as Material;
            //newMaterial.color = new Color(30/255f, 130/255f, 76/255f); //green
            Vector3 auxV = go.transform.position;
            auxV.y += 0.001f;
            go.transform.position = auxV;
```

```
            go.transform.parent = parentRegion.transform;
            GameObjectUtility.SetStaticEditorFlags(go, flags);
        }
        else if(color.Equals("ms")){
            go.name = "ModeledStructure" + numModeledStructure.ToString();
            numModeledStructure++;
            go.tag = "ModeledStructure";
            Material auxMat = new Material(Shader.Find("Standard"));
            go.GetComponent<MeshRenderer>().material = auxMat;
            newMaterial.material.color = new Color(233/255f, 212/255f, 96/255f); //golden
            Vector3 auxV = go.transform.position;
            //auxV.y -= 0.6f;
            go.transform.position = auxV;
            go.transform.parent = parentModeledStructure.transform;
            GameObjectUtility.SetStaticEditorFlags(go, flags);
            goColl.convex = true;
            goColl.isTrigger = true;
        }
        else if(color.Equals("es")){
            go.name = "ExtrudedStructure" + numExtrudedStructure.ToString();
            numExtrudedStructure++;
            go.tag = "ExtrudedStructure";
            Material auxMat = new Material(Shader.Find("Standard"));
            go.GetComponent<MeshRenderer>().material = auxMat;
            newMaterial.sharedMaterial.color = new Color(228/255f, 120/255f, 51/255f);
//brown
            Vector3 auxV = go.transform.position;
            //auxV.y -= 0.6f;
            go.transform.position = auxV;
            go.transform.parent = parentExtrudedStructure.transform;
            GameObjectUtility.SetStaticEditorFlags(go, flags);
            goColl.convex = true;
            goColl.isTrigger = true;
        }
        else{
            print("Knowing type fail");
            Material auxMat = new Material(Shader.Find("Standard"));
            newMaterial.material = auxMat;
            newMaterial.material.color = new Color(0.0f, 0.0f, 0.0f);
        }
```

La tercera i última part d'aquest mètode és la part on es crea el terra del mapa i la frontera, parts que no són extretes de GMP, sinó que han de ser creades a partir de cada mapa que es carrega.

Per trobar el centre d'aquests GameObjects, primer de tot crearem dos variables, una de tipus Vector3 i l'altre de tipus Bounds. Llavors, per cada GameObject creat, en el codi són els fills dels fills del GameObject mapa, es buscaran els seus Bounds, es modificarà el centre i també la variable Bounds creada anteriorment. Un cop acabat el càlcul dels límits, el valor del centre es divideix pel número d'elements que s'han mirat per aconseguir el centre real.

```
Vector3 center = Vector3.zero;
Bounds bounds = new Bounds(center,Vector3.zero);
float numChildren = 0f;
foreach(Transform child in gameObject.transform){
  foreach(Transform grandChild in child.transform){
    Bounds grandChildBounds = grandChild.gameObject.
                                  GetComponent<Renderer>().bounds;
    center+= grandChildBounds.center;
    bounds.Encapsulate(grandChildBounds);
    numChildren++;
  }
}
center /= numChildren;
```

A partir d'aquestes dades es poden crear els dos GameObjects: El GameObject que s'ocupa de les fronteres del mapa és instanciat des de la carpeta Resources ja que és un prefab, explicat amb més detall a la Secció 7.2.2.3. En el cas del terra, es crea un objecte primitiu, un cub que seguidament li canviarem el centre i l'escala amb les dades que hem aconseguit dels bounds. Amb la mida i posició assignats, se li afegirà un component MeshCollider i se li assignarà un material que es troba a la carpeta Resources.

En el cas del GameObject responsable de les fronteres, també se li assignarà un nom, una nova posició, una nova escala, es crearà un material i se li afegirà el component MeshCollider. En el seu cas, es modificarà per a què aquest sigui un trigger i d'aquesta manera es pugui saber quan el personatge surt de les fronteres del mapa.

Per últim, s'especifica que el terra és fill del GameObject mapa i se li assignen els flags corresponents.

```
GameObject fronteraMapa = Instantiate(Resources.Load("MapBorders") as GameObject,
gameObject.transform);
GameObject terra = GameObject.CreatePrimitive(PrimitiveType.Cube);

terra.name = "Floor";
terra.transform.position = new Vector3(bounds.center.x, -0.6f, bounds.center.z);
terra.transform.localScale = new Vector3(bounds.size.x, 1, bounds.size.z);
```

```
Renderer terraMaterial = terra.GetComponent<Renderer>();// = auxMat;
terraMaterial.material = Resources.Load("Materials/FloorMaterial") as Material;
//terraMaterial.color = new Color(71/255f, 174/255f,71/255f);
terra.AddComponent<MeshCollider>();


fronteraMapa.name = "MapBorders";
fronteraMapa.transform.position = bounds.center;
fronteraMapa.transform.localScale = new Vector3(bounds.size.x-5,
                                                bounds.size.y+1, bounds.size.z-5);

Material auxMat2 = new Material(Shader.Find("Standard"));
auxMat2.SetColor("_Color", new Color(135/255f,206/255f,235/255f));
auxMat2.SetFloat("_Mode", 3);
auxMat2.SetInt("_SrcBlend", (int)UnityEngine.Rendering.BlendMode.SrcAlpha);
auxMat2.SetInt("_DstBlend", (int)UnityEngine.Rendering.BlendMode.OneMinusSrcAlpha);
auxMat2.EnableKeyword("_ALPHABLEND_ON");
auxMat2.renderQueue = 3000;

fronteraMapa.GetComponent<MeshRenderer>().material = auxMat2;
MeshCollider colli = fronteraMapa.AddComponent<MeshCollider>();
colli.convex = true;
colli.isTrigger = true;

//fronteraMapa.transform.parent = gameObject.transform;
terra.transform.parent = gameObject.transform;

GameObjectUtility.SetStaticEditorFlags(terra, flags);
```

9.4 MENÚ PRINCIPAL

Figura 79. Menú principal

Aquest és el primer menú que veurà l'usuari. Està format per dos botons i un fons de pantalla d'una població, veure Figura 79. Per fer aquest menú s'ha creat un canvas on s'han creat diferents fills, com un panel que és el que permet donar l'efecte de tenir el fons d'un color més apagat que els botos per a què aquests ressaltin més. Llavors l'altre fill, anomenat MainMenu, és on es troben els dos botons i el component script que dóna funcionalitat als botons. Cada un dels botons té encara un altre fill que és el text, veure Figura 81.

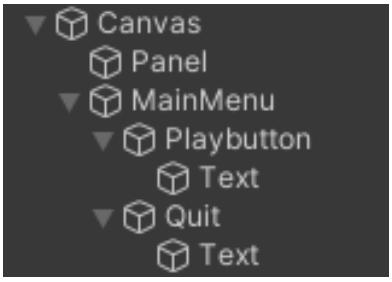

Figura 81. Jerarquia menú principal

Els botons disposen d'un component Button on, des d'allà, es poden afegir les crides que es necessiten quan aquest es clica, veure Figura 80. Els mètodes que es criden es troben a l'script MainMenu. El primer mètode d'aquest script, PlayGame, s'encarrega de carregar la primera escena de la partida.

El segon mètode és l'encarregat de tancar l'aplicació.

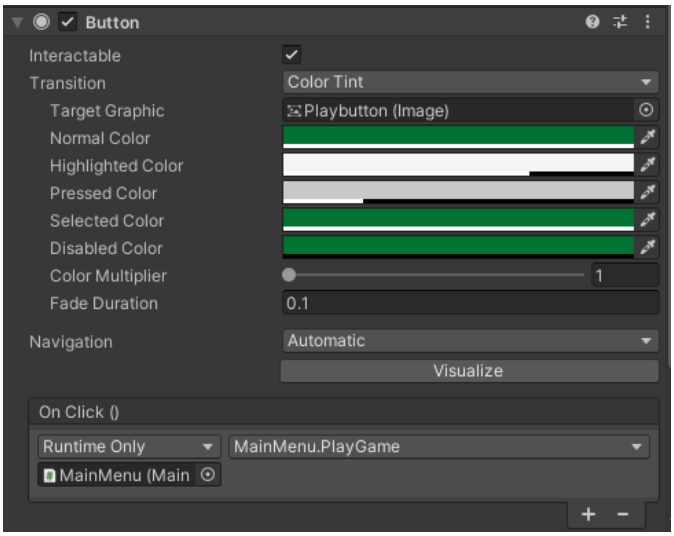

Figura 80. Component Button dels botons

9.5 MENÚ CANVI DE MAPA

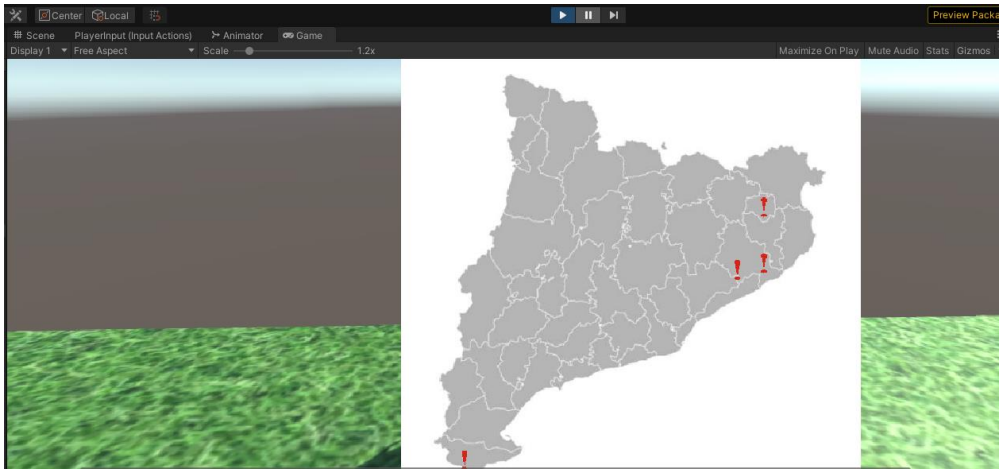

Figura 82. Menú canvi de mapa

Aquest és el menú que apareixerà al jugador quan surti de les fronteres del mapa. El GameObject del menú, amb nom MenuMapa, és fill del GameObject MapBorders. Quan apareix aquest menú, s'habilitarà el cursor i es deshabilitarà el moviment del personatge, més detallat a la Secció 9.8. El Canvas d'aquest menú se li ha afegit un panel que se li ha assignat la imatge del mapa de Catalunya i diferents botons representats amb els signes d'exclamació assignats al

component Button, veure Figura 83. Cada un d'aquests botons, quan es cliqui, cridarà un mètode de l'script CanviarMapaMenu, que és component del GameObject MenuMapa.

Els mètodes que es criden, primer comproven que no es vulgui canviar a la població que s'està actualment i, en el cas de que no s'hi estigui, canviaran de població i tornaran a habilitar el moviment del personatge.

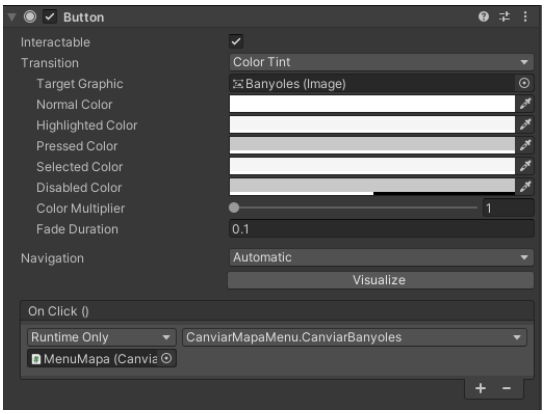

Figura 83 Component Button dels botons del menú

```
public void CanviarBanyoles(){
   if(!CheckCurrentScene("Banyoles")){
      SceneManager.LoadScene("BanyolesScene");
      EnablePlayerMovement();
   }
}

public void CanviarBreda(){
   if(!CheckCurrentScene("Breda")){
      SceneManager.LoadScene("BredaScene");
      EnablePlayerMovement();
   }
}

public void CanviarUlldecona(){
   if(!CheckCurrentScene("Ulldecona")){
      SceneManager.LoadScene("UlldeconaScene");
   }
}

public void CanviarVidreres(){
   if(!CheckCurrentScene("Vidreres")){
      SceneManager.LoadScene("VidreresScene");
      EnablePlayerMovement();
   }
}

private bool CheckCurrentScene(string nameScene){
   return SceneManager.GetActiveScene().name.Equals(nameScene);
}
```

```
private void EnablePlayerMovement(){
    GameObject playerGO = GameObject.FindWithTag("Player");
    try{
        playerGO.GetComponent<MovementAndAnimationPlayerController>()
                .EnableMovement();
    }
    catch(Exception e){
        Debug.Log(e);
    }
}
```

Per fer aparèixer aquest menú, es manté el control de si el personatge surt o no de les fronteres del mapa a partir d'un component script anomenat MapBorders en el GameObject MapBorders creat al crear el mapa. Com que MapBorders és un objecte que té al seu interior tot el mapa, i per tant també el personatge, sabem que el mapa s'ha d'obrir quan el personatge surti de aquest GameObject. Aquesta part s'aconsegueix gràcies al mètode OnTriggerExit que se li passa un Collider. Aquest collider és el del personatge. Tot i així, per seguretat i no obrir el menú quan en surt algun altre GameObject (com un enemic o en un futur, altres personatges), es comprova que el collider que ha sortit tingui el tag "Player". Si és aquest el cas, es cridaran els mètodes UnlockCursor i DisableMovement que es troben a l'script MovementAndAnimationPlayerController que s'explicarà més endavant. Després d'aquestes crides, es sap que el primer fill del GameObject MapBorders és aquest menú, per tant s'activa el primer fill del GameObject.

```
void OnTriggerExit(Collider other){
    if(other.CompareTag("Player")){
        other.gameObject.GetComponent<MovementAndAnimationPlayerController>().
                                                                UnlockCursor();
        other.gameObject.GetComponent<MovementAndAnimationPlayerController>().
                                                                DisableMovement();
        gameObject.transform.GetChild(0).gameObject.SetActive(true);
    }
}
```

9.6 MENÚ MORT

Figura 84. Menú mort

El menú de mort és el menú que apareixerà quan s'acabi l'animació de mort del personatge, veure Figura 84. Aquest menú indica a l'usuari que el personatge s'ha mort i li pregunta què és el següent que vol fer. Aquest menú està format per un canvas que té dos fills. El primer és el fons, que quan mori el personatge, realitzarà un fade out fins que es quedi totalment negre.

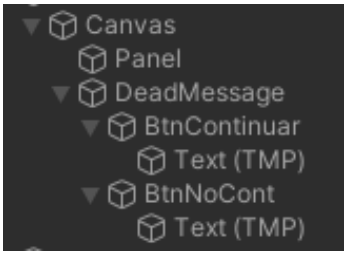

Figura 85. Jerarquia menú mort

El segon fill és el missatge, que té dos fills més que són els botons, veure Figura 85. En aquest menú, però, a l'hora de clicar un botó s'executen mètodes que es troben a l'script component FadeOut, del primer fill. La raó d'aquesta decisió és que en aquest script que s'ocupa del fade out i un cop ha acabat, mostra el missatge i els botons, veure Figura 86. Per aquest motiu, els mètodes dels botons estan en aquest script i no en un altre que sigui component del missatge o del canvas.

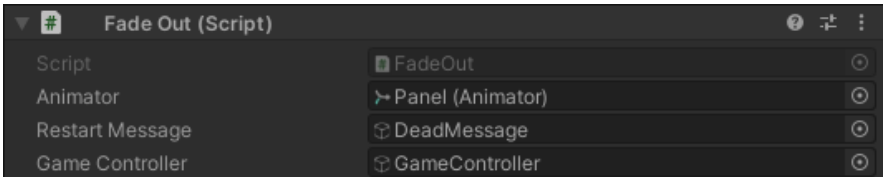

Figura 86. Component Fade Out del Panel

L'script
FadeOut, està format per diferents mètodes on tots són cridats a partir d'esdeveniments o triggers i tres atributs públics.

Els atributs públics són, el primer l'animador, que en aquest cas és el GameObject Panel, ja que no hi han animacions amb transicions complexes i les animacions que es tracten, es tracten des d'aquest script. El segon atribut és el GameObject DeadMessage el que, per defecte estarà,

desactivat i en el mètode OpenDeadMenu habilitarem. El tercer atribut és el controlador de la partida. Aquest atribut es necessita ja que, en cas de que l'usuari vulgui començar de nou, el mètode RestartGame crida un mètode d'aquest GameObject.

El primer dels mètodes és l'encarregat d'executar l'animació del fade out i és cridat quan el personatge es queda sense punts de vida. Aquesta animació està formada per dos Keyframes i un esdeveniment. El primer keyframe en el frame 0 és el Panel transparent i el segon keyframe en el frame 180 amb el Panel completament negre. Unity s'ocupa de la transició. A més en el frame 180, s'hi troba un event. Aquest esdeveniment, crida el mètode OpenDeadMenu que forma part d'aquest script. Veure Figura 87.

Figura 87. Animació Fade Out

El segon mètode, activa el cursor, ja que anteriorment estava invisible, i activa el GameObject DeadMessage que mostra el missatge de mort.

El tercer mètode, es crida quan es clica al botó "Sí", en la jerarquia BtnContinuar, és l'encarregat d'avisar al controlador de la partida que es vol començar de nou i carrega l'escena de Banyoles.

El quart mètode, es crida quan es clica el botó BtnNoCont i aquest tancarà l'aplicació del joc.

L'últim mètode, és cridat cada cop que es carrega una escena i s'ocupa de l'animació del fade in del Panel. Aquesta animació està formada per dos Keyframes, un en el frame 0 on el Panel està completament en negre i un altre Keyframe en el frame 120 on el Panel és completament transparent. Unity s'ocupa de fer la transició entre els dos estats, veure Figura 88.

Figura 88. Animació Fade In

```
public Animator animator;
public GameObject restartMessage;
public GameObject gameController;

public void FadeOutAnimation(){
   animator.Play("FadeOut");
}

public void OpenDeadMenu(){
   Cursor.lockState = CursorLockMode.None;
   restartMessage.SetActive(true);
}

public void RestartGame(){
   Debug.Log("Game restart");
   gameController.GetComponent<GameControllerScript>().RestartGamePlay();
   SceneManager.LoadScene("BanyolesScene");
}

public void QuitGame(){
   Application.Quit();
}

public void FadeIn(){
   animator.Play("FadeIn");
}
```

9.7 CONTROLADOR DE LA PARTIDA

El controlador de la partida, o GameController, és el cor del videojoc. Sense aquest GameObject, el videojoc només estaria format pel mapa, el personatge no apareixeria i, encara que el personatge ja fos a l'escena i per tant es podés moure pel mapa, no apareixerien enemics. El GameObject està format només per un component script i un GameObject fill, que és el menú de mort explicat a l'apartat anterior, que s'ocupa, entre altres, dels fade in i fade out quan apareix el personatge o es mor.

Si mirem el component script, anomenat GameControllerScript, està format per la pròpia classe heretant de MonoBehaviour i, dintre aquesta classe, la resta de la lògica necessària.

Anant per parts, el primer que hi ha a l'script són els atributs. Aquests atributs són els atributs públics on hi haurà el personatge i els enemics per poder-los fer aparèixer en el mapa, una llista de les carreteres on poden aparèixer els enemics, dos atributs públics més on hi ha el nombre màxim d'enemics i l'interval que apareixeran, seguit del nombre d'enemics que hi ha actualment en el mapa, una variable de control de si el personatge està viu o no i, per últim, el GameObject del Panel que responsable dels fade in i fade out.

```
public GameObject personatge;
public GameObject enemy1;
public GameObject enemy2;
public GameObject enemy3;
public GameObject enemy4;
GameObject player;
//private float elapsedTime = 0;
private List<Collider> spawnableRoads;
public float maxEnemies = 5.0f;
public float spawnRate = 10.0f;
private int currentEnemies = 0;

private bool isPlayerDead = false;
public GameObject fadeOutPanel;
```

Seguint amb l'script, en el mètode Start hi ha la lògica necessària per començar la partida. Primer es crearà una nova llista per guardar les carreteres on podran aparèixer els enemics. Seguidament es farà aparèixer el personatge, mètode que s'explicarà en breus, i, utilitzant el mètode InvokeRepeating, es cridaran els mètodes LookPlayersNearRoads i CheckSpawnEnemies. Per últim en aquest mètode, es cridarà el mètode FadeIn.

```
public void Start()
{
    spawnableRoads = new List<Collider>();
    SpawnPlayer();
    InvokeRepeating("LookPlayersNearRoads", 1f, 1f);
    InvokeRepeating("CheckSpawnEnemies", spawnRate, spawnRate);
    fadeOutPanel.GetComponent<FadeOut>().FadeIn();
}
```

El mètode SpawnPlayer, s'ocupa de fer aparèixer el personatge en una carretera aleatòria del mapa. Això ho fa buscant totes les carreteres que hi ha en el mapa, buscant un índex amb el mètode Random. Llavors instanciant el personatge a la carretera i per últim se li assigna al personatge una Layer i un tag necessaris.

```
void SpawnPlayer(){
    GameObject[] roads = GameObject.FindGameObjectsWithTag("Segment");
    int numberOfRoads = roads.Length;
    int spawnRoadNumber = new System.Random().Next(0, numberOfRoads);
    GameObject spawnRoad = roads[spawnRoadNumber];
    Vector3 spawnPosition = spawnRoad.transform.position;
    spawnPosition.y += 2;
    player = Instantiate(personatge, spawnPosition, Quaternion.identity);
    player.layer = 6;
    player.tag = "Player";

}
```

Seguint amb el mètode LookPlayersNearRoads, que la funció és trobar les carreteres que es troben entre una distància mínima i una distància màxima del personatge. Aquesta part s'aconsegueix gràcies el mètode OverlapSphere de Physics que, a partir de la posició del personatge i una distància, retorna tots els colliders que hi hagi. Aquesta part es fa dos cops, una per la distància mínima i una per la distància màxima.

Abans, però, com que retorna tots els colliders, s'han de filtrar les carreteres o segments.

Per últim, a l'atribut spawnableRoads, se li assignaran les carreteres que s'han trobat amb la distància més gran menys les carreteres que hi ha massa prop del personatge, amb el mètode Except.

```
void LookPlayersNearRoads(){
    Vector3 playerPos = player.transform.position;
    //Debug.Log(player.transform.position);
    Collider[] nearGameObjects = Physics.OverlapSphere(playerPos, 150);
    Collider[] tooCloseGameObjects = Physics.OverlapSphere(playerPos, 50);
    List<Collider> nearRoads = new List<Collider>();
    List<Collider> tooCloseRoads = new List<Collider>();

    for(var i = 0; i<nearGameObjects.Length; i++){
        if(nearGameObjects[i].gameObject.tag == "Segment"){
            //Debug.Log(i.ToString());
            nearRoads.Add(nearGameObjects[i]);
        }
    }
    for(var i = 0; i<tooCloseGameObjects.Length; i++){
        if(tooCloseGameObjects[i].gameObject.tag == "Segment"){
            //Debug.Log(i.ToString());
            tooCloseRoads.Add(tooCloseGameObjects[i]);
        }
    }

    spawnableRoads = nearRoads.Except(tooCloseRoads).ToList();

}
```

La funció del mètode CheckSpawnEnemies és fer aparèixer enemics en el cas de que no s'hagi arribat al màxim d'enemics en el mapa. Per tant, el primer que fa el mètode és comprovar que no s'hagi arribat a aquest màxim. Seguidament, es busca un número aleatori entre 1 i 4, per saber el tipus d'enemic que es farà aparèixer i es buscarà un número aleatori entre 1 i el nombre de carreteres que pot aparèixer per saber on apareixerà. Llavors depenent del tipus d'enemic s'instancia en el mapa i se li assigna una Layer i per últim es suma 1 al nombre d'enemics.

```
void CheckSpawnEnemies(){
    if(currentEnemies < maxEnemies){
        //Debug.Log(currentEnemies);
        int kindOfEnemy = new System.Random().Next(0, 4);
        int nSpawnableRoads = spawnableRoads.Count;
        int nSpawnerRoad = new System.Random().Next(0, nSpawnableRoads);
        Collider spawnRoad;
        try{
            spawnRoad = spawnableRoads.ElementAt(nSpawnerRoad);
            /*GameObject enemy = GameObject.CreatePrimitive(PrimitiveType.Cube);
            enemy.name = "enemy"; */
            Vector3 spawnPosition = spawnRoad.transform.position;
            spawnPosition.y += 2;
            switch(kindOfEnemy){
                case 0:
                    Instantiate(enemy1, spawnPosition, Quaternion.identity);
                    enemy1.layer = 7;
                    break;
                case 1:
                    Instantiate(enemy2, spawnPosition, Quaternion.identity);
                    enemy2.layer = 7;
                    break;
                case 2:
                    Instantiate(enemy3, spawnPosition, Quaternion.identity);
                    enemy3.layer = 7;
                    break;
                case 3:
                    Instantiate(enemy4, spawnPosition, Quaternion.identity);
                    enemy4.layer = 7;
                    break;
                default:
                    Debug.Log("No s'ha spawnejat cap enemic");
                    break;
            }
        }
        catch(Exception e){
            Debug.Log(e);
        }
        //enemy.transform.position = spawnPosition;
        currentEnemies++;
    }
}
```

Després, ens trobem amb mètodes públics que són cridats a través d'altres scripts. El primer de tots és el mètode PlayerDead que s'ocupa de la lògica quan el personatge mor. El primer que fa

és tallar els InvokeRepeat cridats al mètode Start, seguit per trobar tots els enemics del mapa i, per cada un d'ells, cridar el mètode playerDead que conté la lògica dels enemics per quan el personatge mor.

```
    public void RestartGamePlay(){
        GameObject[] enemies =
GameObject.FindGameObjectsWithTag("Enemy");
        foreach(GameObject enemic in enemies){
            Destroy(enemic);
            //Debug.Log(enemic.name);
        }
        Destroy(player);
    }

    public void CallFadeOut(){
        fadeOutPanel.GetComponent<FadeOut>().FadeOutAnimation();
    }
```

El segon mètode públic és el que conté la lògica per quan, si l'usuari ho decideix, vol tornar a començar a la partida. El que fa aquest mètode és borrar tots els enemics del mapa i per últim borrar el personatge.

El tercer i últim mètode públic, crida el mètode que s'encarrega d'executar l'animació del fade out.

```
  public void RestartGamePlay(){
    GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
    foreach(GameObject enemic in enemies){
      Destroy(enemic);
      //Debug.Log(enemic.name);
    }
    Destroy(player);
  }

  public void CallFadeOut(){
    fadeOutPanel.GetComponent<FadeOut>().FadeOutAnimation();
  }
```

9.8 PERSONATGE

El personatge és un altre dels elements indispensable per poder jugar al videojoc. El personatge, en comptes de crear-se de nou cada cop que es comença una partida, és guardat com a prefab. D'aquesta manera només farà falta instanciar-lo en el mapa i tots els canvis que se li vulguin fer, es faran en el prefab. Aquest està format per un component Animator, un character controller, un collider i dos scripts, veure Figura 89.

Figura 89. Components personatge

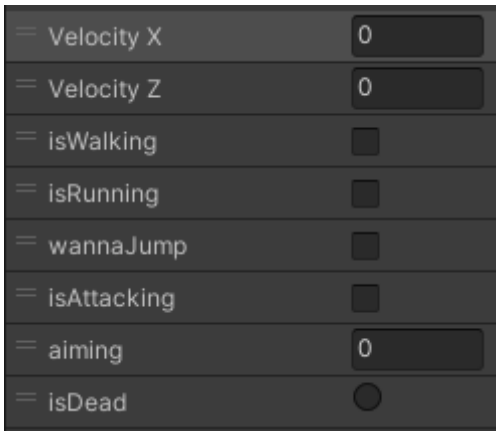

Figura 90. Paràmetres animator

L'animador del personatge està format per dos capes, la Baser Layer i la meleeAttack layer, a més de diferents paràmetres que permeten controlar el flux d'animacions. Els paràmetres són els següents: dos float (Velocity X i Velocity Y) que s'ocuparan del Blend Tree de la Base Layer, quatre bool (isWalking, isRunning, wannaJump, isAttacking) que són paràmetres de control i per transicions, un altre float (aiming) que és per controlar l'animació d'apuntar del personatge i per últim un trigger, que indica quan el personatge es mor. Veure Figura 90.

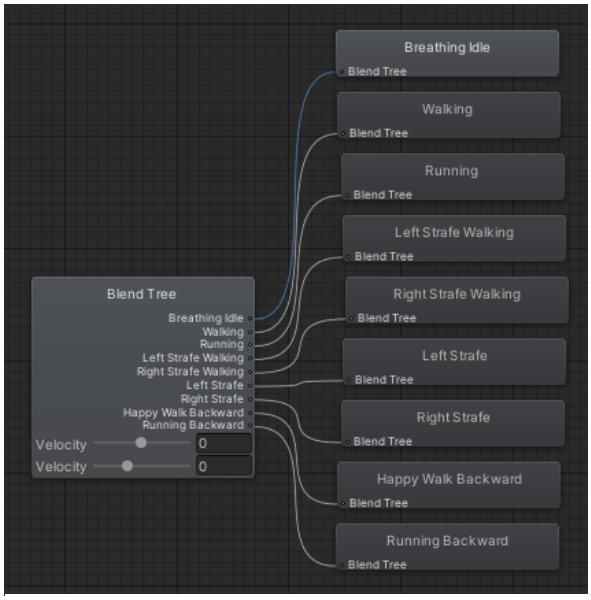

Figura 91. Blend Tree de Base Layer

El funcionament dels Blend Tree és tal que aquest disposa d'una sèrie d'animacions que, a partir del valor dels paràmetres es van combinant les unes amb les altres creant així transicions entre les animacions molt suaus sense la necessitat de disposar d'animacions extra per aconseguir aquest canvi. Els valors dels paràmetres són modificats a partir d'un script, explicat en un següent apartat. Veure Figura 91.

Per controlar quines animacions s'executen a partir d'un determinat valor dels paràmetres, a l'hora d'assignar les animacions que es volen tenir en el Blend Tree, se'ls hi assigna un valor per cada paràmetre. Com més proper sigui el valor del

paràmetre al valor que s'ha posat a l'animació, més pes tindrà aquesta respecte a la resta d'animacions, el que vol dir que aquesta serà l'animació predominant visualment, sense combinar-se amb altres animacions. A l'hora d'assignar aquests valors, Unity disposa d'un mapa de calor per veure en quina quantitat es combinaran les animacions, veure Figura 92.

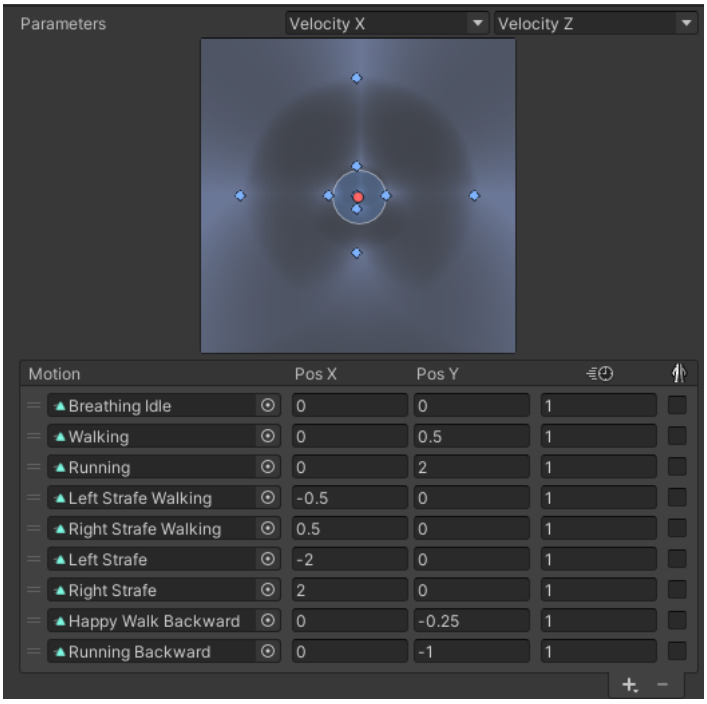

Figura 92. Animacions i valors del Blend Tree

La primera d'aquestes està formada per un Blend Tree, que és l'estat per defecte, amb una transició a l'animació de saltar. Llavors, com veiem, encara que el personatge es trobi en qualsevol altre estat, es pot executar l'animació de mort. Veure Figura 93.

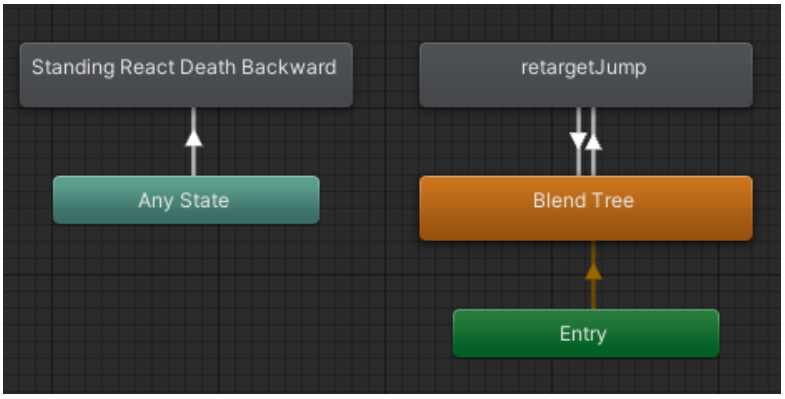

Figura 93. Distribució Base Layer

La transició del Blend Tree cap a l'animació retargetJump es du a terme quan el paràmetre wannaJump és true. La transició de tornada es du a terme quan el paràmetre wannaJump és fals i l'animació retargetJump s'ha acabat.

En canvi, la transició de AnyState a l'animació StandingReactDeathBackward, es du a terme quan el trigger isDead és activat.

Per la segona layer, meleeAttack, s'utilitzen els mateixos paràmetres però l'esquema és diferent, veure Figura 94.

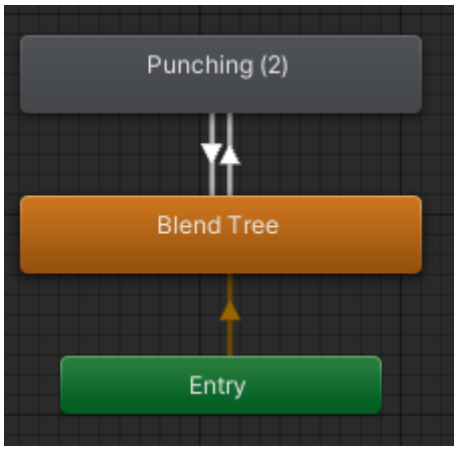

Figura 94. Distribució meleeAttack layer

Aquesta layer està formada per un Blend Tree i l'animació Punching. El Blend Tree és utilitzat per a que la transició entre el personatge sense apuntar i el personatge apuntant sigui suau, veure Figura 94. Aquesta transició és controlada amb el paràmetre aiming, veure Figura 95. I les transicions entre el Blend Tree i l'animació punching són controlades amb el paràmetre isAttacking. El funcionament d'aquestes transicions és el mateix que amb les transicions a l'hora de saltar.

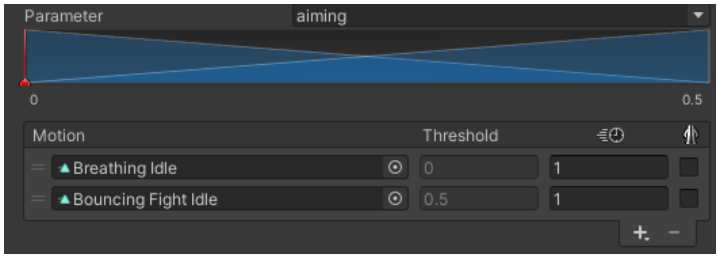

Figura 95. Animacions i valors del Blend Tree

Com s'ha vist, les Layer contenen animacions diferents. Per defecte, només es veurà la layer que s'ha creat abans, en aquest cas la Base Layer. Per tant ara s'ha de trobar una manera per fer que, quan es necessiti, es puguin veure les animacions de la meleeAttack layer. El problema és que tampoc interessa que només es vegi la meleeAttack layer, ja que volem que el personatge es pugui continuar moguent i per tant, necessitem que es mantinguin les animacions de moviment. Per aconseguir-ho, a la meleeAttack layer se l'hi ha assignat una Mask. Una Mask és un element on es poden modificar les parts del personatge que es veuran afectades per la Layer, en aquest cas, només interessa que afecti al tronc superior, veure Figura 96.

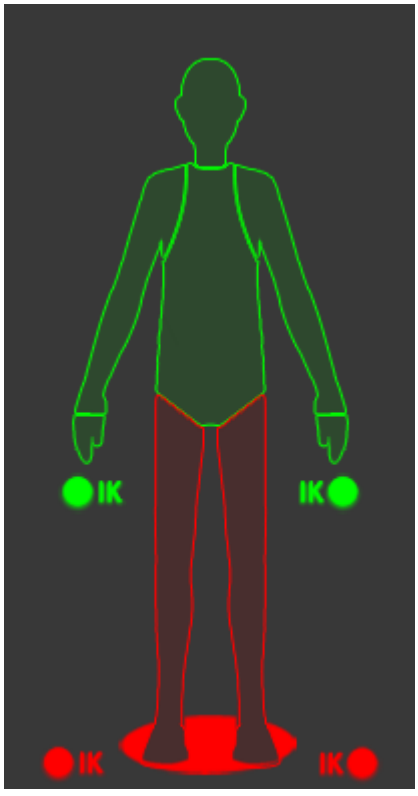

Figura 96. Mask per meleeAttack layer

Llavors per veure les animacions de la meleeAttack layer, es modificarà el pes de la layer, valor que oscil·la entre 0 i 1, siguent 0 que no té cap pes i 1 que només es veuen les animacions d'aquesta layer. Aquest pas s'explicarà més detalladament quan s'expliqui el component PlayerAttack.

El component CapsuleCollider, és una càpsula invisible que envolta al personatge i permet que aquest pugui col·lisionar amb altres GameObjects del mapa.

El component characterController és el que s'ocuparà del moviment del personatge a partir de la crida del mètode Move que es fa a l'script MovementAndAnimationPlayerController.

L'script MovementAndAnimationPlayerController, és l'encarregat de llegir els inputs del jugador respecte als moviments, modificar els paràmetres de l'animador i moure el personatge a partir del characterController. L'script està format per els diferents atributs explicats a la Secció 8.6.6.1.

```
PlayerInput playerInput;
CharacterController characterController;
bool isGrounded;
float gravityValue = -9.81f;
public float jumpSpeed = 8f;

public GameObject freeCamera;

Vector2 currentMovementInput;
Vector3 currentMovement;
Vector3 currentRunMovement;
bool isMovementPressed;

Animator animator;
public float acceleration = 2.0f;
public float deceleration = 2.0f;
public float maximumWalkVelocity = 0.5f;
public float maximumRunVelocity = 2.0f;
public float walkSpeedMultiplier = 1.5f;
public float runMultiplier = 3.0f;
public float runAnimationMultiplier = 5.0f;

public float turnSmoothTime = 1f;
public GameObject staticCamera;
float velocityZ = 0.0f;
float velocityX = 0.0f;
bool jumpPressed;
bool runPressed;
bool activeCamera; //true = static, false = free

float turnSmoothVelocity;

int VelocityZHash;
int VelocityXHash;
int jumpHash;
```

Pel que respecta als mètodes, el primer de tots, l'Awake, és l'encarregat de bloquejar el cursor, crear una instància de la classe PlayerInput, activar la càmera estàtica, guardar el component del character controller i defineix les crides que es faran quan es rebin els inputs.

El nou sistema de inputs de Unity funciona de tal manera que s'ha de crear un Action Map. Aquest action map és el conjunt d'accions amb els inputs corresponents. Les accions són el tipus de input, (ratolí, teclat, comandament de consola) i quin input s'espera (quina tecla, quin botó del ratolí, etc). Un cop s'han creat les accions, Unity genera una classe de C# que per utilitzar-la en un script, se n'ha de crear una instància. Veure Figura 97.

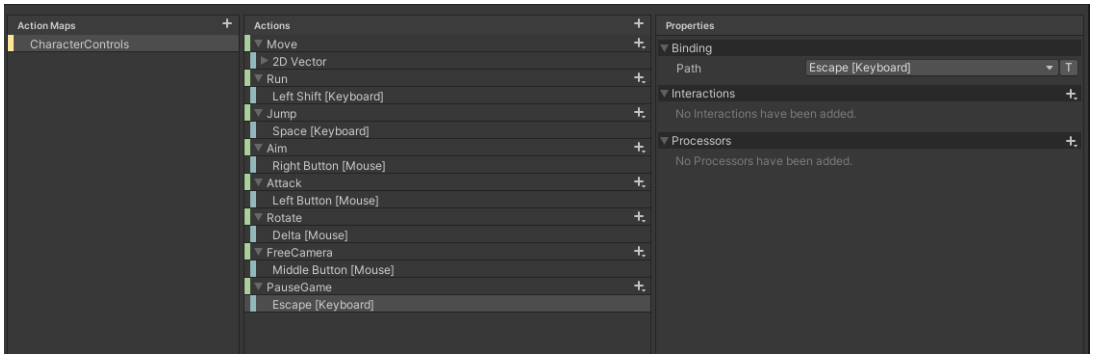

Figura 97. CharacterControls Input System

```
void Awake(){
    Cursor.lockState = CursorLockMode.Locked;

    playerInput = new PlayerInput();
    activeCamera = true;
    characterController = GetComponent<CharacterController>();

    playerInput.CharacterControls.Move.started += onMovementInput;
    playerInput.CharacterControls.Move.performed += onMovementInput;
    playerInput.CharacterControls.Move.canceled += onMovementInput;

    playerInput.CharacterControls.Jump.started += onJumpInput;
    playerInput.CharacterControls.Jump.canceled += onJumpInput;

    playerInput.CharacterControls.Run.started += onRunInput;
    playerInput.CharacterControls.Run.canceled += onRunInput;

    playerInput.CharacterControls.FreeCamera.started += onSetActiveStart;
    //playerInput.CharacterControls.FreeCamera.canceled += onSetActiveCancel;
}
```

Seguidament, a l'script, hi ha els mètodes que es cridaran cada cop que l'usuari fa l'input corresponent. El primer d'ells és onSetActiveStart, que és el mètode que controla si s'utilitza la càmera fixe o la càmera lliure. El segon i el tercer, guarden si l'usuari està clicant el botó de córrer o de saltar respectivament. El quart mètode és l'encarregat de guardar l'input dels moviments del personatge.

```
void onSetActiveCancel(InputAction.CallbackContext context){

   staticCamera.SetActive(true);
   freeCamera.transform.position = staticCamera.transform.position;
   freeCamera.transform.rotation = staticCamera.transform.rotation;
   freeCamera.SetActive(false);
}

void onSetActiveStart(InputAction.CallbackContext context){
   if(activeCamera){
      freeCamera.SetActive(true);
      freeCamera.transform.position = staticCamera.transform.position;
      freeCamera.transform.rotation = staticCamera.transform.rotation;
      staticCamera.SetActive(false);
   }
   else{
      staticCamera.SetActive(true);
      freeCamera.transform.position = staticCamera.transform.position;
      freeCamera.transform.rotation = staticCamera.transform.rotation;
      freeCamera.SetActive(false);
   }
   activeCamera = !activeCamera;
}
void onRunInput(InputAction.CallbackContext context){
   runPressed = context.ReadValueAsButton();
}

void onJumpInput(InputAction.CallbackContext context){
   jumpPressed = context.ReadValueAsButton();
}
void onMovementInput(InputAction.CallbackContext context){
      currentMovementInput = context.ReadValue<Vector2>();
      //Debug.Log(currentMovementInput);
      currentMovement.x = currentMovementInput.x * walkSpeedMultiplier;
      currentMovement.z = currentMovementInput.y * walkSpeedMultiplier;
      currentRunMovement.x = currentMovementInput.x * runMultiplier;
      currentRunMovement.z = currentMovementInput.y * runMultiplier;
      isMovementPressed = currentMovementInput.x != 0 || currentMovementInput.y != 0;
}
```

Els dos mètodes següents, s'ocupen d'habilitar i deshabilitar l'ActionMap corresponent, per no llegir inputs quan no és necessari.

```
void OnEnable(){
    playerInput.CharacterControls.Enable();
}

void OnDisable(){
    playerInput.CharacterControls.Disable();
}
```

A continuació el mètode Start. Aquest guarda el component animator en l'atribut animator i passa els paràmetres que s'utilitzaran de l'animator al seu hash per millorar el rendiment.

```
void Start()
{
    animator = GetComponent<Animator>();

    VelocityZHash = Animator.StringToHash("Velocity Z");
    VelocityXHash = Animator.StringToHash("Velocity X");
    jumpHash = Animator.StringToHash("wannaJump");
    /*aimHash = Animator.StringToHash("aiming");
    attackHash = Animator.StringToHash("isAttacking");*/
}
```

El següent mètode és l'encarregat d'incrementar els paràmetres Velocity X i Velocity Y depenent dels inputs de l'usuari i enviar el nou valor a l'animador. El seu funcionament és tal que, mira els inputs de l'usuari en aquell moment, i si no s'ha arribat a la velocitat màxima del moment (la velocitat màxima pot variar si el botó de córrer està clicat o no), incrementa el paràmetre necessari depenent de l'input sumant el valor actual l'atribut acceleration multiplicat per Time.deltaTime i en el cas necessari, el multiplicador de córrer.

```
void changeVelocity(bool forwardPressed, bool backPressed, bool leftPressed, bool
rightPressed, bool runPressed, float currentMaxVelocity){
    if(forwardPressed && velocityZ < currentMaxVelocity){
        if(runPressed){
            velocityZ += Time.deltaTime * acceleration * runAnimationMultiplier;
        }
        else{
            velocityZ += Time.deltaTime * acceleration;
        }

    }
```

```
    if(leftPressed && velocityX > -currentMaxVelocity){
      if(runPressed){
        velocityX -= Time.deltaTime * acceleration * runAnimationMultiplier;
      }
      else{
        velocityX -= Time.deltaTime * acceleration;
      }
    }
    if(rightPressed && velocityX < currentMaxVelocity){
      if(runPressed){
        velocityX += Time.deltaTime * acceleration * runAnimationMultiplier;
      }
      else{
        velocityX += Time.deltaTime * acceleration;
      }
    }
    if(backPressed && velocityZ > -(currentMaxVelocity/2)){
      if(runPressed){
        velocityZ -= Time.deltaTime * acceleration * runAnimationMultiplier;
      }
      else{
        velocityZ -= Time.deltaTime * acceleration;
      }
    }
  }
```

El següent mètode, s'ocupa del contrari que l'anterior, disminueix el valor dels paràmetres Velocity X i Velocity Y si troba a faltar algun input de moviment. El funcionament és semblant a l'anterior també, es mira els botons d'input que no estan clicats i es mira si el valor del paràmetre és el mínim, si no ho és, disminueix el valor restant l'atribut deceleration multiplicat per Time.DeltaTime i en el cas necessari, el multiplicador de córrer.

```
  void lockOrResetVelocity(bool forwardPressed, bool backPressed, bool leftPressed, bool
rightPressed, bool runPressed, float currentMaxVelocity){
    if(!forwardPressed && velocityZ > 0.0f){
      if(runPressed){
        velocityZ -= Time.deltaTime * deceleration * runAnimationMultiplier;
      }
      else{
        velocityZ -= Time.deltaTime * deceleration;
      }
    }

    if(!backPressed && velocityZ < 0.0f){
      velocityZ += Time.deltaTime * deceleration;
    }
```

```
    if(!forwardPressed && !backPressed && velocityZ != 0.0f && (velocityZ > -0.05f &&
velocityZ < 0.05f)){
        velocityZ = 0.0f;
    }

    if(!leftPressed && velocityX < 0.0f){
        if(runPressed){
            velocityX += Time.deltaTime * deceleration * runAnimationMultiplier;
        }
        else{
            velocityX += Time.deltaTime * deceleration;
        }

    }
    if(!rightPressed && velocityX > 0.0f){
        if(runPressed){
            velocityX -= Time.deltaTime * deceleration * runAnimationMultiplier;
        }
        else{
            velocityX -= Time.deltaTime * deceleration;
        }
    }

    if(!leftPressed && !rightPressed && velocityX != 0.0f && (velocityX > -0.05f && velocityX <
0.05f)){
        velocityX = 0.0f;
    }

    if(forwardPressed && runPressed && velocityZ > currentMaxVelocity){
        velocityZ = currentMaxVelocity;
    }
    else if(forwardPressed && velocityZ > currentMaxVelocity){
        velocityZ -= Time.deltaTime * deceleration;
        if(velocityZ > currentMaxVelocity && velocityZ < (currentMaxVelocity + 0.05f)){
            velocityZ = currentMaxVelocity;
        }
    }
    else if(forwardPressed && velocityZ < currentMaxVelocity && velocityZ
>(currentMaxVelocity - 0.05f)){
        velocityZ = currentMaxVelocity;
    }
    if(backPressed && runPressed && velocityZ < -(currentMaxVelocity/2)){
        velocityZ = -(currentMaxVelocity/2);
    }
```

```
    else if(backPressed && velocityZ < -(currentMaxVelocity/2)){
        velocityZ += Time.deltaTime * deceleration;
        if(velocityZ < currentMaxVelocity && velocityZ > (-(currentMaxVelocity/2) - 0.05f)){
            velocityZ = -(currentMaxVelocity/2);
        }
    }
    else if(backPressed && velocityZ > -(currentMaxVelocity/2) && velocityZ < (-
(currentMaxVelocity/2) + 0.05f)){
        velocityZ = -(currentMaxVelocity/2);
    }

    if(leftPressed && runPressed && velocityX < -currentMaxVelocity){
        velocityX = -currentMaxVelocity;
    }
    else if(leftPressed && velocityX < -currentMaxVelocity){
        velocityX += Time.deltaTime * deceleration;
        if(velocityX < -currentMaxVelocity && velocityX > (-currentMaxVelocity - 0.05f)){
            velocityX = -currentMaxVelocity;
        }
    }
    else if(leftPressed && velocityX > -currentMaxVelocity && velocityX < (-
currentMaxVelocity + 0.05f)){
        velocityX = -currentMaxVelocity;
    }

    if(rightPressed && runPressed && velocityX > currentMaxVelocity){
        velocityX = currentMaxVelocity;
    }
    else if(rightPressed && velocityX > currentMaxVelocity){
        velocityX -= Time.deltaTime * deceleration;
        if(velocityX > currentMaxVelocity && velocityX < (currentMaxVelocity + 0.05f)){
            velocityX = currentMaxVelocity;
        }
    }
    else if(rightPressed && velocityX < currentMaxVelocity && velocityX >(currentMaxVelocity
- 0.05f)){
        velocityX = currentMaxVelocity;
    }
```

El pròxim mètode, checkJumping, és l'encarregat d'executar o deixar d'executar l'animació de saltar. Això ho aconsegueix mirant el valor del paràmetre wannaJump. Si no està saltant i vol saltar, assigna true al paràmetre de l'animator, que farà que el personatge salti. Pel contrari, si el personatge està saltant i no hi ha l'input del botó de saltar, s'assignarà false al paràmetre.

```
void checkJumping(){
    bool isJumping = animator.GetBool(jumpHash);
    if(jumpPressed && !isJumping){
        animator.SetBool(jumpHash, true);
    }
    if(!jumpPressed && isJumping){
        animator.SetBool(jumpHash, false);
    }
}
```

El següent mètode, handleRotation, s'ocupa de la rotació del personatge quan l'usuari fa un input dels botons per moure's cap a un costat. Aquest mètode calcula el nou Vector3 de moviment que s'enviarà al characterController.

```
void handleRotation(){
    if(currentMovementInput.magnitude >= 0.1f){
        float targetAngle = Mathf.Atan2(currentMovementInput.x, currentMovementInput.y) *
                                        Mathf.Rad2Deg + staticCamera.transform.eulerAngles.y;
        float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y, targetAngle,
                                            ref turnSmoothVelocity, turnSmoothTime);
        transform.rotation = Quaternion.Euler(0f, angle, 0f);
        currentMovement = Quaternion.Euler(0f, targetAngle, 0f) * Vector3.forward *
                                                                  walkSpeedMultiplier;
        currentRunMovement = currentMovement * runMultiplier;

    }
}
```

Seguidament ens trobem el mètode handleJumpsAndGravity que, a diferència del mètode checkJumping, aquest s'ocupa del moviment del personatge en comptes de l'animació. El que fa aquest mètode és mirar si el personatge està tocant el terra a partir del mètode la posició del characterController i actualitzarà el valor del component Y del Vector3 de moviment, que és la responsable del moviment vertical, restant-hi el valor de la gravetat. En el cas que el personatge estigui tocant el terra i l'usuari vulgui que el personatge salti, es modificarà la component Y posant-hi l'atribut jumpSpeed. Per últim s'actualitzarà el component Y dels Vector3 de moviment.

```
void handleJumpsAndGravity(){
    isGrounded = characterController.transform.position.y < 0.30f;
    //Debug.Log(isGrounded);
    float ySpeed = currentMovement.y;
    ySpeed += gravityValue * Time.deltaTime;
```

```
    if(isGrounded){
      ySpeed = -0.5f;
      if(jumpPressed){
        ySpeed = jumpSpeed;
      }
    }
    //Debug.Log(ySpeed);
    currentMovement.y = ySpeed;
    currentRunMovement.y = ySpeed;
  }
```

Els següents dos mètodes, EnableMovement i DisableMovement, són mètodes públics que s'encarreguen d'habilitar o desactivar el component characterController.

```
public void EnableMovement(){
  characterController.enabled = true;
}

public void DisableMovement(){
  characterController.enabled = false;
}
```

El pròxim mètode, UnlockCursor, desbloqueja el cursor per tal que l'usuari el pugui moure i clicar els botons dels menús.

```
  public void UnlockCursor(){
    Cursor.lockState = CursorLockMode.None;
  }
```

Per últim, el mètode Update que és l'encarregat de cridar, a cada frame del joc, els mètodes que actualitzen els paràmetres de moviment i donar moviment al personatge.

```
  void Update()
  {
    bool forwardPressed = currentMovementInput.y > 0.0f;
    bool backPressed = currentMovementInput.y < 0.0f;
    bool leftPressed = currentMovementInput.x < 0.0f;
    bool rightPressed = currentMovementInput.x > 0.0f;

    float currentMaxVelocity = runPressed ? maximumRunVelocity :  maximumWalkVelocity;
    changeVelocity(forwardPressed, backPressed, leftPressed, rightPressed, runPressed,
                   currentMaxVelocity);
    lockOrResetVelocity(forwardPressed, backPressed, leftPressed, rightPressed, runPressed,
                        currentMaxVelocity);
    checkJumping();
    animator.SetFloat(VelocityZHash, velocityZ);
```

```
    animator.SetFloat(VelocityXHash, velocityX);
    //animator.SetFloat(aimHash, aiming);

    handleRotation();
    handleJumpsAndGravity();

    //Debug.Log(currentRunMovement.y);
    if(runPressed){
        characterController.Move(currentRunMovement * Time.deltaTime);
    }
    else{
        characterController.Move(currentMovement * Time.deltaTime);
    }
}
```

L'últim component del prefab del personatge, és l'script responsable de l'estat, vida, valors a l'hora d'atacar, a qui atacar, etc. Els atribus són els de la Secció 8.6.5.1.

```
Animator animator;
float aiming = 0.0f;
bool aimPressed;
public float aimAcceleration = 0.05f;
bool attackPressed;
int aimHash;
int attackHash;
int isDeadHash;
public Transform attackPoint;
public float attackRange = 0.5f;
public LayerMask enemyLayers;
public float attackInterval = 1.0f;
public int meleeDamage = 30;
float nextAttack = 0f;
public int maxHealth = 100;
int currentHealth;
PlayerInput playerInput;
private bool isDead = false;
```

Pel que respecta als mètodes, el primer de tots, el mètode Awake, és molt semblant al mètode Awake de l'script de moviments. Crea una instància de la classe PlayerInput i declara els mètodes que es cridaran depenent de l'input de l'usuari.

```
void Awake(){
    playerInput = new PlayerInput();
    playerInput.CharacterControls.Aim.started += onAimInput;
    playerInput.CharacterControls.Aim.canceled += onAimInput;

    playerInput.CharacterControls.Attack.started += onAttackInput;
    playerInput.CharacterControls.Attack.canceled += onAttackInput;

    playerInput.CharacterControls.PauseGame.started += onPauseInput;
    //playerInput.CharacterControls.PauseGame.canceled += onPauseInput;
}
```

Seguidament es troben aquest mètodes, onPauseInput, onAttackInput i onAimInput. El primer d'ells és l'encarregat de posar pausa al joc, modificant l'escala de temps a 0, és a dir, que no passa el temps. En el cas que el joc ja estigués en pausa, es posaria a 1. El segon i tercer mètode, guarden el valor de l'input, true si hi ha l'input, false altrament.

```
void onPauseInput(InputAction.CallbackContext context){
    if(Time.timeScale == 0){
        Time.timeScale = 1;
        Debug.Log("Game unpaused");
    }
    else{
        Time.timeScale = 0;
        Debug.Log("Game paused");
    }
}
void onAttackInput(InputAction.CallbackContext context){
    attackPressed = context.ReadValueAsButton();
}

void onAimInput(InputAction.CallbackContext context){
    aimPressed = context.ReadValueAsButton();
}
```

El següent mètode, changeAiming, reb com a paràmetre attackLayerWeight. Aquest és el paràmetre que permet saber el pes que té la meleeAttack layer. Per tant, en el cas que l'usuari vulgui apuntar i l'animació d'apuntar no s'hagi completat, si el pes de la meleeAttack layer és inferior a 1, se li donarà el pes màxim de 1 ja que ens interessa que només es visualitzi l'animació d'apuntar. Seguidament s'augmentarà el valor del paràmetre aiming.

```
void changeAiming(float attackLayerWeight){
    if(aimPressed && aiming < 1.0f){
        if(attackLayerWeight <
1.0f){animator.SetLayerWeight(animator.GetLayerIndex("meleeAttack"), 1.0f);}
        aiming += aimAcceleration * Time.deltaTime;
        //Debug.Log(aimPressed);
    }
}
```

Igual que amb els moviments, en el mètode anterior s'augmenta el valor del paràmetre aiming. Per tant en el següent mètode, lockOrResetAiming, es farà el contrari: en el cas de que l'usuari no tingui el botó d'apuntar clicat i l'animació encara no s'hagi acabat, es disminuirà el paràmetre aiming. Per acabar el mètode, si aiming té valor de 0, es tornarà a assignar un pes de 0 a la meleeAttack layer ja que, en aquest moment, no interessa que es visualitzin les animacions d'aquesta layer.

```
void lockOrResetAiming(float attackLayerWeight){
    if(!aimPressed && aiming > 0.0f){
        aiming -= aimAcceleration * Time.deltaTime;
        if(aiming<0.0f){aiming = 0.0f;}
    }
    if(aiming == 0.0f) {animator.SetLayerWeight(animator.GetLayerIndex("meleeAttack"),
                                0.0f);}
}
```

Els dos següents mètodes, checkAttack i lockOrResetAttacking, l'utilitat és molt semblant a la dels dos mètodes anteriors, però ocupant-se del paràmetre isAttacking. En el primer dels mètodes, si el personatge està apuntant, es clica el botó d'atacar i, sinó s'està atacant, s'assignarà true al paràmetre. En el segon mètode, mira que el botó d'atacar no estigui clicat i s'estigui atacant per assignar false al paràmetre. Aquest dos mètodes només s'ocupen de l'animació, més endavant es veurà el mètode Attack, que s'ocupa de l'acció.

```
void checkAttack(bool attacking){
    if(aimPressed && attackPressed && !attacking){
        animator.SetBool(attackHash, true);
    }
}

void lockOrResetAttacking(bool attacking){
    if(!attackPressed && attacking){
        animator.SetBool(attackHash, false);
    }
}
```

Seguidament ens trobem el mètode Start, que assigna la instància del component Animator a l'atribut animator, passa els paràmetres al seu hash per millorar el rendiment i assigna la vida al personatge.

```
void Start()
{
   animator = GetComponent<Animator>();
   aimHash = Animator.StringToHash("aiming");
   attackHash = Animator.StringToHash("isAttacking");
   isDeadHash = Animator.StringToHash("isDead");
   currentHealth = maxHealth;
}
```

Llavors, el mètode Update, com s'ha explicat, s'executa una vegada per frame. S'ocupa de, si el personatge no està mort, agafa el valor isAttacking de l'animator i el pes de la meleeAttack layer. Seguidament crida els mètodes que s'ocupen de les animacions d'apuntar i atacar i, per acabar, si l'usuari té clicats els botons d'apuntar i atacar a la vegada, es cridarà el mètode Attack.

```
void Update()
{
   if(!isDead){
      bool attacking = animator.GetBool(attackHash);
      float attackLayerWeight =
            animator.GetLayerWeight(animator.GetLayerIndex("meleeAttack"));

      changeAiming(attackLayerWeight);
      lockOrResetAiming(attackLayerWeight);
      checkAttack(attacking);
      lockOrResetAttacking(attacking);

      animator.SetFloat(aimHash, aiming);
      if(aimPressed && attackPressed){
         Attack();
      }
   }
}
```

El mètode Attack, és el que s'ocupa de l'acció d'atacar i infringir mal als enemics. Primer de tot mira que hagi passat un cert temps des de l'últim atac. Si ha passat aquest temps, i per tant es pot atacar, actualitza aquest temps perquè no es fagin atacs massa seguits i, utilitzant el mètode OverlapSphere passant-li la posició central, el radi i la layer, es guardaran els Collider dels enemics que hi hagi. Llavors, per a cada enemic, es buscarà el controlador i es cridarà el mètode TakeDamage.

```
void Attack(){
    if(Time.time > nextAttack){
        nextAttack = Time.time + attackInterval;
        Collider[] hitEnemies = Physics.OverlapSphere(attackPoint.position, attackRange,
enemyLayers);
        foreach(Collider enemic in hitEnemies){
            enemic.GetComponent<EnemyController>().TakeDamage(meleeDamage);
            //Debug.Log(enemic.name);
        }
    }
}
```

Seguint amb els mètodes, igual que amb els moviments, es necessiten els mètodes OnEnable i OnDisable per la classe PlayerInput, per no llegir inputs quan no és necessari.

```
void OnEnable(){
    playerInput.CharacterControls.Enable();
}
void OnDisable(){
    playerInput.CharacterControls.Disable();
}
```

El següent mètode, TakeDamage, és public i cridat des del controlador dels enemics. La funció és, si el personatge no s'ha quedat a 0 punts de vida, restarà el paràmetre damage que se li passa a l'atribut currentHealth i, si la vida es queda a 0 o a menys de 0, es cridarà el mètode Die.

```
public void TakeDamage(int damage){
    if(!isDead){
        Debug.Log("Player takes dmg");
        currentHealth -= damage;
        if(currentHealth <= 0){
            Die();
        }
    }
}
```

El mètode Die conté la lògica per quan el personatge mor. Primer de tot s'actualitzarà l'atribut isDead per a que l'update no cridi els mètodes que conté i s'activarà el trigger isDead de l'animator que executarà l'animació de quan l'usuari mor. Seguidament es desactivarà el component de moviments del personatge i per últim, es cridarà el mètode playerDead del controlador de la partida.

```
void Die(){
    Debug.Log("Player died");
    isDead = true;
    animator.SetTrigger(isDeadHash);
    GetComponent<MovementAndAnimationPlayerController>().enabled = false;
    GameObject gameController = GameObject.FindWithTag("GameController");
    gameController.GetComponent<GameControllerScript>().playerDead();
    //this.enabled = false;
}
```

Per últim, el mètode CallFadeOut, s'utilitzar per cridar el mètode CallFadeOut del controlador de la partida.

```
public void CallFadeOut(){
    GameObject gameController = GameObject.FindWithTag("GameController");
    gameController.GetComponent<GameControllerScript>().CallFadeOut();
}
```

Tot i haver cobert tots els components que té el personatge, aquest també disposa varis fills. Aquests fills són, la càmera estàtica (StaticCameraPlayer), la càmera lliure (FreePlayerCamera) i el punt on ataca el personatge (AttackPoint).

La càmera estàtica, és la càmera que es veurà per defecte quan es comenci la partida. Aquesta només seguirà el personatge i no es podrà moure ni modificar el zoom.

A diferència, la càmera lliure pot ser controlada per l'usuari a través del moviment del ratolí i el zoom a través de la rodeta del ratolí. El canvi de càmera estàtica a càmera lliure està a l'script PlayerAttack.

Per aconseguir el moviment, s'ha afegit un nou sistema de input anomenat CameraInput i component script, CameraControllerScript, que s'ocupa de gestionar aquests inputs. L'script està format per diferents atributs explicats a la Secció 8.6.4.1. En quant a mètodes, el primer és el mètode Awake, que, com als altres scripts on hi ha un Input, es crea l'instància de la classe i es defineixen les crides que es faran quan es detectin inputs. Aquestes crides, criden els mètodes onZoomInput i onMovementInput.

```
public float sensitivity = 5.0f;
public float smoothFactor = 0.5f;
CameraInput cameraInput;
Vector2 currentMovementInput;
Vector2 currentZoomInput;
bool isMoving;
private float rotationX;
private float rotationY;
private Transform target;
public float maxFOV = 70f;
public float minFOV = 30f;
```

```
void Awake(){
    cameraInput = new CameraInput();
    target = this.transform.parent.gameObject.transform;

    cameraInput.CameraControls.Move.started += onMovementInput;
    cameraInput.CameraControls.Move.performed += onMovementInput;
    cameraInput.CameraControls.Move.canceled += onMovementInput;

    cameraInput.CameraControls.Zoom.started += onZoomInput;
    cameraInput.CameraControls.Zoom.performed += onZoomInput;
    cameraInput.CameraControls.Zoom.canceled += onZoomInput;
}
```

OnZoomInput llegeix el valor de l'input. Si el valor és negatiu, significa que el jugador vol ampliar el zoom, però, abans de fer-ho, es comprovarà que no s'hagi arribat al màxim de zoom. Pel contrari, si el valor de l'input és positiu, vol dir que l'usuari vol disminuir el zoom i, igual que amb l'augment, primer es mirarà que no s'hagi arribat al mínim de zoom. Per últim, per evitar problemes a l'hora de fer el zoom i que un cop fet no es modifiqui la posició de la càmera, s'assigna la posició on es trobava la càmera abans del zoom.

```
void onZoomInput(InputAction.CallbackContext context){
    currentZoomInput = context.ReadValue<Vector2>();
    Vector3 currentPosition = transform.localPosition;
    //Debug.Log(currentZoomInput);
    if(currentZoomInput.y < 0f){
        if(GetComponent<Camera>().fieldOfView < maxFOV){
            GetComponent<Camera>().fieldOfView++;
        }
    }else if(currentZoomInput.y > 0f){
        if(GetComponent<Camera>().fieldOfView > minFOV){
            GetComponent<Camera>().fieldOfView--;
        }
    }
    transform.localPosition = currentPosition;

}
```

OnMovementInput llegeix l'input del moviment del ratolí. Aquest moviment és transformat multiplicant-lo per la sensibilitat del ratolí i Time.deltaTime i guardat als atributs rotationX i rotationY. Amb el moviment guardat, es crida el mètode RotateAround que modifica el component transform de la càmera i el rota.

```
void onMovementInput(InputAction.CallbackContext context){
    currentMovementInput = context.ReadValue<Vector2>();
    //Debug.Log(currentMovementInput);

    isMoving = currentMovementInput.x != 0 || currentMovementInput.y != 0;

    rotationX = currentMovementInput.x * sensitivity * Time.deltaTime;
    rotationY = currentMovementInput.y * sensitivity * Time.deltaTime;

    transform.RotateAround(target.transform.position, Vector3.up, rotationX);
}
```

Igual que amb els altres scripts on hi ha una instància d'una classe Input, són necessaris els mètodes OnEnable i on Disable que habiliten o desactiven l'input. En aquest cas és el més clar el perquè són necessaris aquests mètodes, ja que la càmera lliure no estarà sempre activa i per tant, mentre no ho està, els inputs del ratolí no interessen al programa el que faria que s'estiguessin llegint inputs innecessàriament.

```
void OnEnable(){
    cameraInput.CameraControls.Enable();
}

void OnDisable(){
    cameraInput.CameraControls.Disable();
}
```

L'últim mètode d'aquest script és l'Update, aquest mètode és utilitzat per assegurar que la càmera sempre apunta al personatge del joc. Això s'aconsegueix buscant la posició del personatge, sumant l'alçada que es vulgui i cridant el mètode LookAt de transform.

```
void Update()
{
    Vector3 targetPosition = target.position;
    targetPosition.y += 1.7f;
    transform.LookAt(targetPosition);

}
```

9.9 ENEMICS

Els enemics són l'altre component necessari per fer del videojoc un videojoc de supervivència. Com s'ha vist en apartats anteriors, hi ha 4 tipus d'enemic. Tot hi haver-n'hi 4 de diferents, els components i comportaments són els mateixos, veure Figura 98.

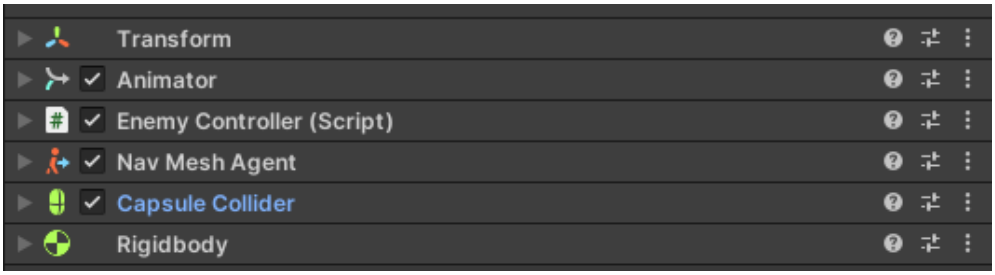

Figura 98. Components enemics

El primer d'aquest components, després del Transform que és comú a tots els GameObjects, és l'Animator. Aquest component, igual que amb el personatge, és el gestor de les animacions dels enemics. Està format per layers i paràmetres. En aquest cas, de layers només hi ha la Base Layer i de paràmetres hi ha, tres bool, (StandUpKind, isCloseEnough i walkKind) a més de dos triggers, enemyDead, playerDead. Aquest animator, pot ser més senzill que el que s'ha fet pel personatge, ja que només està format per animacions i transicions, sense Blend Trees.

Les animacions que s'ha posat en aquest animator són: la primera animació que farà un enemic serà l'animació ZombieStandUp(2), però a partir de codi modificant el paràmetre StandUpKind, es pot saltar directament i que s'executi l'animació ZombieStandUp. Aquesta part està feta d'aquesta manera ja que, com que els enemics seran instanciats a partir d'un prefab, no seria eficient tenir dos Animators per les dos maneres d'aixecar-se que tenen els enemics i llavors, un cop instanciats, assignar un animator aleatòriament. Pel que, fet d'aquesta manera, encara que s'acabi executant l'animació ZombieStandUp, les dos animacions de ZombieStandUp no es solaparan.

Un cop s'ha acabat d'executar una d'aquestes dos animacions, s'executarà l'animació ZombieScream que, un cop acabada, depenent de si el paràmetre walkKind és true o false, s'executarà l'animació RunningCrawl o ZombieRunning. En qualsevol cas, aquestes animacions es faran en bucle (volent dir que l'enemic esta perseguint el personatge) fins que el paràmetre isCloseEnough sigui true, quan es transicionarà a l'animació ZombieNeckBite. Un cop acabada l'animació, si el paràmetre continua siguent true, es tornarà a executar, i si és false, es tornarà a l'animació RunningCrawl o ZombieRunning.

Aquest seria el flux principal d'animacions, però, independent de l'estat en que es trobin, si l'enemic es queda sense punts de vida, es morirà. Aquesta transició és la que comença a l'estat AnyState, es dirigeix a ZombieDying i és cridada a través del trigger enemyDead.

També pot passar que el personatge es mori, el que fa que s'acabi el joc i llavors que ja no sigui necessari que els enemics continuïn perseguint el personatge o atacant-lo. Per tant quan el personatge mor, s'activa el trigger playerDead que farà que s'executi l'animació de Idle en els enemics, estiguin perseguint-lo o atacant-lo. Si l'estan atacant, primer s'acabarà l'animació. Com es modifiquen els paràmetres es veurà quan s'expliqui el funcionament de l'script. Veure Figura 99.

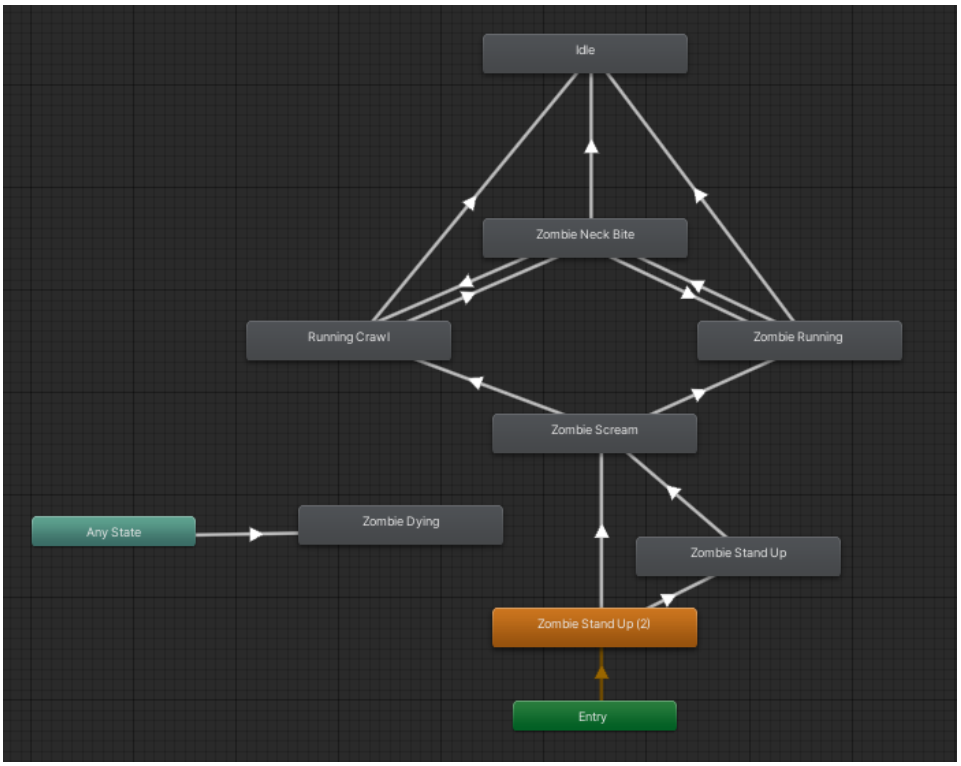

Figura 99. Animator enemics

Els components CapsuleCollider i RigidBody són els components necessaris per a que els enemics puguin col·lisionar amb altres objectes del mapa i com que no disposen de character controller, també siguin afectats per la gravetat.

Llavors també hi ha el component NavMeshAgent que, com s'ha explicat a la Secció 7.2.2.21, transforma el GameObject en un agent capaç de moure's per la NavMesh.

El component EnemyController és l'script que controlarà el comportament de l'enemic i modificarà els paràmetres de l'animator. 8.6.7.1. Pel que respecta als mètode d'aquest script, tot i no tenir una classe input, es necessita el mètode Awake per triar el tipus de StandUp i el tipus de caminar a més d'aprofitar i guardar el component NavMeshAgent.

```
Animator animator;
int standUpKindHash;
int walkKindHash;
int isCloseEnoughHash;
int enemyDeadHash;
int playerDeadHash;
int randomStandUp;
int randomWalk;
private NavMeshAgent navMeshAgent;
GameObject movePositionTransform;
public int maxHealth = 100;
int currentHealth;
```

```
public Transform attackPoint;
public float attackDistance = 0.5f;
public LayerMask playerLayers;
public float attackInterval = 1.0f;
public int attackDamage = 30;
float nextAttack;
private bool isPlayerDead = false;

void Awake(){
   randomStandUp = new System.Random().Next(0, 2);
   //randomWalk = new System.Random().Next(0, 2);
   randomWalk = 1;
   navMeshAgent = GetComponent<NavMeshAgent>();
}
```

Es continua l'script amb el mètode Start, el que guarda el component de l'animator i transforma els paràmetres en el hash per millor rendiment. També s'assigna l'objectiu que perseguiran els enemics, en aquest cas, el GameObject que tingui de tag "Player".

```
void Start()
{
   animator = GetComponent<Animator>();
   standUpKindHash = Animator.StringToHash("StandUpKind");
   //bool standUpAux = Convert.ToBoolean(randomStandUp);
   animator.SetBool(standUpKindHash, Convert.ToBoolean(randomStandUp));
   isCloseEnoughHash = Animator.StringToHash("isCloseEnough");
   enemyDeadHash = Animator.StringToHash("enemyDead");
   walkKindHash = Animator.StringToHash("walkKind");
   playerDeadHash = Animator.StringToHash("playerDead");
   animator.SetBool(walkKindHash, Convert.ToBoolean(randomWalk));
   navMeshAgent.isStopped = true;
   currentHealth = maxHealth;

   movePositionTransform = GameObject.FindWithTag("Player");

}
```

El següent mètode és un mètode per assegurar-se que l'enemic apareix en un lloc correcte. Com que el personatge es pot trobar en mig d'una ciutat, pot ser que els enemics apareguin dins d'un edifici o dins de l'aigua. Com que no és el que es vol, amb OnTriggerStay, si l'enemic apareix en un d'aquests elements, es desactivaran els components i es destruirà el GameObject de l'enemic.

```
  void OnTriggerStay(Collider other){
    if(other.CompareTag("ExtrudedStructure") || other.CompareTag("ModeledStructure") ||
other.CompareTag("AreaWater")){
        animator.enabled = false;
        navMeshAgent.enabled = false;
        Destroy(gameObject);
        Debug.Log("enemy destroyed");
    }
  }
```

A continuació hi ha un mètode públic, enableFollowing, que quan és cridat, mira si el jugador està viu i si ho està, habilita el seguiment del navMeshAgent.

```
  public void enableFollowing(){
    if(!isPlayerDead) navMeshAgent.isStopped = false;
  }
```

Un altre mètode públic és el TakeDamage. En aquest mètode se li passa un paràmetre, damage, i és cridat quan el personatge fa un atac i l'enemic està dins de la OverlapSphere. El que fa aquest mètode és restar el paràmetre damage a la vida que té l'enemic i llavors, si la vida que li queda és igual o inferior a 0, crida el mètode Die.

```
  public void TakeDamage(int damage){
    currentHealth -= damage;

    if(currentHealth <= 0){
      Die();
    }
  }
```

El mètode Die és l'encarregat de dur a terme la lògica quan l'enemic s'ha quedat sense punts de vida. Es diu al navMeshAgent que no es continui seguint al personatge, activem el trigger enemyDead que executarà l'animació de mort, es desactiva el RigidBody per tal que aquest enemic ja no causi col·lisions i per últim, es desactiva el component de l'script.

```
  void Die(){
    navMeshAgent.isStopped = true;
    animator.SetTrigger(enemyDeadHash);
    GetComponent<Rigidbody>().detectCollisions = false;
    this.enabled = false;
  }
```

A continuació, un altre mètode públic playerDead conté la lògica que s'ha de dur a terme quan el personatge s'ha mort. S'activarà el trigger playerDead, es desactivaran les col·lisions del RigidBody, es posarà l'atribut isPlayerDead a true i es desactivarà el component navMeshAgent.

```
public void playerDead(){
    animator.SetTrigger(playerDeadHash);
    GetComponent<Rigidbody>().detectCollisions = false;
    isPlayerDead = true;
    navMeshAgent.isStopped = true;
    navMeshAgent.enabled = false;
}
```

El mètode Attack és el responsable tant de l'animació com de l'acció d'atacar de l'enemic. Primer de tot es demana a l'animator el valor del paràmetre isCloseEnough i es mira la distància entre l'enemic i el personatge. Llavors si ha passat el suficient temps des de l'últim atac, i fins aquest moment no estava prou aprop del jugador i la distància entre l'enemic i el personatge és inferor a attackDistance, primer s'actualitzarà l'atribut nextTime per tal que l'enemic no ataqui massa ràpid, i llavors farà l'atac.

Per fer l'atac, a partir del mètode OverlapSphere passant-li els paràmetres posició de l'attackPoint, l'attackDistance i playerLayers, es tindrà una llista dels Collider que estan dins d'aquesta zona. Llavors, per cada collider, es buscarà el component PlayerAttack i es cridarà el mètode TakeDamage, passant-li l'atribut attackDamage. En aquest cas no és necessari fer un foreach, ja que sabem que només hi ha un jugador, però futures implementacions amb múltiples jugadors, poden necessitar aquest codi. Interessa que l'enemic no es mogui quan ataca, per tant, primer de tot es canvia el valor del paràmetre isCloseEnough de l'animator a true perquè executi l'animació d'atacar i si el personatge no es mor, es desactiva el navMeshAgent perquè d'aquesta manera no ataqui i es mogui a la vegada.

Per acabar el mètode, es mira si l'enemic continua estant aprop del personatge, en el cas que la variable isCloseEnough sigui true i la distància entre l'enemic i el personatge sigui superor a attackDistance, es modificarà el paràmetre isCloseEnough de l'animator i es tornarà a activar el seguiment del navMeshAgent.

```
void Attack(){
   bool isCloseEnough = animator.GetBool(isCloseEnoughHash);
   float distanceToPlayer = Vector3.Distance(this.transform.position,
                                                movePositionTransform.transform.position);
   //Debug.Log(distanceToPlayer);
   if(Time.deltaTime > nextAttack && !isCloseEnough &&  distanceToPlayer <
                                                            attackDistance){
      nextAttack = Time.deltaTime + attackInterval;
      Collider[] hitPlayer = Physics.OverlapSphere(attackPoint.position, attackDistance,
                                                      playerLayers);
      foreach(Collider player in hitPlayer){
         player.GetComponent<PlayerAttack>().TakeDamage(attackDamage);
      }
      animator.SetBool(isCloseEnoughHash, true);
      if(!isPlayerDead) navMeshAgent.isStopped = true;
   }

   if(isCloseEnough && distanceToPlayer > attackDistance){
      animator.SetBool(isCloseEnoughHash, false);
      navMeshAgent.isStopped = false;
   }
}
```

L'últim mètode és el mètode Update que, si el personatge no està mort, actualitza la posició on han d'anar els enemics i crida el mètode Attack per si l'enemic està suficientment a prop del personatge.

```
void Update()
{
   if(!isPlayerDead){
      navMeshAgent.destination = movePositionTransform.transform.position;
      Attack();
   }
}
```

9.10 NAVMESH

Com s'ha explicat a la Secció 7.2.2.22, la NavMesh és l'element que permet als GameObject que contenen un NavMeshAgent anar a una posició automàticament, on en el cas d'aquest videojoc, perseguir al personatge.

Per crear la NavMesh no és necessari cap script. Obrint la pestanya de navigation de Unity, es pot veure que hi ha 4 diferents apartats. El primer d'aquests apartats fa referència a quines característiques pot tenir l'agent a l'hora de perseguir, és a dir, quina altura poden saltar, de quina alçada són, per si han de passar per túnels o llocs amb sostre, quina amplada tenen i fins quin angle d'elevació poden pujar. Veure Figura 100.

Figura 100. Apartat Agent

El segon dels apartats és el el de Areas. Aquestes són les zones del mapa on es podrà crear la NavMesh. Per defecte hi ha les Walkable, Not Walkable i Jump. Pel cas d'aquest videojoc no fa falta afegir més zones. Veure Figura 101.

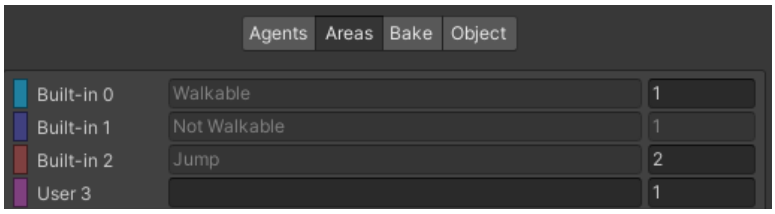

Figura 101. Apartat Areas

El tercer dels apartats, Bake, és on es crearà la NavMesh. La NavMesh es crearà, també a partir de les característiques de l'agent. A diferència de l'apartat Agent, que formarà part del component NavMeshAgent, en aquest es volen les característiques per poder crear la zona per on els agents podràn passar, és a dir, si l'agent és gaire àmple, potser no pot passar per algun passadís o carretera. Les opcions avançades no s'han modificat. Veure Figura 102.

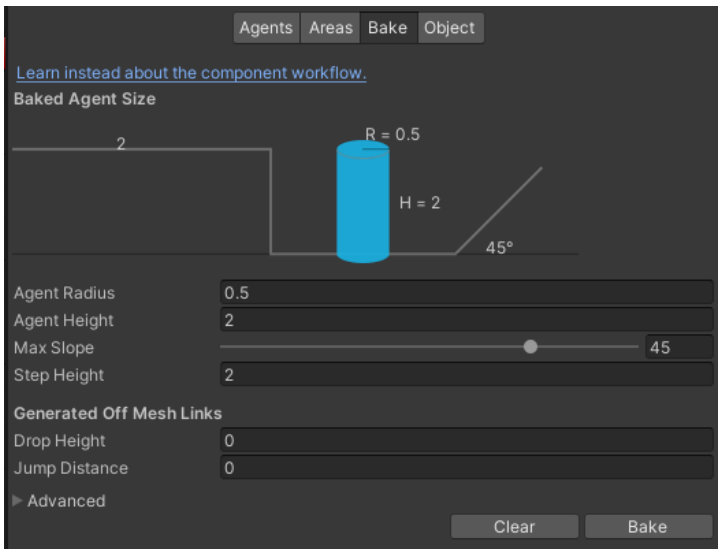

Figura 102. Apartat Bake

L'últim apartat, són els GameObjects on es mirarà si es pot crear NavMesh. Aquests GameObject són els que tenen el flag de NavigationStatic. Aquest flag, és assignat als GameObjects a l'hora de crear el mapa. Veure Figura 103.

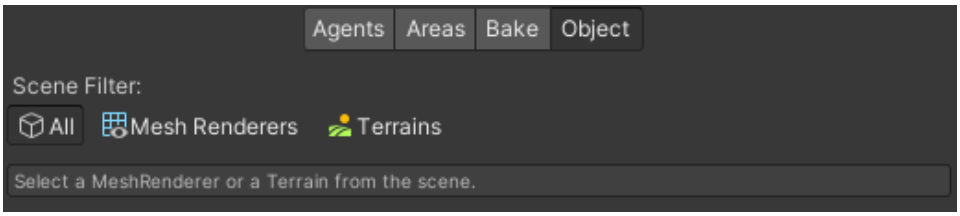

Figura 103. Apartat Object

Per crear la NavMesh, un cop s'han posat els flag i les opcions necessàries, es clica el botó de Bake que hi ha a l'apartat Bake. Llavors Unity començarà a crear la NavMesh. Podem veure la Figura 104 pel resultat de la NavMesh d'aquest videojoc amb el mapa en ocult.

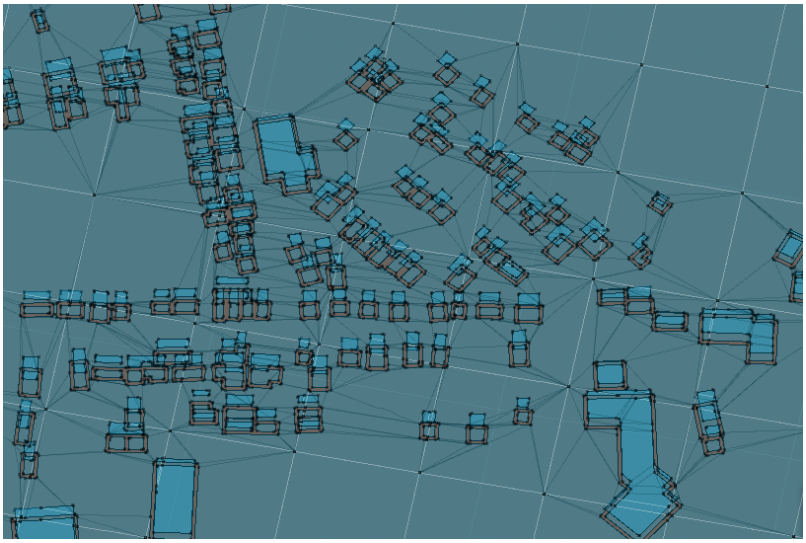

Figura 104. NavMesh del mapa

10 RESULTATS

10.1 LEGISLACIÓ I NORMATIVA VIGENT

Aquest projecte no presenta cap problema pel que respecte a la normativa vigent i des del punt de vista legislatiu.

Com que el videojoc no utilitza ni tracte amb dades de caràcter personal de l'usuari, no s'ha necessitat tenir en compte la llei orgànica de protecció de dades de caràcter personal (LOPD).

Pel que respecta a la llei de serveis de la societat de la informació i comerç electrònic (LSSICE), el projecte tampoc presenta cap problema ja que no desenvolupa cap activitat econòmica.

10.2 CAPTURES DE PANTALLA

En aquest apartat es veuran diferents captures de pantalla del videojoc. Les captures *In Game* s'han fet mentre es feia l'acció i clicant el botó de pausa.

La primera captura de totes, el primer que veurà l'usuari al obrir el joc, el menú principal, Figura 105.

Figura 105. Menú principal

Seguidament, es mostraran els quatre mapes disponibles en vista aèria; Banyoles Figura 106, Breda Figura 107, Vidreres Figura 108 i Ulldecona Figura 109. Els problemes visuals que es poden veure a les següents captures de pantalla són a causa que hi ha dos elements molt propers. Tot i que es veuen en aquetes captures de pantalla, en el joc hi ha la distància suficient com per a que aquests problemes no hi siguin.

Figura 106. Captura de pantalla mapa Banyoles

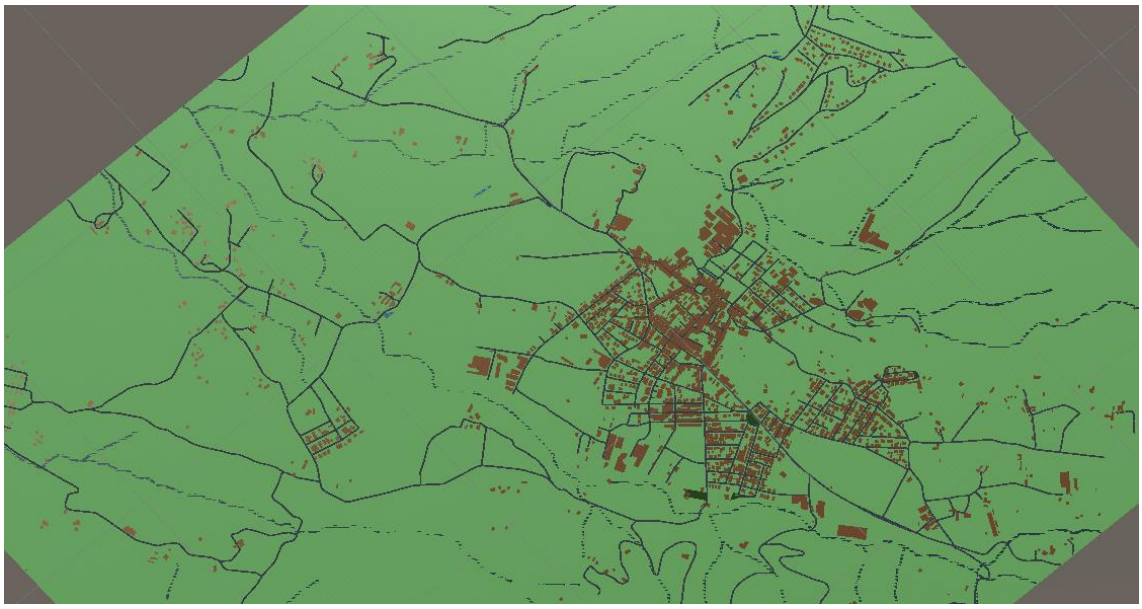

Figura 107. Captura de pantalla mapa Breda

Figura 108. Captura de pantalla mapa Vidreres

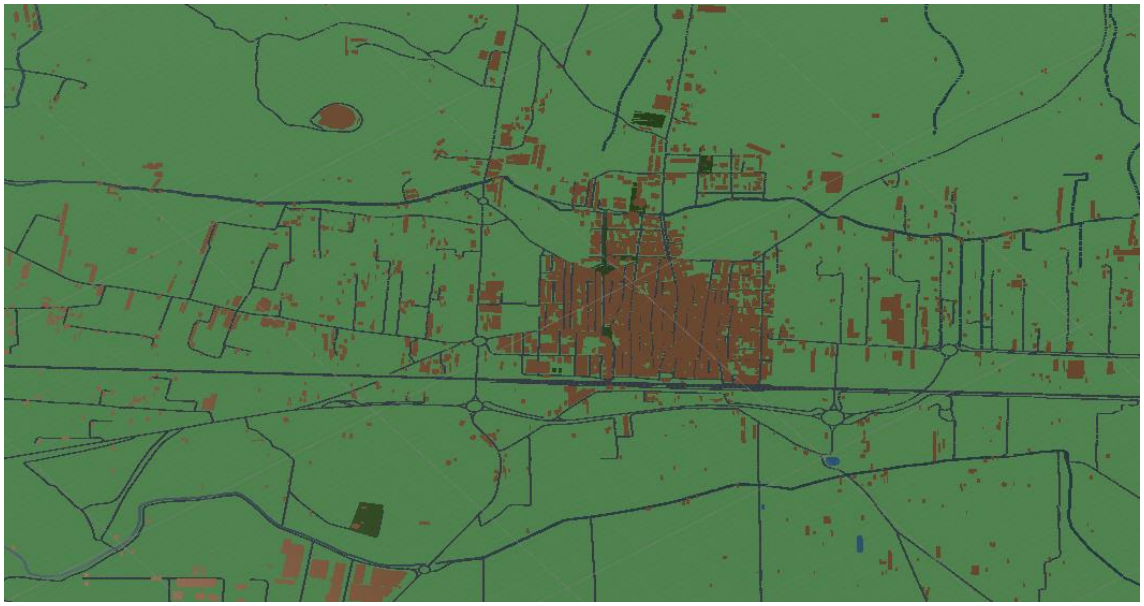

Figura 109. Captura de pantalla mapa Ulldecona

Si mirem la Figura 110, es pot veure el personatge havent acabat d'aparèixer en una carretera aleatòria del mapa de Banyoles.

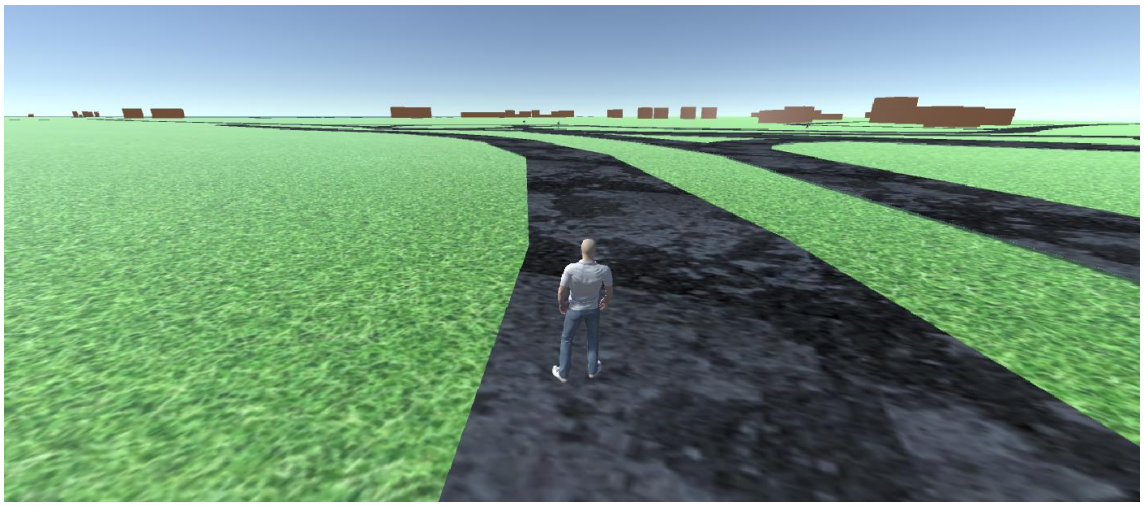

Figura 110. Personatge aparegut a carretera aleatoria

A la Figura 111, es veu al personatge caminant i l'ús de la càmera lliure. Darrera del personatge també es poden veure varis enemics acostant-se al personatge. A la Figura 112 es pot veure el personatge corrent, perseguit per enemics.

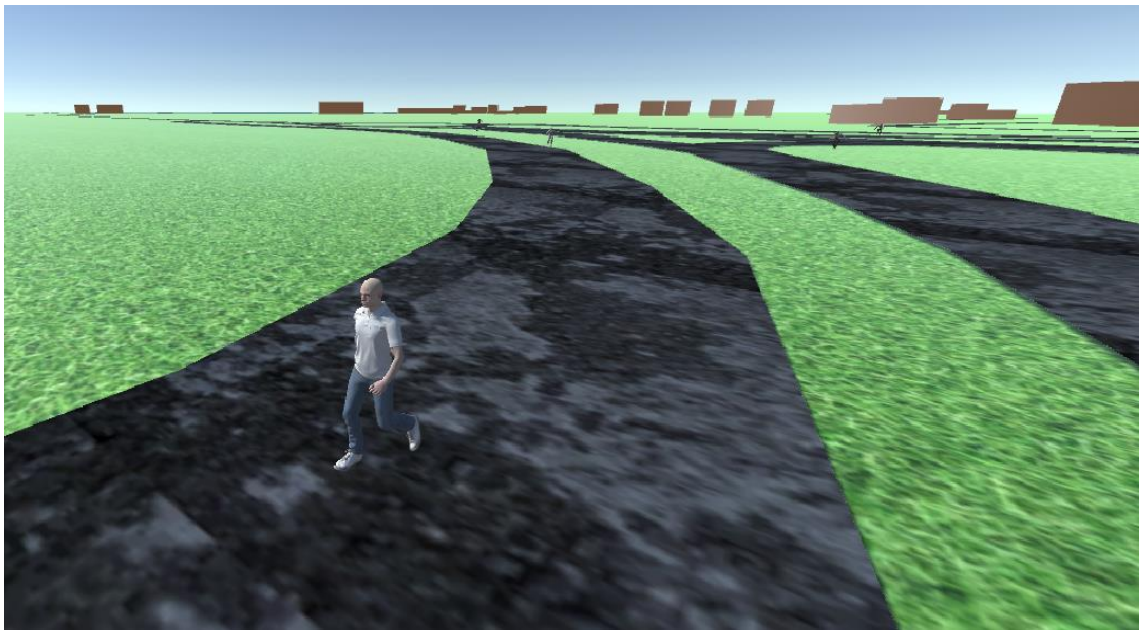

Figura 111. Personatge caminant i ús càmera lliure

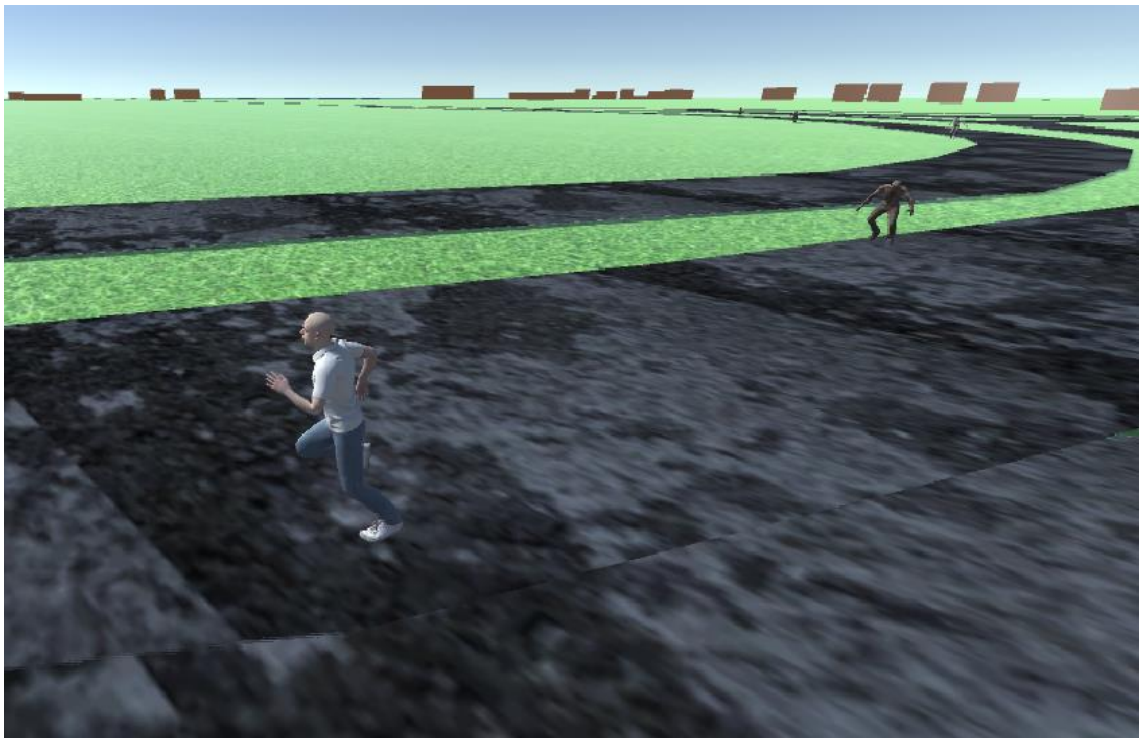

Figura 112. Personatge corrent

A la Figura 113, es pot veure al personatge preparat per atacar als enemics que s'acosten, mentre que a la Figura 114 es pot veure l'animació d'atacar.

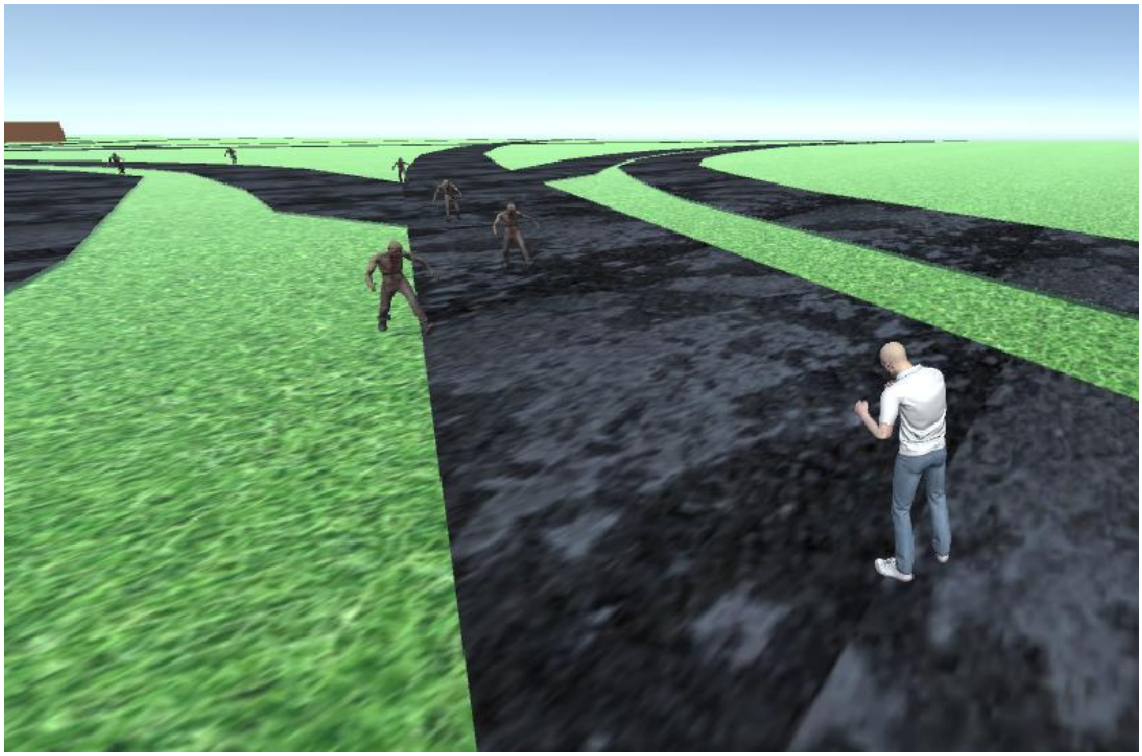

Figura 113. Personatge esperant per atacar

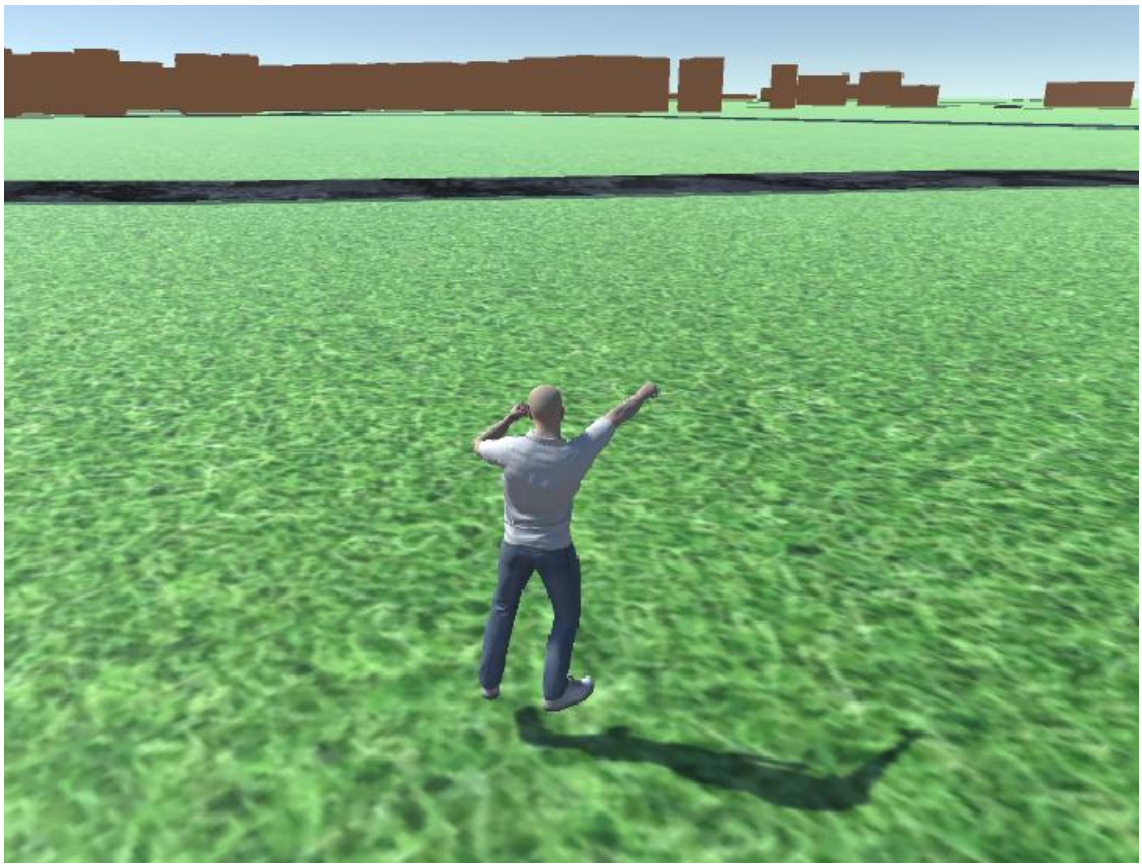

Figura 114. Personatge atacant.

A la Figura 115 es pot veure com un enemic ha atacat al personatge i aquest s'ha quedat sense punts de vida, per tant s'està executant l'animació de morir. A la Figura 116 es pot veure el menú de mort.

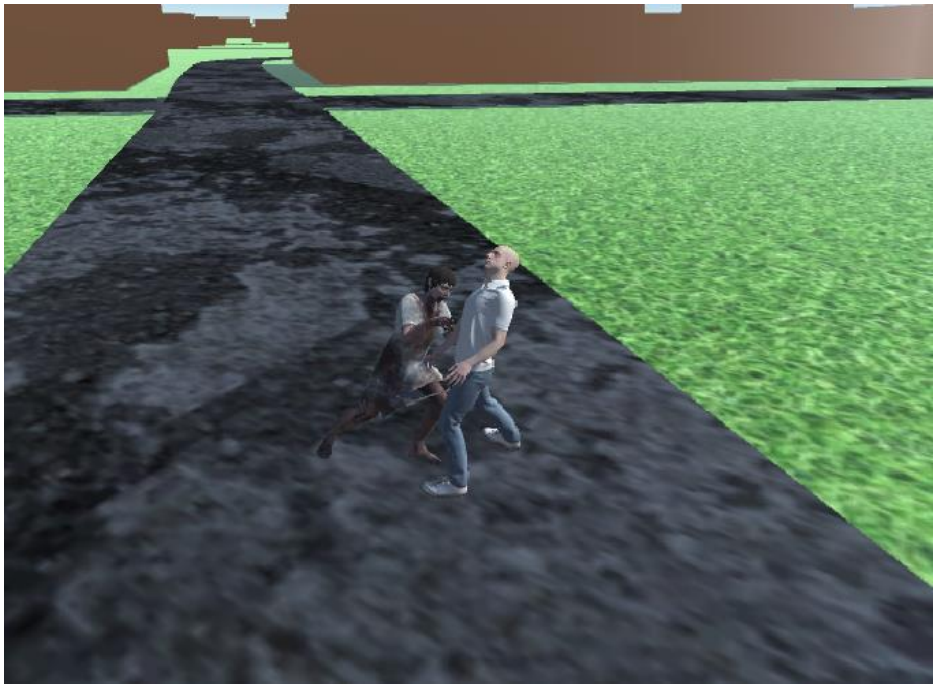

Figura 115. Enemic atacant, personatge morint

Figura 116. Menú de mort

A la següent imatge, la Figura 117, es pot veure al personatge atacant a un enemic, matant-lo. Aquesta captura de pantalla s'ha tret des del mode escena en comptes de des de la pestanya Game. A la Figura 118, es pot veure a l'enemic que s'ha atacat, al terra, sense punts de vida.

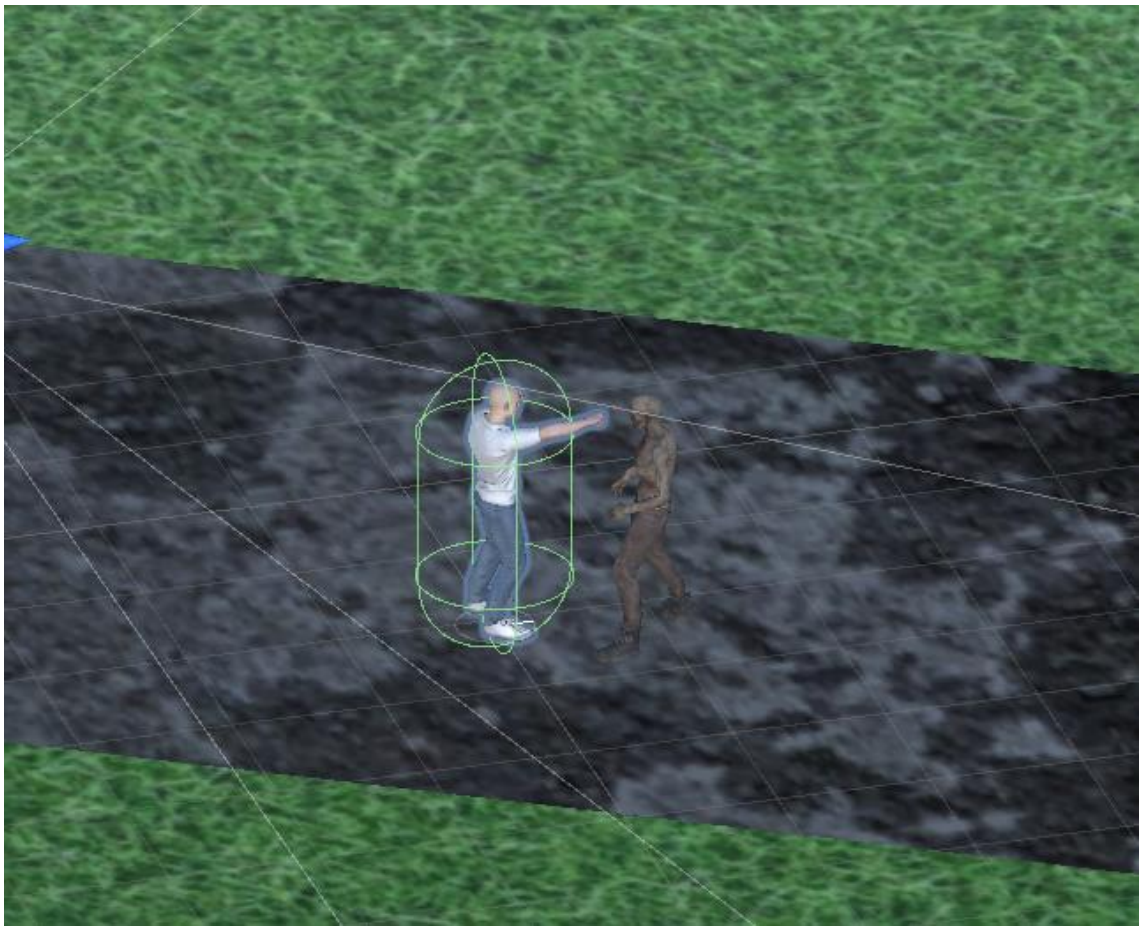

Figura 117. Personatge atacant a enemic

Figura 118. Enemic mort al terra

A la Figura 119 es pot veure el menú per canviar de població. A la Figura 120, es veu el personatge en una població nova, en aquest cas a Breda.

Figura 119. Menú canviar població

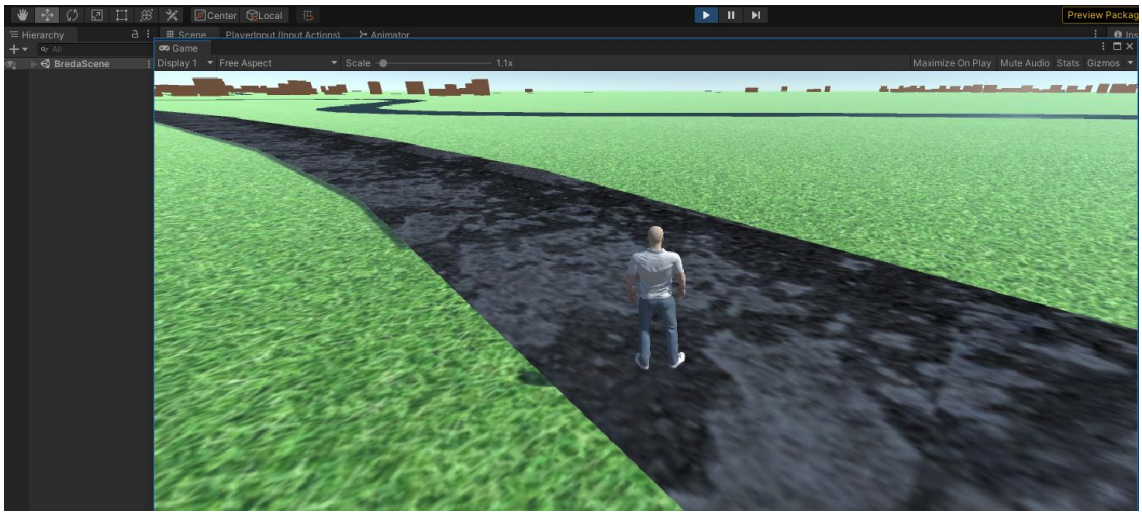

Figura 121. El personatge ha canviat a Breda.

Figura 120. Enemic acabat d'aparèixer

11 CONCLUSIONS

Aquest projecte és un dels projectes més complets que he realitzat mai, tot i no estar ni de bon tros acabat. Continuo tenint moltes idees que m'agradaria anar implementant al joc però que per falta de temps no he pogut dur a terme. Tot i així, al ser dels projectes més grans que he realitzat, també, i al fer-lo jo sol, he aprés molt de la meva manera de treballar.

Un dels aspectes en el que he aprés més, ha estat en l'aspecte de tenir més paciència a l'hora de buscar informació per internet, no tenir por a provar possibles solucions que trobi i sobretot, no esperar a trobar la solució perfecta. Això també ha fet que pari molta més atenció a les solucions que busco i entendre-les millor per, d'aquesta manera, poder agafar diferents idees que he trobat i ajuntar-les en una solució que funcioni en el projecte.

Un altre dels aspectes que m'ha ajudat ha estat a perdre més la por a equivocar-me. És un projecte gran i errors se n'han comés molts, el que no vol dir que no s'hagin solucionat o s'hagi trobat una altre manera d'afrontar el problema.

En general també he aprés a estar més concentrat en el que faig. El fet d'estar més concentrat en el projecte m'ha ajudat, també, a explorar GMP i Unity més detalladament descobrint funcionalitats que anteriorment no sabia i que han estat útils en el projecte.

Un altre gran aspecte que he millorat realitzant aquest projecte és la organització. Tot i tenir el projecte mitjanament pensat des de feia temps, dur-lo a terme és posar-hi moltes hores ja sigui per aprendre, dissenyar o desenvolupar. Sumant el fet de havia de combinar els dies amb les classes de la universitat, m'ha obligat a millorar la meva organització. He aprés molt a identificar millor les prioritats, quan i quin temps dedicar-hi.

Una altre part en la que he millorat molt, és en la part de dissenyar algorismes. Aquest és un aspecte que se'ns ensenya a la carrera però que encara abans de començar el projecte, no era un dels meus punts forts. El fet de, al final, veure que val la pena passar-se una bona estona dissenyant l'algorisme que es vol desenvolupar, m'ha fet guanyar moltes hores de desenvolupament.

Tenint en compte tots els aspectes, durant aquest projecte he aprés moltíssim i, com s'ha explicat al principi de la memòria, desenvolupar un videojoc com aquest és un objectiu personal i dur-lo a terme m'ha fet molta il·lusió. També el fet de desenvolupar un videojoc des de 0 amb, des del meu punt de vista, tant potencial m'anima a continuar aprenent i desenvolupant-lo. El fet de que cregui que té tant de potencial, és el fet que a partir de la base d'extreure les poblacions que es vulguin, és fer volar la imaginació i posar-se a desenvolupar.

També m'ha donat la confiança com per posar-me a detallar i algun dia desenvolupar altres projectes que tinc pensats.

Pel que fa al desenvolupament del projecte, al principi vaig veure GMP com una plataforma molt llunyana i poc accessible per una persona que encara no ha acabat la carrera com jo, el que em va fer perdre molt de temps buscant documentació que amb prou feines existia. Tot i així, després de explorar la plataforma pel meu compte, canviant opcions i modificant scripts, he aconseguit entendre com funciona la plataforma. Un altre aspecte, aquesta totalment fora del meu abast, és que GMP s'ha actualitzat i com s'ha comentat en aquesta memòria, hi ha eines

utilitzades que ja no estaran disponibles, a més del fet de que la plataforma sigui de pagament, encara que existeixi un període de prova gratuït, que no es pot ampliar. Això ens pot donar una pista que GMP pot no ser la millor eina per desenvolupadors que no volen comercialitzar el seu joc.

En referència als resultats esperats explicats a l'Apartat 4.3, tot i que no s'ha pogut complir tots, per exemple no es pot atacar amb armes de foc, estic molt satisfet amb el resultat de la resta. La raó de que algun dels resultats esperats no s'hagin implementat és que, a mesura que avançava el projecte, es prioritzaven altres parts per millorar el resultat final.

Tot i així, aquest videojoc és una bona base per continuar desenvolupant-lo o prendre'l com a referència per a futurs jocs ambientats en entorns reals ja siguin jocs de supervivència o qualsevol altre gènere, el que vol dir, que és un estil de videojoc que pot guanyar-se un lloc en la indústria.

12 TREBALL FUTUR

Des d'un primer moment es va pensar aquest videojoc per poder-se desenvolupar de moltes maneres diferents i poder afegir moltes millores.

Respecte a l'estat actual del videojoc, la primera millora que s'ha de realitzar, és la implementació del so. Com s'ha explicat al principi d'aquets memòria, el so és una part molt important dels videojocs i no s'ha implementat en aquest. El motiu principal és que, des del punt de vista del projecte, era més interessant refinar algunes parts de la creació del mapa o del comportament del personatge i dels enemics abans que el so.

Una segona millora que s'ha d'implementar és el fet que el videojoc, tal com es troba actualment, no té cap objectiu. Seria interessant dissenyar una petita història o dissenyar una sèrie d'objectius per animar al jugador a explorar el mapa. El que també fa pensar que una altre millora, és la implementació d'un inventari i de diferents objectes que el jugador es pugui trobar en el mapa.

Una altre millora és respecte a l'aspecte del videojoc. S'ha aconseguit posar textures a gairebé tots els elements menys a les sctructures que, al final, són un dels elements que es repeteixen més en el mapa i que actualment tenen un color pla. Per tant, estudiar com funciona el sistema de UV i crear una array de UV personalitzada per cada structure, és una millora molt interessant a implementar. També, relatiu a aquest tema, millorar visualment els menús.

També, respecte a l'aspecte del videojoc, una altre part a millorar és tant l'elevació del terreny, que actualment no n'hi ha, i la implementació de vegetació. Ara mateix, els mapes són completament plans i una millora a realitzar seria trobar una plataforma d'on es podés extreure dades de l'elevació del terreny per implementar-ho en el videojoc. Respecte a la vegetació, en el mapa, ara mateix no existeix, només hi ha els elements de GMP, però segurament es pot trobar alguna base de dades pública on hi hagi dades de les zones forestals. Fent només aquestes millores, el joc guanyaria molt nivell de qualitat.

Com s'ha explicat a l'apartat de conclusions, tinc la intenció de continuar desenvolupant el videojoc però es presenta el problema de les actualitzacions que ha tingut GMP i el fet que, al final, és una plataforma de pagament. Per tant, una altre millora pel projecte seria trobar una nova plataforma, si pot ser gratuïta, per poder disposar de les dades sense preocupacions de tenir un límit de temps i/o diners. No obstant, en el cas de continuar amb la plataforma GMP, s'hauria de trobar una altre manera d'extreure les dades. La que s'ha implementat en aquest projecte és correcta, però queda lluny de ser perfecta ja sigui perquè les dades s'extreuen movent-se pel mapa, es poden repetir dades o poden quedar zones que no s'han carregat i que per tant, no es guardin.

Una altre part a millorar són les animacions. Gràcies a la plataforma de Mixamo s'ha facilitat molt la feina d'aquesta part. Tot i així, les animacions que es troben en diferents capes, a l'hora de combinar-se no s'acaben de combinar correctament, el que fa que quedin descentrades. Això en part s'ha solucionat utilitzant Blender, que ha permés modificar les animacions individualment per així quan es combinen quedin més naturals.

Pel que és referent a les millores dels scripts, tot i haver quedat content amb el resultat, hi ha parts que es podrien millorar. Una d'elles són algunes funcionalitats que es fan crides creuades

entre classes i no queda clar quina és la classe responsable de la funcionalitat. Això pot ser confús a l'hora d'entendre el funcionament o haver de modificar el codi.

13 BIBLIOGRAFIA

Unity Technologies. “Unity User Manual 2020.3”. *Unity Documentation.* 23 desembre 2021. https://docs.unity3d.com/Manual/index.html/. 2021.

Unity Technologies. “Scripting API”, *Unity Documentation.* 23 desembre 2021. https://docs.unity3d.com/ScriptReference/index.html. 2021 .

Stack Overflow. Stack Exchange, 15 setembre 2008. https://stackoverflow.com/. Accedit 2021.

Unity Technologies. *Unity. 2020.* https://answers.unity.com/. Accedit 2021.

Unity Technologies. *Unity. 2020.* https://forum.unity.com/. Accedit 2021.

Unity Technologies, *Unity Learn.* 2022. https://learn.unity.com/. Accedit 2021.

Ilkinulas, *UV Mapping (of a Cube),* GitHub, 6 març 2016, http://ilkinulas.github.io/development/unity/2016/05/06/uv-mapping.html, Accedit 2021.

Microsoft. “C# documentation”. *Technical documentation.* 2022. https://docs.microsoft.com/en-us/dotnet/csharp/. Accedit 2021.

Wikipedia. Jimmy Wales, Larry Sanger, Wikimedia community, 2001, http://en.wikipedia.org/. Accedit 2021.

Youtube. Google, 14 febrer 2005, https://www.youtube.com/. Accedit 2021.

J. Clement, *Video game industry – Statistics & Facts,* Statista, 19 novembre 2021. https://www.statista.com/topics/868/video-games/. Accedit 18 desembre 2021.

“Video Game Industry Statistics, Trends and Data In 2021”. *Wepc.* Wepc. 16 desembre 2021. https://www.wepc.com/news/video-game-statistics/. Accedit 18 desembre 2021.

Sean. *Video Game Genres by Year: 1980-2016.* Savvystatistics. 6 juny 2019. http://savvystatistics.com/video-game-genres-by-year-1980-2016/. Accedit 18 desembre 2021.

Tomić, Nenad. *Economic Model of Microtransactions in video Games.* Research Gate. Gener 2019. https://www.researchgate.net/figure/total-video-games-industry-revenue-with-shares-of-individual-market-segments-in-the_fig1_331674647. Accedit 18 desembre 2021.

Universitat de Girona. “Publicacions web”. *Biblioteca UdG.* Universitat de Girona. Juliol 2021. https://biblioteca.udg.edu/ca/estil-mla/publicacions-web. Accedit 3 gener 2022.

14 MANUAL D'USUARI

Per jugar al videojoc no és necessària cap mena d'instal·lació, ni la del propi videojoc.

14.1 CANVIAR POBLACIÓ

Per canviar de població, s'ha de portar al personatge a un dels extrems del mapa fins que s'obri el menú. Un cop en aquest menú, clicar a un dels signes d'exclamació i el videojoc portarà el personatge a la població escollida.

14.2 CONTROL PERSONATGE

A continuació s'expliquen els controls del personatge:

- **Moviment:** Per moure el personatge caminant, s'utilitzen les tecles W, A, S, D del teclat. Sent W moure's cap endavant, S moure's cap endarrere, A moure's a l'esquerra i D moure's a la dreta. Es poden combinar tecles per aconseguir més graus de moviment. Per córrer, s'utilitzen les mateixes tecles però mantenint clicat el botó shift del teclat. Per saltar, es pot estar en moviment i s'ha de clicar l'espai del teclat.
- **Càmera:** Per controlar el moviment de la càmera, es clicarà el botó central del ratolí (la rodeta) per habilitar la càmera lliure i el moviment del ratolí controla el moviment de la càmera.
- **Zoom:** Per ampliar o disminuir el zoom, s'ha estar amb la càmera lliure i llavors, moure la rodeta. Si es mou la rodeta en direcció al centre del ratolí es farà zoom out, altrament es farà el zoom in.
- **Atacar:** Per atacar el personatge primer ha d'estar apuntant. Per apuntar cal mantenir clicat el botó dret del ratolí. Un cop el personatge apunta, clicant el botó esquerra del ratolí, el personatge atacarà.