Universitat de Girona
Escola Politècnica Superior

Grau en Enginyeria Informàtica

PROJECTE FINAL DE GRAU

Disseny i desenvolupament d'un videojoc "rogue-like" d'acció en temps real

Autor:
Albert Falgàs Ribas

Tutors:
Gustavo Ariel Patow
Sergio Gonzalo Besuievsky Glikberg

MEMÒRIA

Convocatòria:
Setembre 2023

Departament :
Departament d'informàtica, matemàtica aplicada i estadística

Projecte: Projecte Final de Grau
Document: Memòria
Títol: Disseny i desenvolupament d'un videojoc "rogue-like" d'acció en temps real
Autor: Albert Falgàs Ribas
Data: Setembre 2023

Estudi:
Grau en Enginyeria Informàtica
Universitat de Girona

Supervisor 1:
Gustavo Ariel Patow
Universitat de Girona
Email: gustavo.patow@udg.edu
Web: UdG Plana personal - Dr. Besuievsky Glikberg, Sergio Gonzalo

Supervisor 2:
Sergio Gonzalo Besuievsky Glikberg
Universitat de Girona
Email: gonzalo.besuievsky@udg.edu
Web: UdG Plana personal - Dr. Patow, Gustavo Ariel

Índex

Capítol 1

Introducció, motivacions, propòsit i objectius del projecte

1.1 Introducció

La indústria del videojoc va adquirint més força a mesura que passen els anys, naixent grans empreses que llencen de manera anual projectes de pressupostos bilionaris a mesura que aquest hobby es va estenent. Aquest fet ha desembocat en què el consumidor tingui uns estàndards de qualitat cada vegada més elevats, així com aquest exigeixi una gran oferta en hores de joc per tal satisfer-lo.

Tot això, però, no ha aturat a petits equips de desenvolupadors de llençar les seves propostes al mercat i inclús han aconseguit una gran rellevància i referència en la indústria tot i competir amb els gegants de la indústria, els anomenats "doble A (AA)" i "triple A (AAA)".

1.2 Motivacions

El procés de creació d'un videojoc consisteix en la suma i coordinació de múltiples disciplines molt diferents entre elles. Això fa que siguin productes complexos els quals solen requerir un equip gran i divers per poder produir un producte a l'altura dels estàndards de qualitat i extensió que avui dia existeixen en la indústria.

En els darrers anys hem pogut observar, però, el naixement de diversos projectes independents o "indies", els quals sense el suport d'una gran empresa i amb un petit equip de persones o inclús una sola, han esdevingut obres de referència d'una enorme qualitat i originalitat.

Aquests jocs sovint destaquen per l'ús de tècniques de generació procedimentals, les quals fan recaure sobre el desenvolupador la càrrega més gran de feina, ja que aquest pot expandir la duració i atractiu del joc, delegant la combinació aleatòria de diferents components a algoritmes. El fet de reduir la dependència d'àrees alienes a la programació fa que aquests tipus de projectes siguin ideals per a petits equips que volen entrar dins la indústria, amb propostes que destacaran per la jugabilitat, però que no tenen per què decaure en temes d'atractiu artístic, ja que aquests no han de ser produïts de manera massiva.

1.3 Motivacions de caràcter personal

Molt abans de plantejar-me cursar el grau d'Enginyeria Informàtica, vaig arribar a la conclusió que volia formar part de la indústria dels videojocs en alguna de les seves àrees. Això em va impulsar a realitzar un grau superior que em va permetre adquirir les bases de disciplines essencials en la producció de videojocs tan diverses com: el modelatge 3D, dibuix, programació, animació, texturització, "rigging", disseny. Aquest mateix període de la meva vida em va fer explorar els anomenats jocs independents o "indies", els quals presentaven jocs produïbles per un petit equip o inclús per una única persona. En altres paraules, projectes que permetien als nous desenvolupadors de videojocs a donar-se a conèixer.

Propostes com el videojoc *The binding of Isaac* [1] , d'Edmund McMillen, em van sorprendre per la seva enorme rejugabilitat, la qual supera en gran manera la majoria d'obres de grans estudis de videojocs, però pel contrari havia estat desenvolupat per una sola persona. El que més es desprenia de l'èxit en rejugabilitat d'aquest videojoc era la modularitat, la qual combinada amb aleatorietat, permetia que cada partida fos diferent, tant pels nivells a explorar com pels poders que el jugador va obtinguen al llarg de la partida.

FIGURA 1.1: Captura del videojoc *The binding of Isaac*.

Un altre concepte que em fascinava era, la relativa poca quantitat de recursos artístics que aquests jocs realment requerien, cosa que, en la meva opinió, permetia fer un enfocament artístic més enfocat en la qualitat que en la quantitat, obtinguen així jocs prou atractius a l'hora de vendre però podent dedicar la majoria dels recursos a la jugabilitat.

Tot això va fer interessar-me per tota l'algorítmica que hi ha al darrere a l'hora de combinar els diferents mòduls que construeixen un joc amb aquestes característiques, així com els requisits que han de tenir aquests mòduls, per tal de poder ser usats com a blocs de construcció. És per això que vaig decidir entrar a Enginyeria

informàtica, i així reunir els coneixements per a realitzar jocs d'aquesta complexitat tècnica i explorar-ho amb aquest projecte.

1.4 Propòsit

El propòsit d'aquest projecte és desenvolupar un primer prototip de videojoc que faci ús de tècniques de generació per procediments per tal d'expandir-ne la duració i jugabilitat. Això s'aconseguirà, concretament amb la generació de nivells que consistiran en sales interconnectades entre si de manera aleatòria, amb un inici i un final. Tot això dins d'un marc de videojoc de perspectiva superior de tipus "roguelike" de trets, inspirat en gran manera amb els videojocs *Enter the Gungeon*[2] i *The binding of Isaac*.

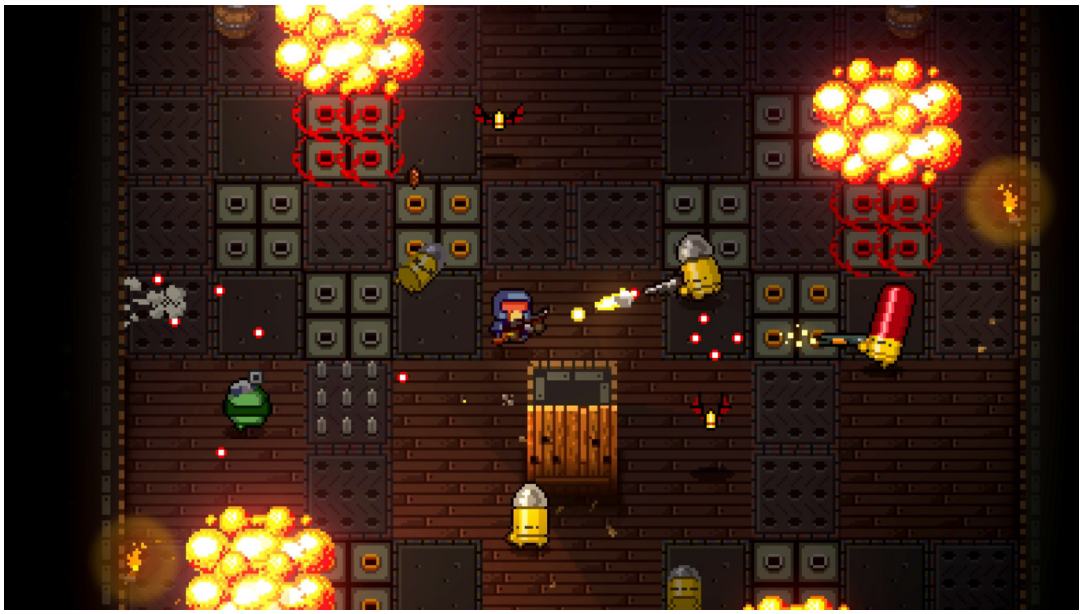

FIGURA 1.2: Captura del videojoc *Enter the Gungeon*.

1.5 Objectius del projecte

Aquest projecte se centra en els següents objectius:

- Estudi del motor *Unity*[3] i del llenguatge C#.
- Estudi, disseny i implementació de mecanismes que expandeixin la jugabilitat minimitzant la creació de nous elements específics per aquest propòsit (models d'enemics, trampes, armes, terrenys...).
- Generació de nivells aleatoris dividits en sales interconnectades.
- Sistema de moviment i interacció del jugador amb l'entorn.
- Implementació d'enemics amb tècniques d'intel·ligència artificial.
- Estudi i creació de "shaders".
- Desenvolupament de menús.

- Desenvolupament de la interfície de joc.

1.6 Breu introducció als "rogue-like"

El terme "rogue-like" consisteix en un subgènere de videojocs que, a grans trets, es caracteritzen per un cicle de joc marcat per la mort permanent del jugador, la qual obliga a reiniciar la partida després de cada intent fallit. A diferència de clàssics jocs àrcade com *Ghosts'n Goblins* [4], que també consisteixen a arribar al final del joc en una mateixa partida, aquests no acostumen a implementar un sistema de vides que disminueixi la pèrdua de progressió en una mateixa partida. Addicionalment, també solen buscar que cada partida sigui diferent, sovint implementant tècniques de generació per procediments a l'hora de construir els nivells.

FIGURA 1.3: Captura del videojoc *Ghosts'n Goblins*.

Aquest subgènere rep el nom gràcies al joc per a terminal *Rogue* [5] de l'any 1980, el qual va ser el primer a introduir els aspectes anteriorment mencionats de la mort permanent i la generació aleatòria dels nivells.

En l'actualitat, però hem vist com és més comú que els jocs d'aquest tipus suavitzin l'aspecte de la mort permanent lligat al reinici total de la progressió. Ens referim a aquests jocs com a "rogue-lites". Aquesta suavització s'aconsegueix introduint sistemes de progressió que es mantenen entre les partides, sovint en forma d'estadístiques bàsiques, personatges i habilitats desbloquejables, d'entre altres. Afegir aquests sistemes, no només incrementa més la rejugabilitat d'aquests títols sinó que també serveix com un sistema que permet al jugador visualitzar les fites que ha obtingut per a poder obtenir els desbloquejables. Un altre aspecte a destacar és que, aquests poden ser utilitzats per a balancejar el joc per a més nivells d'habilitat dels jugadors, de manera que, per exemple, un jugador que ha jugat molt al títol, però que per habilitat, no assoleix superar certs reptes, pugui obtenir millores que facilitin aquests reptes en els següents intents.

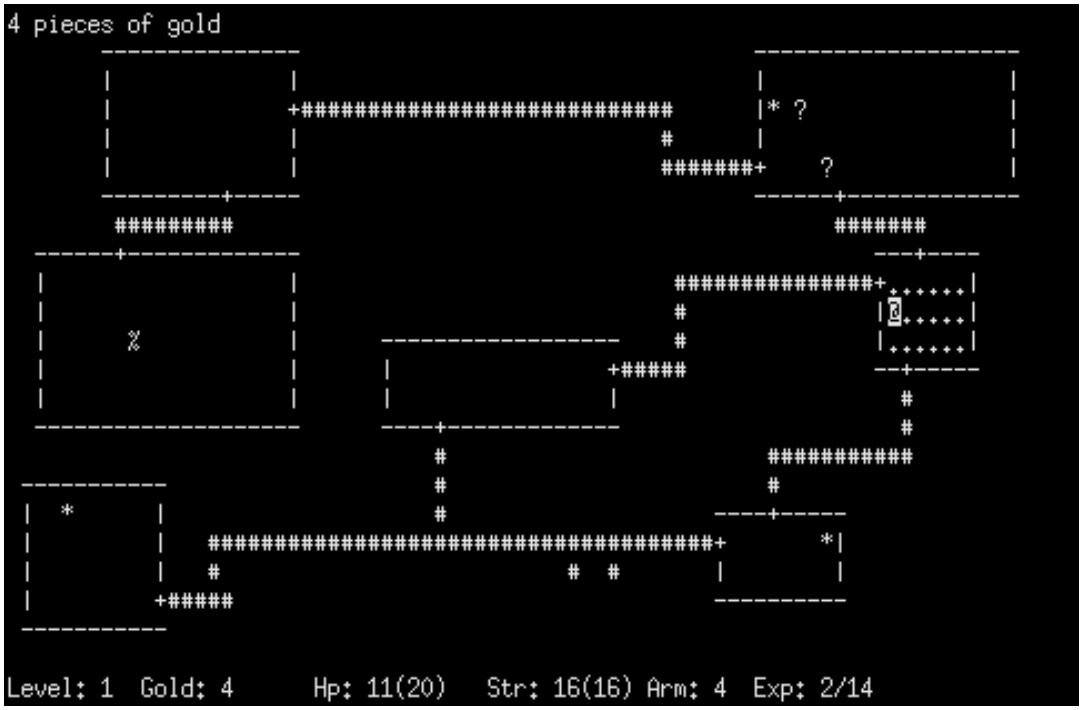

FIGURA 1.4: Captura del videojoc *Rogue*.

Capítol 2

Estudi de viabilitat

Per tal de desenvolupar aquest projecte s'ha fet us del següent programari:

- *Windows 10 Pro N*: Sistema operatiu de l'equip utilitzat
- *Unity*: Motor multiplataforma per a desenvolupament de videojocs.
- *MediBang Paint Pro*[6]: Programa d'edició d'imatges i dibuix digital.
- *Inkscape*[7]: Programa per dibuix vectorial.
- *Maya*[8]: Programa per a animació i modelatge 3D.
- *Visual Studio*[9]: Entorn de Desenvolupament Integrat (IDE).

L'equip utilitzat consistia en un ordinador de sobretaula amb les següents característiques a destacar:

- Processador: Intel(R) Core(TM) i7-6700k cpu @ 4.00GHz 4.01 GHz
- Targeta gràfica: NVIDIA GeForce GTX 1070
- Memòria instal·lada (RAM): 16 GB

2.1 Pressupostos Inicials

En matèria d'inversions inicials, per aquest projecte, només hem de contemplar la llicència de *Maya* de 2.245€ anuals, tot i que s'hauria pogut escollir *Blender*[10] com a alternativa completament gratuïta. Cal dir però, que *Maya* compta amb una llicència gratuïta per a estudiants, sempre que no es comercialitzi el producte.

Pel que fa a l'equip, no s'ha comptat, ja que s'ha fet servir un ordinador d'ús personal i no ha calgut cap inversió en el mateix per al desenvolupament del projecte.

Cal afegir que, per publicar el joc a la plataforma *Steam*, també caldria una inversió d'uns 100$ i que aquesta plataforma també es quedaria un 30% dels guanys. Addicionalment, si els ingressos del videojoc superessin els 100000$ s'hauria de pagar una llicència de *Unity* "Plus" de 40$ i si aquests superessin els 200000$ seria necessària una llicència "Pro" de 185$ mensuals.

2.2 Recursos humans

Un videojoc comportat tant un repte d'enginyeria , pel que fa a la complexitat dels seus sistemes, com també un gran repte de disseny, ha de estar especialment dissenyat per ser divertit, aixi com ha de ser prou atractiu en els apartats mes artístics per tal de vendre.

Es per aixo que per la producció del producte final caldria un equip multidisciplinari que cobreixi els següents rols:

- Cap de projecte
- Analista
- Programador
- Dissenyador
- Generalista 3D (Modelatge, texturització, "rigging", animació)
- Il·lustrador
- Compositor
- Artista d'efectes de so

Cal dir però que aquest treball s'ha limitat a cobrir el disseny, anàlisis i programació del joc, tot i que també degut a la meva formació prèvia com a generalista 3d i il·lustrador, s'han afegit de manera parcial recursos artístics amb els quals comptaria el producte final.

Capítol 3

Metodologia

En aquest projecte hem optat per una metodologia en cascada [11] parcial, per al desenvolupament de les parts principals del projecte. Aquesta, ha consistit a dividir el projecte en 2 grans blocs, generació aleatòria del nivell i controls del personatge, per tal d'abordar-los com a dos sistemes diferents i independents. Aquesta metodologia ha consistit en:

1. Anàlisi: Fase on s'identifiquen les característiques, funcionalitats i objectius del sistema a crear.

2. Disseny: Fase d'estructuració dels diferents components del sistema i la integració amb la resta d'elements del videojoc.

3. Implementació: Creació del codi i recursos artístics.

4. Prova: Fase on s'executa de manera parcial o total el videojoc per tal de detectar-ne possibles errors i corregir-los

5. Desplegament: S'afegeix el sistema complet al projecte principal.

6. Manteniment: A mesura s'integren noves parts, es determina si cal actualitzar el sistema.

Amb el projecte prou avançat hem decidit utilitzar una metodologia en espiral [12] per a realitzar iteracions en el projecte que afegissin noves i més petites funcionalitats, així com realitzar correccions d'errors de comunicació entre les parts. Pel que fa a aquesta altra metodologia, la podem dividir en les següents fases:

1. Determinació d'objectius: Es defineix que es vol resoldre amb el nou component o funcionalitat.

2. Anàlisis de riscos: S'avalua els problemes que es poden derivar a causa de la implementació i es busquen solucions per mitigar-los.

3. Desenvolupament i proves: Es desenvolupa la funcionalitat i es duen a terme proves per tal de comprovar-ne el funcionament.

4. Planificació: Observant els resultats obtinguts de la fase anterior es planifica una possible següent iteració per tal de corregir errors o expandint-ne la funcionalitat.

Capítol 4

Planificació

L'inici del desenvolupament del projecte es remunta el Juny de 2022. Aquesta data, però, no inclou la concepció de la idea del projecte, la qual va néixer abans de la realització d'aquest grau, ni tampoc les reunions extraoficials amb els professors que més endavant van esdevenir tutors.

Les tasques principals del projecte han estat les següents:

- Generació de l'estructura d'un pis
- Generació interna bàsica d'una sala
- Importació de personatges i sistema d'animacions
- Controls del personatge
- Sistema d'armes de foc
- Sistema de bales i patrons
- Sistema d'habilitats
- IA dels enemics amb "behaviour trees"
- Menús
- HUD
- Flux de joc
- Correcció d'errors
- Documentació

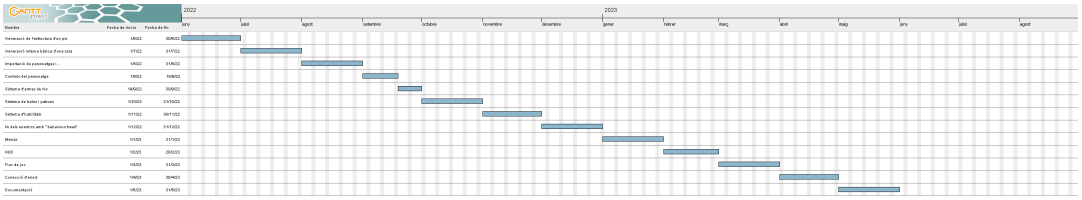

FIGURA 4.1: Diagrama de *Gantt* de la planificació original.

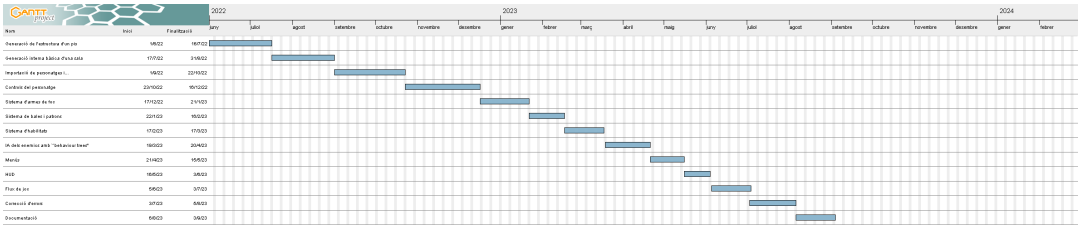

FIGURA 4.2: Diagrama de *Gantt* de la temporització final.

4.0.1 Canvis en la temporització inicial

Com podem veure en les figures anteriors, la temporització del projecte s'ha acabat estenent fins al setembre. Això és degut, a què en un inici es volia entregar per la convocatòria de juny. Això però, es va voler canviar, posant com a objectiu la següent convocatòria, per així poder profunditzar més en els objectius plantejats en la introducció (capítol 1).

Capítol 5

Marc de treball i conceptes previs

Aquest capítol té per objectiu fer una breu introducció a aspectes que són imprescindibles per a la comprensió dels diferents aspectes que suposen el desenvolupament d'un videojoc d'aquestes característiques.

5.1 Motors de videojocs

A causa de la complexitat tècnica d'un videojoc, en l'àmbit comercial es desenvolupen sempre sobre un motor de videojocs, el qual pot ser comercial o propi de l'empresa desenvolupadora. Un motor és un entorn de desenvolupament que proporciona al desenvolupador un conjunt d'eines que permeten a aquests centrar-se a alt nivell en el qual serien les mecàniques i flux de joc. Entre les eines que sovint trobem en els més coneguts motors de videojocs, podem destacar-ne les següents:

- Motor de físiques.
- Motor gràfic.
- Abstracció de "hardware" per al desenvolupament multi-plataforma.
- Editor visual.

Aquests aspectes, a part de la comunitat i documentació al darrere, solen ser els aspectes més decisius i crítics a l'hora d'escollir un motor de videojocs que s'adapti a les necessitats del nostre projecte.

5.1.1 Exponents rellevants

En aquesta subsecció veurem alguns dels motors més utilitzats, tant per a projectes independents, com. fins i tot, per estudis de videojocs consolidats.

Godot

Es un projecte de codi obert, completament gratuït i finançat a través de donacions. Va ser desenvolupat el 2001 per un estudi de videojocs Argenti tot i que va ser reescrit i llençat com a codi obert el 2014.

Godot[13] ofereix la capacitat de crear videojocs tant en 2D com en 3D de qualsevol classe i poder-los exportar fàcilment a ordinador d'escriptori, dispositius mòbils i

web. També és possible desenvolupar per a consoles amb ell, però no ofereix les mateixes facilitats.

Pel que fa a llenguatges de programació, ofereix la possibilitat tant de treballar amb *C#* com també amb un llenguatge específic d'aquest motor anomenat *GDScript*.

FIGURA 5.1: Logo del motor*Godot*.

Unreal

En l'actualitat és un dels motors més populars i utilitzats. La primera versió va sortir el 1998 per *Epic Games* però no va ser fins a 2015 que es va llençar de manera pública. Pel que fa a la monetització, la utilització d'aquest motor és completament gratuïta, però té un sistema de "royalties" d'un 5% dels beneficis en cas que la comercialització d'un producte desenvolupat amb ell superi els 1.000.000€.

Quan parlem de què ofereix *Unreal*[14], parlem d'un paquet complet que permet desenvolupar videojocs de tota classe, tan 2D com 3D, i exportar-los a un gran nombre de dispositius com, ordinadors d'escriptori, dispositius mòbils, web i fins i tot consoles de nova generació. A causa de la seva gran versatilitat, *Unreal* també destaca per ser utilitzat per a crear altres aplicacions a banda dels videojocs. Algunes d'aquestes aplicacions a destacar en són els simuladors i aplicacions de realitat virtual. Un altre aspecte a destacar es que es un motor ideal per a desenvolupar aplicacions amb gràfics altament foto-realistes, a causa del funcionament del motor gràfic.

Aquest motor ofereix tant la possibilitat de programar amb "C++" com fer servir un editor visual de "blueprints". Aquest editor permet crear la totalitat del codi mitjançant un sistema de blocs i diagrames de flux, essent una opció molt popular per a creadors amb pocs coneixements de programació.

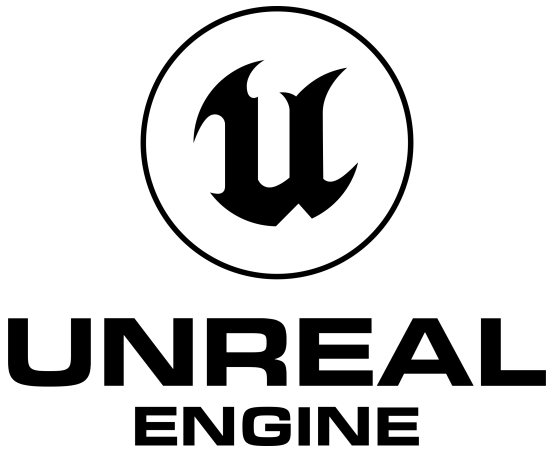

FIGURA 5.2: Logo del motor*Unreal*.

Unity

Unity és un motor enormement popular i amb la comunitat probablement més gran. Va llançat el 2005 per *Unity technologies* com a un programa gratuït amb versions de pagament obligatòries en cas de percebre ingressos superiors a uns llindars. Si se superen els 100.000$ cal una llicència ”Plus"de 40$ i si se superen els 200.000$ cal una llicència ”Pro"de 185$ mensuals.

Aquest motor, a banda d'oferir un paquet complet per al desenvolupament de videojocs i l'exportació multidispositiu, que ofereix de manera molt similar a *Unreal*, destaca per ser el motor amb més quantitat d”'assets” descarregables a través de la seva ”asset store”, a causa de la seva gran comunitat.

Pel que fa a la programació *Unity* ofereix un sistema de ”scripting” amb C# i també fa ús de *ShaderLab* per a la programació de “shaders” i efectes visuals personalitzats. Alternativament, també es pot programar en *UnityScript* o *Boo*, però aquests llenguatges han anat caient en desús, afavorint la programació en *C#*.

També cal dir que l*API* que ofereix és molt fàcil d'entendre i està molt ben documentada a la pàgina web de *Unity* [15].

FIGURA 5.3: Logo del motor*Unity*.

5.1.2 Motor escollit

Tot i que *Unreal* és una opció molt bona i no caldria pagar si els beneficis no superin els 1.000.000€, m'he decantat per *Unity*.

El motiu principal d'aquesta elecció és el meu coneixement previ del motor, així com del llenguatge *C#*. Tot i que el meu coneixement sobre el motor i el llenguatge es remuntaven a una època on mancava de suficient habilitat com a programador, conèixer el funcionament de diverses eines de l'editor gràfic, facilitaven enormement un gran conjunt de tasques. Aquest fet em donava l'oportunitat de poder dedicar més hores a l'elaboració del codi, sobretot dels algoritmes de generació per procediments que hauria de desenvolupar.

Un altre motiu per no haver-me decantat per *Unreal* és que el meu videojoc no busca uns gràfics foto-realistes sinó que busca uns gràfics d'estil “toon”.

Per acabar, he d'afegir que també resulta una opció amb un gran atractiu a causa de l'enorme facilitat que existeix tant per trobar informació oficial sobre el funcionament, així com la gran quantitat de guies oficials i extraoficials sobre desenvolupament de videojocs que existeixen.

5.2 Programes de producció 3D

Un programa de producció 3D consisteix en un entorn visual que ofereixen un gran conjunt d'eines per a la producció d'"assets" en 3D. Aquestes eines es solen agrupar en els següents grups:

- Modelatge: Creació de models consistents en una col·lecció de punts amb plans que els connecten. Aquest grup també inclou eines per a editar els "UVs" els quals són el model desplegat en un pla, cosa imprescindible per a l'elaboració de textures i l'aplicació de materials.

- "Rigging": Creació d'una estructura jeràrquica en forma d'esquelet que determina les articulacions d'un model 3D i com es mouen, així com la creació de controls per a realitzar-ne el moviment. És un pas indispensable per a poder crear animacions, sovint de personatges.

- Animació: Eines per editar les corbes de moviment d'interpolacions entre posicions claus d'un model 3D.

- Efectes especials: Edició de partícules, simulacions físiques de roba, fluids d'entre altres.

- Renderitzat: Edició de "shaders", materials i llums.

D'aquests grups d'eines, per a exportar cap a un videojoc, ens interessen evidentment el modelatge, però també el "rigging" i l'animació. Tot i que els principals motors de videojocs solen tenir mòduls o "plugins" per a poder realitzar el "rigging" o animacions, aquests no solen ser aconsellables per a l'elaboració de "riggings" o animacions complexes. Això fa que s'utilitzin sobretot per a afegir modificacions dinàmiques al "rigging" i animacions base.

5.2.1 Exponents rellevants

Seguidament, farem una anàlisi dels dos principals exponents a l'hora de produir recursos 3D que proporcionen més facilitats d'exportació a motors de videojocs.

Blender

És una opció, amb una gran comunitat al darrere que li dona suport, completament gratuïta i de codi obert llançat el 2002 per la *Fundació Blender*.

A causa de la seva naturalesa de codi obert, ens trobem amb una gran quantitat de "-plugins" que n'expandeixen les funcionalitats, fen que aquest programa, a part de la producció de recursos en 3D, serveixi per a coses tan variades com per exemple l'edició de vídeo. Un altre aspecte a destacar, és que, a causa de la seva gran comunitat, existeixen moltes guies i tutorials.

FIGURA 5.4: Logo del programa de producció 3D*Blender*.

Maya

Maya és una opció comercial de pagament desenvolupada per l'empresa *Autodesk* i llençat al mercat el 1998. Ofereix diversos models de pagament tant mensuals com anuals, essent l'anual de 2.245 € el més desitjable. També ofereix una versió gratuïta per a estudiants, però no permet la comercialització de cap producte produït amb aquesta llicència.

Maya ofereix un paquet molt clar i complet per a la producció de recursos en 3D, amb una separació molt ben delimitada entre els diferents fluxos de treball. Cal destacar, que en ser programari privatiu, no compte amb una comunitat tan gran produint tutorials, però que, per altra banda, ofereix una documentació oficial molt clara.

FIGURA 5.5: Logo del programa de producció 3D*Maya*.

5.2.2 Programa de producció 3D escollit

Tot i que *Maya* és un programa privatiu amb una llicència, considerablement cara, m'he decantat per aquesta opció a causa de la meva formació prèvia amb ell.

Pel que fa a aspectes com el control de les eines de modelatge, aquests poden ser alterats per tal de ser pràcticament idèntics a *Maya*. No obstant això, aspectes tècnics, com l'elaboració d'un "rigging" adequat per a exportar a *Unity*, resultaven massa costosos pel que fa a temps d'aprenentatge per a aquest projecte.

Capítol 6

Requisits del sistema

6.1 Requisits funcionals

Aquest videojoc es caracteritza pels següents requisits funcionals:

- El generador de nivells sempre haurà de generar un nivell completable. Aquest nivell consistirà en un conjunt de sales interconnectades (d'estructura predeterminada) que podran ser accedides en un determinat ordre i que hauran de ser completades (eliminar tots els enemics) per tal de seguir. Addicionalment, les sales tindran un nivell d'accés numèric determinat, el qual el jugador haurà d'haver superat anteriorment, com a mínim, una sala amb un nombre d'accés immediatament inferior per tal d'accedir-hi.

- El jugador podrà interactuar amb les portes i tele-transportadors per a poder avançar en el joc. Això només ho podrà fer si la sala en la qual es troben ha estat completada.

- El jugador es podrà moure en qualsevol direcció des d'una vista superior. La càmera el seguirà de manera automàtica.

- El jugador podrà apuntar, de manera independent al moviment, en qualsevol direcció des d'una vista superior.

- El jugador disposarà de 3 armes amb configuracions de tret diferents. Podrà canviar entre aquestes armes en qualsevol moment.

- El jugador podrà disparar bales, segons l'arma seleccionada, per tal d'eliminar enemics i completar les sales.

- El jugador podrà recarregar l'arma que tingui seleccionada.

- El jugador podrà utilitzar una habilitat de teletransport que el desplaçarà una determinada distància cap endavant fins a trobar un obstacle.

- El jugador podrà aturar el joc en qualsevol moment, mentre duri la partida, i a través d'aquesta pantalla podrà reprendre la partida, tornar al menú principal o sortir del joc.

- El jugador morirà un cop hagi estat impactat per tantes bales enemigues com determini la vida màxima. Aquest fet el portarà a un menú de derrota on podrà

començar una nova partida, tornar al menú principal o sortir del joc.

- El jugador guanyarà la partida un cop superi una sala final i interaccioni amb el tele-transportador de fi de nivell. Aquest fet el portarà a un menú de victoria on podrà començar una nova partida, tornar al menú principal o sortir del joc.
- El jugador podrà utilitzar tant teclat i ratolí com un "gamepad" compatible amb PC per a realitzar totes les accions.
- Quan el jugador entri en una sala no completada, aquesta s'emplenarà d'un determinat nombre d'enemics.
- Els enemics intentaran localitzar al jugador en la sala que es trobin. Un cop hagin detectat el jugador, intentaran posicionar-se de manera que puguin disparar amb prou exactitud.
- Durant la partida es mostrarà una interfície (HUD) amb la barra de vida del jugador, el nivell d'accés actual, el selector d'armes amb la munició de l'arma equipada i un minimapa.

6.2 Requisits no funcionals

El programa només pot ser executat en mode usuari, i per-tant no calen mesures addicionals d'accés. Tampoc es guarda cap mena de dada ni es disposa de cap sistema de connexió en línia, per tant, no és necessària l'aplicació de mesures per a la protecció de dades.

Pel que fa a l'exportació a diferents plataformes, en utilitzar el motor *Unity*, aquest s'encarregarà de poder fer l'exportació a diferents sistemes, tot i que per l'exportació d'aquest treball només utilitzarem l'opció d'exportació a *Windows* que aquest motor ofereix.

Aquest videojoc ha estat testejat pel que fa a rendiment amb un ordinador de sobretaula amb les següents característiques:

- Sistema Operatiu: Windows 10 Pro N
- Processador: Intel(R) Core(TM) i7-6700k cpu @ 4.00GHz 4.01 GHz
- Targeta gràfica: NVIDIA GeForce GTX 1070
- Memòria instal·lada (RAM): 16 GB

Pel que fa a requisits mínims es considera que aquests serien els següents:

- Sistema Operatiu: Windows 8
- Processador: Intel(R) Core(TM) i5-6500 cpu @ 3.60GHz 3.60 GHz
- Targeta gràfica: NVIDIA GeForce GTX 960
- Memòria instal·lada (RAM): 8 GB

Capítol 7

Estudis i decisions

En aquest capítol descriurem amb detall el programari escollit per al desenvolupament d'aquest projecte, aixi com les diferents llibreries i paquets utilitzats.

7.1 Unity

Unity consisteix en l'eix principal del desenvolupament del projecte i tot el material creat haurà de ser compatible amb aquest motor, el qual ofereix totes les eines necessàries per a desenvolupar videojocs per pràcticament qualsevol plataforma i controls.

7.1.1 Conceptes rellevants

Per tal de poder explicar correctament el funcionament d'aquest motor, és necessari descriure breument un seguit de conceptes que seran imprescindibles per a la comprensió d'aquesta documentació.

Model

És una representació d'un objecte en 3D construït mitjançant punts connectats per cares. En el cas de *Unity*, aquestes cares sempre seran triangulars.

Textura

Consisteix en una imatge que es pot utilitzar per a projectar sobre un model 3D o per a il·lustrar un element 2D. *Unity* permet diferents opcions per tal d'ajustar paràmetres com el seu tipus, dimensions, compressió, mètode de repetició ...

Animació 3D

Consisteix en un conjunt d'interpolacions entre les articulacions dels ossos d'un esquelet. *Unity* permet realitzar ajustos sobre la seva duració, velocitat, repetició...

Shader

És un programa gràfic que determina com la informació sobre il·luminació i textura es combinen per a generar els píxels de l'objecte renderitzat en la pantalla.

Material

Un material consisteix en una configuració particular, per qualsevol objecte que l'utilitzi, que defineix propietats d'un *Shader*.

Scene

Es la unitat mínima en que es divideix un videojoc creat en *Unity*. Es on es col·loquen els objectes, anomentats *GameObjects*, per tal de construir el joc. Un joc pot tenir diverses escenes, les quals s'han de carregar per tal que siguin executades.

GameObject

Consisteix en una col·lecció de components que defineixen les propietats i comportaments d'un objecte instanciable en una escena de *Unity*.

Component

La filosofia principal d'aquest motor és la de la programació mitjançant scripts, els quals s'afegeixen als *GameObjects* i els hi proporcionen les seves característiques. Aquests scripts són el que anomenem components, dels quals els més destacables són els següents:

- *Transform*: Proporciona les propietats de posició, rotació i escala en escena. És el *Component* mínim i indispensable per a cada *GameObject*.

- *Camera*: Crea una càmera virtual amb un seguit de paràmetres de paràmetre per ajustar-ne la visió i resolució, la qual renderitzarà la seva vista de l'escena durant l'execució del joc. Com a mínim ha d'existir una càmera per tal de poder visualitzar el joc durant la seva execució.

- *Canvas*: Crea un marc que es renderitza per sobre la vista de la càmera principal. En aquest marc es poden posicionar *GameObjects* 2D preexistents (botons, textos, imatges...) per així crear interfícies. Aquest marc sempre s'ajusta a la resolució de la càmera, i els elements que el componen mantenen un component *Rect Transform* (modificació del *Transform*) que permet que es puguin adaptar a aquests canvis.

- *Collider*: Proporciona un volum a l'objecte del qual es poden ajustar les dimensions. Aquest s'utilitza per a la detecció de col·lisions amb altres *Colliders*. Pel que fa a la forma del volum existeixen *Colliders* en forma de càpsula, esfera, caixa i malla (aproximació d'una geometria complexa). Els *Colliders* també es poden classificar segons si funció com a barreres físiques o si són únicament de detecció (això s'indica amb la propietat *IsTrigger*).

- *Rigidbody*: Fa que l'objecte sigui afectat pel motor de físiques. Ofereix paràmetres per definir la massa i fricció de l'objecte, i també per a limitar l'efecte de les físiques a uns eixos de posició i rotació concrets de l'objecte que el conté. Aquest component és indispensable juntament amb un *Collider* per a poder detectar col·lisions d'objectes en moviment, tot i que es deshabiliti l'aplicació de físiques activant el "flag" *IsKinematic*.

- *Character Controller*: Fa que un objecte no sigui afectat per físiques, però que sí que es trobi restringit per col·lisions. Ofereix el mètode *Move* per tal de realitzar el moviment i paràmetres per tal d'ajustar conceptes com la seva velocitat i el pendent màxim pel qual es pot desplaçar. Aquest component també afegeix un volum en forma càpsula ajustable, amb el qual realitza els càlculs.
- *Mesh Filter*: Permet la càrrega d'un model 3D no deformable.
- *Mesh Renderer*: Utilitza el model 3D carregat amb el *Mesh Filter* per a renderitzar l'objecte.
- *Skinned Mesh Renderer*: Permet la càrrega d'un model 3D deformable amb el seu esquelet per a renderitzar-lo.
- *Animator*: Permet aplicar a l'objecte les animacions i transicions d'un *Animator Controller*.

MonoBehaviour

És la classe de la qual hereten tots els components de *Unity*. Aquesta proporciona els mètodes que controlen el flux d'execució del programa així com la resposta davant la detecció de col·lisions.

Prefab

Consisteix en un *GameObject* o conjunt d'aquests en estructura jeràrquica, guardat dins del projecte. Els *Prefabs* s'utilitzen per a crear instàncies d'ells dins l'escena del joc o altres *Prefabs*, de manera que modificant el *Prefab*, es puguin alterar totes les instàncies, però no a la inversa (modificar una instància únicament mantindrà aquella instància modificada).

Physics

És una classe que permet accedir a propietats globals de les físiques del motor. També proporciona mètodes de suport per a la detecció de col·lisions com ho serien les diferents variants de *Raycast* els quals són molt útils per a tractar la detecció d'objectes a distàncies específiques.

Layers

És un sistema de noms dels quals cada *GameObject* en tindrà un d'assignat. *Unity* els utilitza principalment, per a limitar el motor de físiques (avaluar o no col·lisions de *GameObjects* segons les *Layers* que tinguin assignades) i també per a limitar el renderitzada d'una càmera (renderitzar únicament els *GameObjects* de les *Layers* marcades). El seu ús, però, no es limita als anteriorment mencionats, sinó que permet al programador usar aquest sistema per a qualsevol propòsit d'identificació entre *GameObjects*.

Tags

De la mateixa manera que les *Layers*, consisteixen en un sistema de noms dels quals cada *GameObject* en tindrà un d'assignat. La principal diferència amb les *Layers* és

que aquest sistema és per a ús específic del programador i no s'usa per a configuracions del motor de físiques i gràfics, tot i que sí que es fa servir per tasques com poder determinar quin *GameObject* és la càmera principal (amb el tag *MainCamera*) o el jugador (amb el tag *Player*).

Scriptable Object

És una classe de la qual heretaran classes que es vulguin fer servir per a crear instàncies en el nostre projecte que serveixin per a guardar i contenir unes dades determinades. Aquestes instàncies se solen fer servir com a arxius de configuració per a altres sistemes.

Avatar

En exportar un model juntament amb el seu esquelet, *Unity* permet crear un arxiu (*Avatar*) que contindrà la informació d'aquest esquelet per a poder ser utilitzat per a interpolar-ne les animacions.

Avatar Mask

Aquest arxiu, permet utilitzar un *Avatar* creat anteriorment, per tal de poder seleccionarne un seguit d'ossos i excloure'n d'altres. Això es fa servir per limitar l'aplicació d'una animació a uns determinats ossos.

7.1.2 Entorn visual

Per tal d'agilitzar i fer mes intuïtiu el desenvolupament, *Unity* disposa d'un editor visual amb diverses eines en forma de finestres de l'aplicació. D'aquestes eines en podem destacar les següents:

- *Scene*: Ens permet moure lliurement per l'escena actual de forma similar a un programa d'edició 3D. Aquesta finestra s'utilitza per a construir el joc col·locant en l'escena els diferents *GameObjects* que la constitueixen.

- *Hierarchy*: Mostra una jerarquia de la construcció de l'escena actual. Aquesta interfície permet col·locar els diferents *GameObjects* dins d'altres, creant així una relació de parentiu, de manera que moure l'element pare també provocarà el moviment dels elements fills.

- *Inspector*: Mostra i permet modificar les propietats configurables d'un element seleccionat en l'escena o el projecte. Pel que als *GameObjects*, en mostrarà tots els seus components amb les seves respectives propietats.

- *Project*: És una finestra dedicada a poder visualitzar tots els arxius en l'estructura de fitxers del projecte.

- *Animator*: Ofereix la possibilitat d'organitzar les animacions de l'*Animator Controller*, generalment d'un personatge. Aixo es fa en forma de maquina d'estats amb les seves respectives condicions de transició. Aquesta eina també permet configurar les interpolacions entre les animacions i limitar la seva influencia en el personatge a traves d'*Avatar Masks* i capes amb pesos d'influència.

- *Game*: Mostra com es veurà el joc quan s'estigui executant i consisteix en el mètode principal per a realitzar proves d'execució. Aquesta finestra també ofereix la possibilitat de mostrar indicadors, que no es veuran en el producte final, com ara "colliders", i també permet mostrar el rendiment del joc i ajustar la visualització a qualsevol resolució de pantalla.

- *Console*: Juntament amb la finestra de *Game*, ens serveix per a realitzar les proves d'execució. Aquesta finestra de text mostra, durant l'execució del joc, els diferents errors, missatges de depuració o excepcions que es puguin arribar a produir.

7.1.3 Paquets i recursos addicionals instal·lats

Input System

És un paquet oficial que serveix com a substitut de l'*Input Manager* donat per defecte. A grans trets, ofereix una interfície molt intuïtiva que permet crear esquemes per a diferents mètodes de control (teclat i ratolí, controlador...) per al joc que es vol desenvolupar. Això ho fa associant accions (atacar, saltar, moure...), a diferents "inputs". Mitjançant el component *Player Input*, proporcionat per aquest paquet, es pot associar aquestes accions a un *GameObject* en concret, com per exemple al personatge que podrà controlar el jugador.

Cinemachine

Consisteix en un paquet oficial, el qual posa a la disposició del programador, d'un seguit de components (a destacar el *Cinemachine Brain*) que permeten dotar comportaments a una càmera (per exemple de seguiment d'un personatge) a través de la modificació de paràmetres.

AI Navigation

Unity de base ja incorpora un mòdul per a la creació de *NavMesh* o malles per on personatges controlats per la IA, *Nav Mesh Agents*, es poden moure utilitzant un algoritme de "path finding". Aquest paquet oficial de *Unity*, inclou components que fan d'aquesta generació un procés més modular i que també permeten realitzar-la en temps d'execució, de manera que resulta imprescindible per usar el *NavMesh* en nivells creats de manera procedimental.

Universal RP

És un dels paquets oficials més comunament usats, ja que es pot crear un projecte directament amb aquesta opció. Aquest paquet dona múltiples eines per tal d'agilitzar la creació artística pel que fa al renderitzat. Aquest paquet també integra *Shader Graph* el qual consisteix en una interfície visual (en forma de finestra) que permet crear "shaders" sense haver d'escriure codi.

Visual Effect Graph

Aquest paquet oficial de *Unity*, ofereix eines per crear, de manera rapida i intuïtiva, efectes visuals procedimentals, com a sistemes de partícules.

Animation Rigging

Es un paquet oficial de *Unity* que ofereix un seguit de components (com el *Bone Renderer* i el *Rig Builder*) per tal de crear animacions completament procedimentals o per afegir dinamisme a animacions d'altre manera completament estàtiques.

TextMeshPro

Aquest paquet oficial ens permet crear text, per a les nostres interfícies, amb mes opcions de format que el text bàsic de *Unity*, com ara la possibilitat d'escollir una font externa.

Behaviour Tree (TheKiwiCoder)

És un paquet completament gratuït i desenvolupat per *TheKiwiCoder* [16] que ofereix una interfície visual per tal de crear "behaviour trees" per a la IA. Això ens permet crear una IA molt modular en forma de diferents nodes (scripts amb comportaments senzills), els quals els podem combinar per obtenir comportaments complexos i situacionals, els quals formaran les branques del nostre arbre. Per tal d'afegir aquest comportament als nostres personatges, aquest paquet posa a la nostra disposició el component *Behaviour Tree Runner*.

7.1.4 AStar

És una classe adaptada a *C#* per *Amitkapadi* d'una implementació de l'algoritme de *A** feta per *Andrea Giammarchi*. Ofereix la possibilitat de calcular camins mínims, en forma de llista de vectors 2D (posicions d'una matriu), proporcionant una matriu booleana (per indicar les caselles navegables), una posició d'inici, una posició de fi, i el mode de calcular el camí (*Manhattan, Diagonal, DiagonalFree, Euclidean, EuclideanFree*). És essencial per a l'algoritme de generació del pis, ja que permet assegurar que les habitacions poden ser visitades en l'ordre indicat. Pel que fa al mode de calcular el camí utilitzarem *Manhattan*, perquè les connexions entre sales no són diagonals i aquest mode calcula camins mínims sense fer moviments d'aquest tipus.

7.1.5 Llibreries utilitzades

Les llibreries utilitzades en aquest projecte son les següents:

- *UnityEngine*: Afegeix les funcionalitats més bàsiques i essencials que ens ofereix *Unity*.
- *UnityEngine.UI*: Permet accedir i crear elements de la interfície gràfica de *Unity*.
- *UnityEngine.EventSystems*: Permet accedir al sistema d'"events" de la interfície gràfica de *Unity*, l'*EventSystem*.
- *UnityEngine.Rendering*: Permet accedir a les funcions i constants del sistema de renderitzat de *Unity*.
- *UnityEngine.Events*: Permet la creació i us d'*UnityActions*, les quals permeten dotar als scripts de *Unity* de funcionalitats dinàmiques.

- *UnityEngine.Pool*: Afegeix un contenidor de dades optimitzat per a la reutilització d'objectes instanciables en una escena de *Unity* (*ObjectPool*).
- *UnityEngine.InputSystem*: Permet accedir a les funcionalitat del *InputSystem* de *Unity*.
- *UnityEngine.Random*: Afegeix funcionalitats per a generar nombres aleatoris a *Unity*.
- *UnityEngine.AI*: Afegeix funcionalitats per a poder accedir als *Nav Mesh Agents* de *Unity*.
- *UnityEngine.SceneManagement*: Afegeix les funcions per a realitzar la carrega d'escenes de *Unity*.
- *System*: Afegeix les classes més utilitzades i bàsiques de *C#*.
- *System.Collections*: Afegeix contenidors de dades com llistes, cues, taules i diccionaris.
- *System.Collections.Generic*: Conte interfícies i classes que defineixen col·leccions genèriques.
- *System.Linq*: Proporciona classes i interfícies que admeten consultes que utilitzen *Language - Integrated Query (LINQ)*.
- *Unity.AI.Navigation*: Afegeix les funcionalitats del paquet *AI Navigation* per tal de poder crear el *NavMesh* mitjançant codi.
- *TMPro*: Permet utilitzar les funcions i elements del paquet *TextMeshPro*.
- *TheKiwiCoder*: Permet utilitzar les funcions i elements del paquet *Behaviour Tree (TheKiwiCoder)*.

7.2 Visual Studio

Unity no disposa d'un editor de codi propi, per tant escollir un entorn per a escriure el codi és imprescindible per tal de desenvolupar els "scripts" en *C#* del projecte. Aquest entorn té una molt bona integració amb *Unity*, oferint ajuda de codi per a funcions específiques de l'API de *Unity* i també ofereix la possibilitat de depurar mitjançant "breakpoints".

7.3 Maya

Maya principalment, és conegut com un programa per a la producció artística de models 3D, de fet, ha estat utilitzat per a fer els models de personatges que és s'utilitzen en aquest projecte. Tot i això, és important saber que existeix una rellevant complexitat tècnica (sobretot pel que fa a la creació d'animacions), per tal d'aconseguir que aquest material pugui ser usat en un motor de videojocs com *Unity*. És per això, que aquest mateix programa ofereix una sèrie d'eines per a fer-ho possible.

Capítol 8

Anàlisi i disseny del sistema

Aquest capítol té per objectiu descriure, de manera conceptual, els diferents aspectes que conformen el videojoc que desenvoluparem.

8.1 Descripció general

A grans trets, aquest videojoc, s'emmarca dins dels "rogue-likes" d'acció de vista superior. Concretament, el joc consistirà a navegar per un pis compost per sales, les quals el jugador haurà de completar, eliminant tots els enemics, per tal d'accedir-ne a un altre, fins a així arribar al final o morir i haver de repetir el procés en una nova partida. El combat se centra en l'intercanvi de bales entre el jugador i els enemics, seguint com a inspiració els clàssics "Bullet hell" com *Touhou* [17].

FIGURA 8.1: Captura de pantalla del videojoc *Touhou*.

8.2 Atributs dels personatges i armes

Tant el personatge que controla el jugador com els enemics comptaran amb un seguit d'atributs que permeten una manera més fàcil i directa per a poder-los balancejar, així com, en un treball futur, implementar millores d'atributs en la partida. Cal dir que aquests atributs es trobaran a diferents nivells els quals a continuació descriurem:

- Atributs base del personatge:
  - Vida: Punts de mal que pot rebre un personatge. En arribar a 0, aquest morirà.

  - Velocitat: Desplaçament del personatge per l'entorn.
  - Selecció d'armes: Sempre tindrà una arma seleccionada d'entre 3 possibles armes. Aquestes tindran configuracions que podran canviar per complet la manera de disparar del personatge.

- Atributs d'arma:
  - Temps de reutilització: Temps d'espera entre ús de l'arma (després de l'aturada de foc).
  - "Flag" de foc automàtic: Determina si per disparar s'ha de donar una comanda cada vegada.
  - "Flag" d'aturar el foc: Determina si es pot cancel·lar el llançament d'una rafega de projectils.
  - Cadència de foc: Determina el temps entre projectils d'una rafega.
  - Nombre de projectils: Determina quants de projectils es dispararan en una rafega.
  - Munició de cartutx: Munició que es pot utilitzar sense recarregar. Aquesta pot ser infinita i fer servir directament la munició addicional.
  - Munició de magatzem: Munició que s'usa per recarregar l'arma. Aquesta pot ser infinita.
  - Temps de recàrrega: Temps necessari per a poder carregar l'arma donada un "input" de recàrrega.
  - "Flag" de recàrrega automàtica: Determina si en un "input" de disparar, sense munició al magatzem, es recarregarà l'arma.
  - Patró de bales: Patró instanciat per l'arma en un tret.

- Atributs de patró de bales:
  - Components: Cada patró està conformat alhora de diversos subpatrons amb els següents atributs:
    * Flag de dispersió aleatòria: Determina si les bales s'instanciaran de manera aleatòria en l'angle d'instanciació.
    * Velocitat de bala: Desplaçament de la bala per l'entorn.
    * Nombre de bales: Nombre de bales a instanciar en l'angle d'instanciació.
    * Angle d'instanciació: Angle en el qual es repartiran les bales a instanciar.
    * Direcció d'angle d'instanciació: Direcció on apunta el centre de l'angle d'instanciació.

    * Distància d'instanciació respecte al centre: Distància del centre que tindran les bales instanciades.
    * Retard d'instanciació: Temps abans d'instanciar les bales.
    * Bala: Bala que s'instanciarà.

- Atributs de bala:
  - Dany: Punts de mal que rebran els enemics.

8.3 Disseny del funcionament

En aquest joc, el jugador controlarà un robot i haurà d'arribar a la sala final d'un nivell laberíntic conformat per sales distribuïdes de manera aleatòria.

Les portes de les sales es podran obrir si el jugador compta amb un nivell d'accés adequat (es podrà accedir aquelles sales que tinguin un nivell igual o inferior que el nivell d'accés del jugador més un). A cada nova sala que s'entri, apareixeran enemics que buscaran al jugador i intentaran matar-lo, fent que així perdi la partida. El jugador haurà de derrotar a tots els enemics per a poder sortir de la sala i continuar avançant.

La partida acabarà quan el jugador interactuï amb el tele-transportador de nivell que contindran les sales finals i podran ser interactuats en completar-les.

8.3.1 Blocs de construcció

Aquests blocs conformen els límits de les sales, i per tant del joc. No poden ser interactuats i fan de barrera per a qualsevol element del joc.

Terra

Bloc que delimita la superfície transitable per a qualsevol personatge.

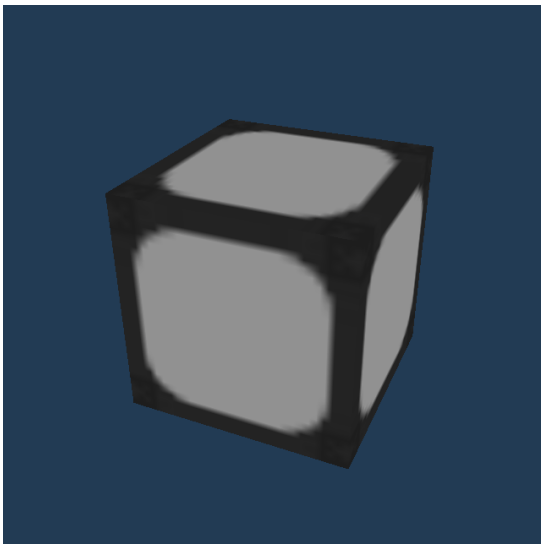

FIGURA 8.2: Captura de pantalla del terra.

Paret

Barrera indestructible que delimita una sala.

FIGURA 8.3: Captura de pantalla de la paret.

8.3.2 Blocs d'accés

Consisteixen en blocs amb els quals el jugador pot interaccionar per tal de continuar avançant en el nivell. Per a poder ser interaccionats serà imprescindible que el jugador hagi superat tant la sala en què es troba i, com a mínim, una sala de nivell immediatament inferior.

Porta

És la via d'accés a la sala annexionada, de la qual en mostrarà el nivell d'accés. Es tancarà automàticament, canviant d'aparença, mentre el jugador es trobi completant la sala i només tornarà a la seva aparença normal un cop aquesta sigui completada.

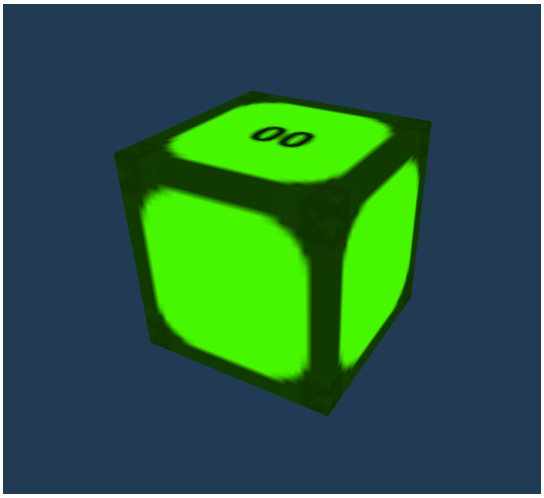

FIGURA 8.4: Captura de pantalla de la porta.

Tele-transportador de vincle

És una via d'accés a una sala isolada de la resta de l'estructura del pis. No serà visible mentre el jugador es trobi completant la sala.

FIGURA 8.5: Captura de pantalla del tele-transportador de vincle.

Tele-transportador de nivell

És l'objectiu final del jugador per tal de guanyar el joc. No serà visible mentre el jugador es trobi completant la sala.

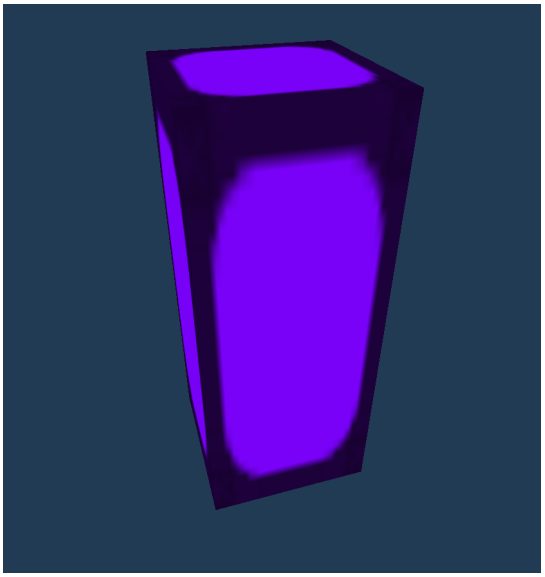

FIGURA 8.6: Captura de pantalla del tele-transportador de nivell.

8.3.3 Enemics

Son els elements del joc que intentaran fer que el jugador perdi la partida. De la mateixa manera que el jugador, atacaran mitjançant dispars, els quals variaran segons l'enemic.

Clon

Són una còpia inferior del personatge principal controlat pel jugador. Aquests enemics, quan apareixen, naveguen de manera aleatòria per la sala fins que el jugador es trobi en el seu camp de visió. Havent-lo detectat, tindran dos comportaments. Si el jugador es troba massa lluny per a disparar-lo amb prou precisió, el perseguiran. Altrament, l'aniran rodejant mentre el disparen, sempre i quant, tinguin una línia de tir neta (sense obstacles).

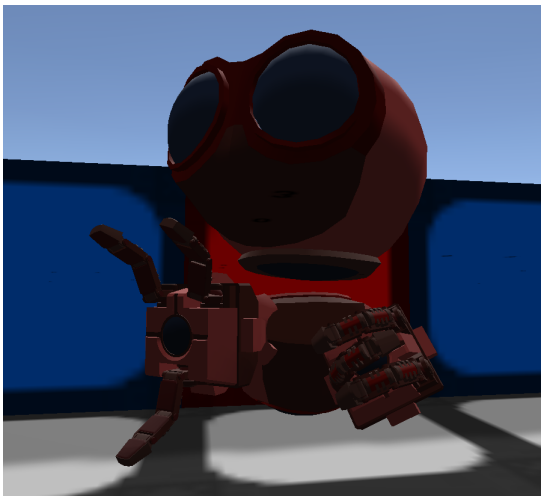

FIGURA 8.7: Captura de pantalla del enemic *Clon*.

HEX-p0p

Aquest té un comportament pràcticament idèntic al *Clon*, però es caracteritza per ser més lent, més resistent i tenir una forma de tret diferent.

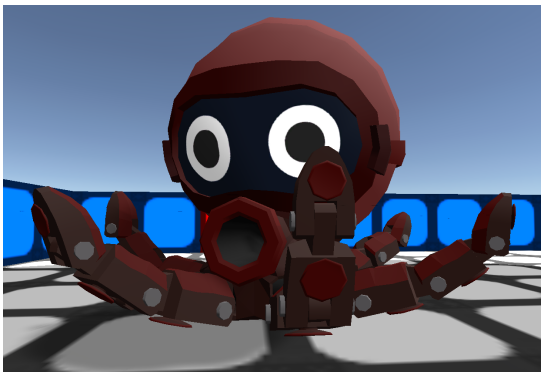

FIGURA 8.8: Captura de pantalla del enemic *HEX-p0p*.

8.3.4 G3-INF

És el robot que controla el jugador i amb el qual haurà d'arribar al final del joc. Aquest robot permetrà al jugador les següents accions:

- Desplaçar-se en qualsevol direcció d'una vista superior.
- Apuntar en qualsevol direcció des de la vista superior.
- Alternar entre 3 armes diferents.
- Recarregar l'arma actual.
- Tele-transportar-se cap endavant una distància determinada (sense travessar obstacles).
- Interactuar amb els blocs d'accés del nivell.

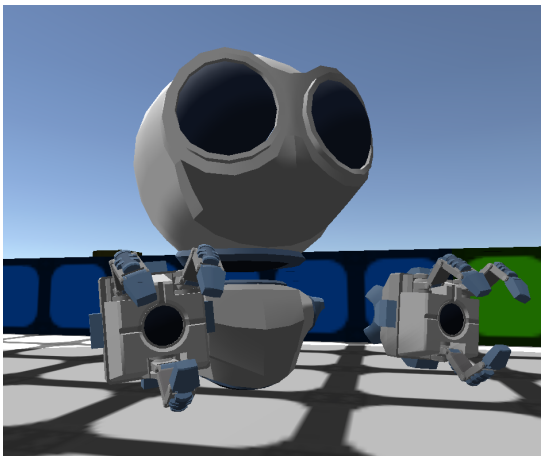

FIGURA 8.9: Captura de pantalla del personatge principal *G3-INF*.

8.3.5 Interfícies d'usuari

Menú principal

Es el primer menú que es mostra al iniciar el joc. Aquest permet seleccionar una de les següents opcions:

- Començar la partida: Inicia la construcció del nivell, indicat amb una pantalla de carrega. Tant bon punt aquesta finalitza la partida comença.
- Sortir del joc: Finalitza l'aplicació.

Menú de pausa

És el menú que es mostrarà quan, en qualsevol moment de la partida, el jugador premi el botó de pausa. El jugador podrà sortir d'aquest menú sense seleccionar cap opció tornant a prémer el botó de pausa. Aquest menú compta amb les següents opcions:

- Tornar-ho a intentar: Torna a iniciar la construcció del nivell, indicat amb una pantalla de càrrega, per tal de començar una nova partida.

- Tornar al menú principal: Finalitza la partida tornant al menú principal.
- Sortir del joc: Finalitza l'aplicació.

Menú de victòria

És el menú que es mostrarà després de que el jugador interactuí amb un tele-transportador de nivell, marcant el fi de la partida com a victòria.

- Començar la partida: Inicia la construcció del nivell, indicat amb una pantalla de carrega. Tant bon punt aquesta finalitza la partida comença.
- Tornar al menú principal: Finalitza la partida tornant al menú principal.
- Sortir del joc: Finalitza l'aplicació.

Menú de derrota

És el menú que es mostrarà després que els punts de vida del jugador arribin a 0, marcant el fi de la partida com a derrota.

- Començar la partida: Inicia la construcció del nivell, indicat amb una pantalla de càrrega. Tan bon punt aquesta finalitza la partida comença.
- Tornar al menú principal: Finalitza la partida tornant al menú principal.
- Sortir del joc: Finalitza l'aplicació.

Interfície de la partida (HUD)

Aquesta interfícies una capa que s'afegeix a la pantalla de joc durant la partida i que mostra informació important del jugador. Aquests elements són els següents:

- Barra de vida: Mostrar la vida actual del jugador tant en format de barra de vida actual com en format numèric sobre el seu total. Quan la vida actual arribi a 0 el jugador perdrà la partida
- Nivell d'accés: Mostra el nivell d'accés actual sobre el nivell d'accés màxim del pis. El nivell d'accés actual indica que es podrà interactuar amb totes les portes i tele-transportador de com a màxim un nivell més.
- Selector d'armes: Mitjançant un sistema de colors, indica quina arma es té seleccionat i en mostra la seva munició de cartutx (MAG) i de magatzem (STO). La munició de cartutx (MAG) és la que es podrà disparar sense recarregar i la munició de magatzem (STO) indica quants trets es poden recarregar.
- Minimapa: Mostra una àrea més gran del voltant del jugador de manera simplificada.

8.4 Identificació dels actors

Aquest videojoc identifica dos tipus d'actors diferents, el jugador humà i la IA. Això és a causa del fet que, tant el jugador humà com la IA controlaran un personatge cadascú, havent-se d'eliminar entre ells. No obstant això, el jugador humà serà l'únic que tindrà la capacitat de navegar entre els diferents menús del videojoc.

8.5 Casos d'ús

En aquest apartat veurem en detall els casos d'ús d'aquest videojoc agrupats en les seccions en què aquest es divideix. Aquestes seccions són el menú principal, la partida, el menú de pausa i el menú de fi de partida.

8.5.1 Diagrames

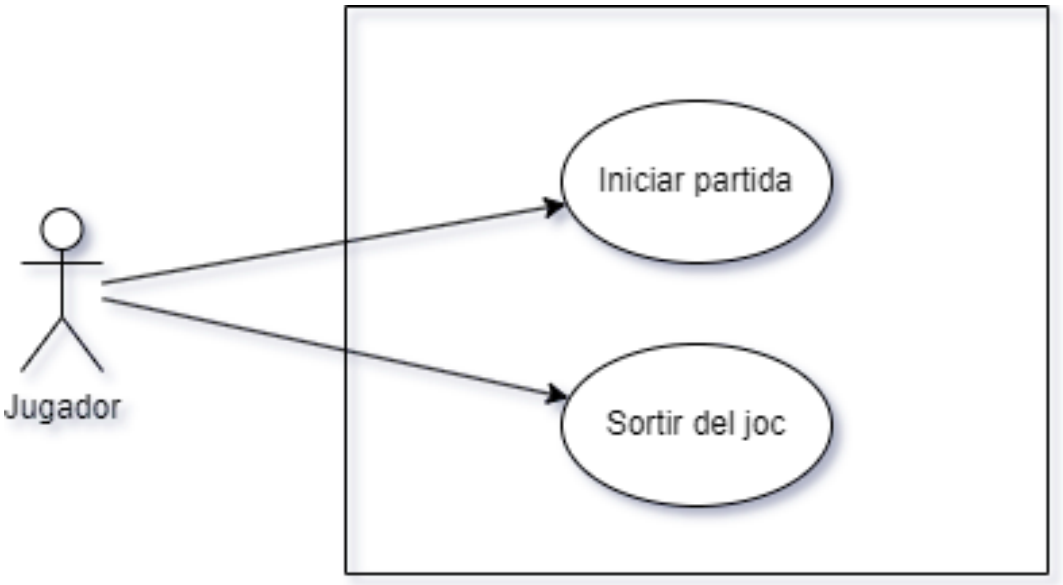

FIGURA 8.10: Diagrama de casos d'ús del menú principal.

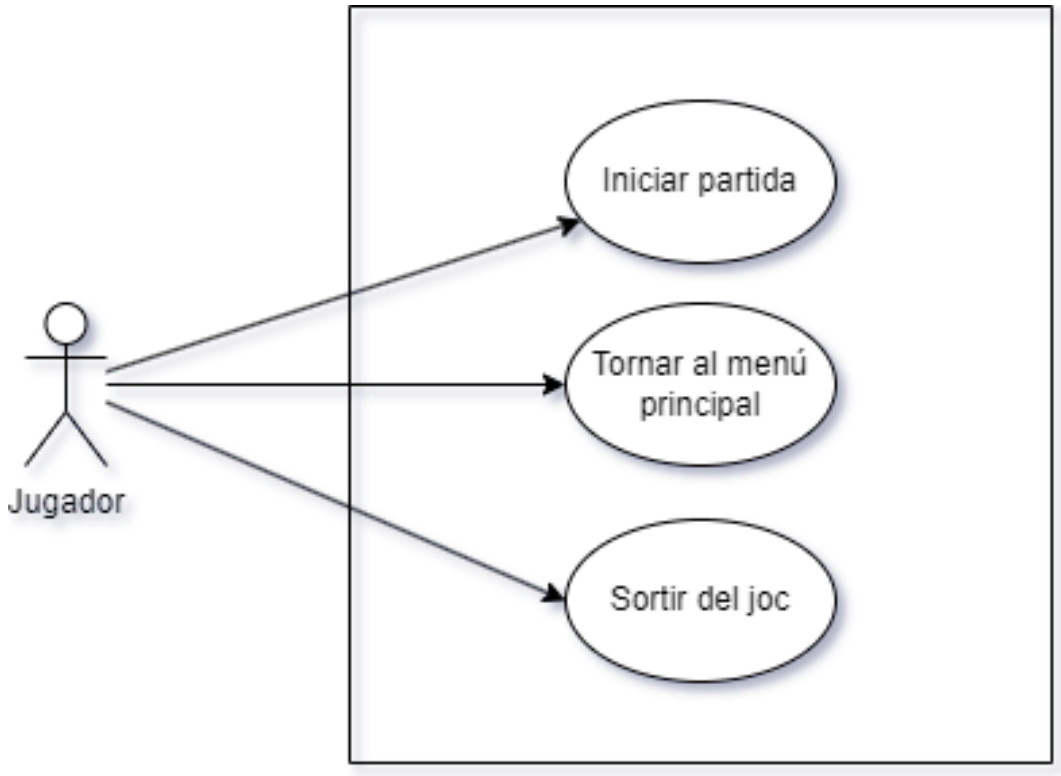

FIGURA 8.11: Diagrama de casos d'ús del menú de fi de partida.

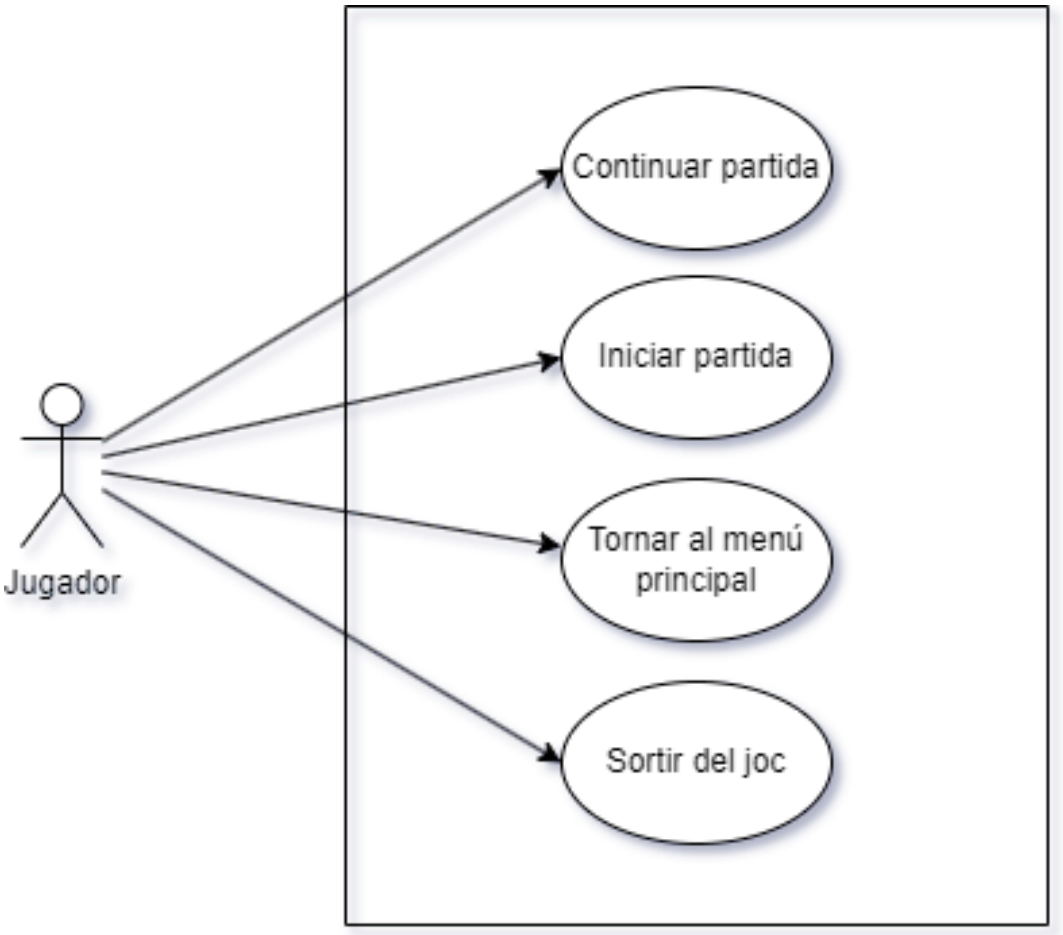

FIGURA 8.12: Diagrama de casos d'ús del menú de pausa.

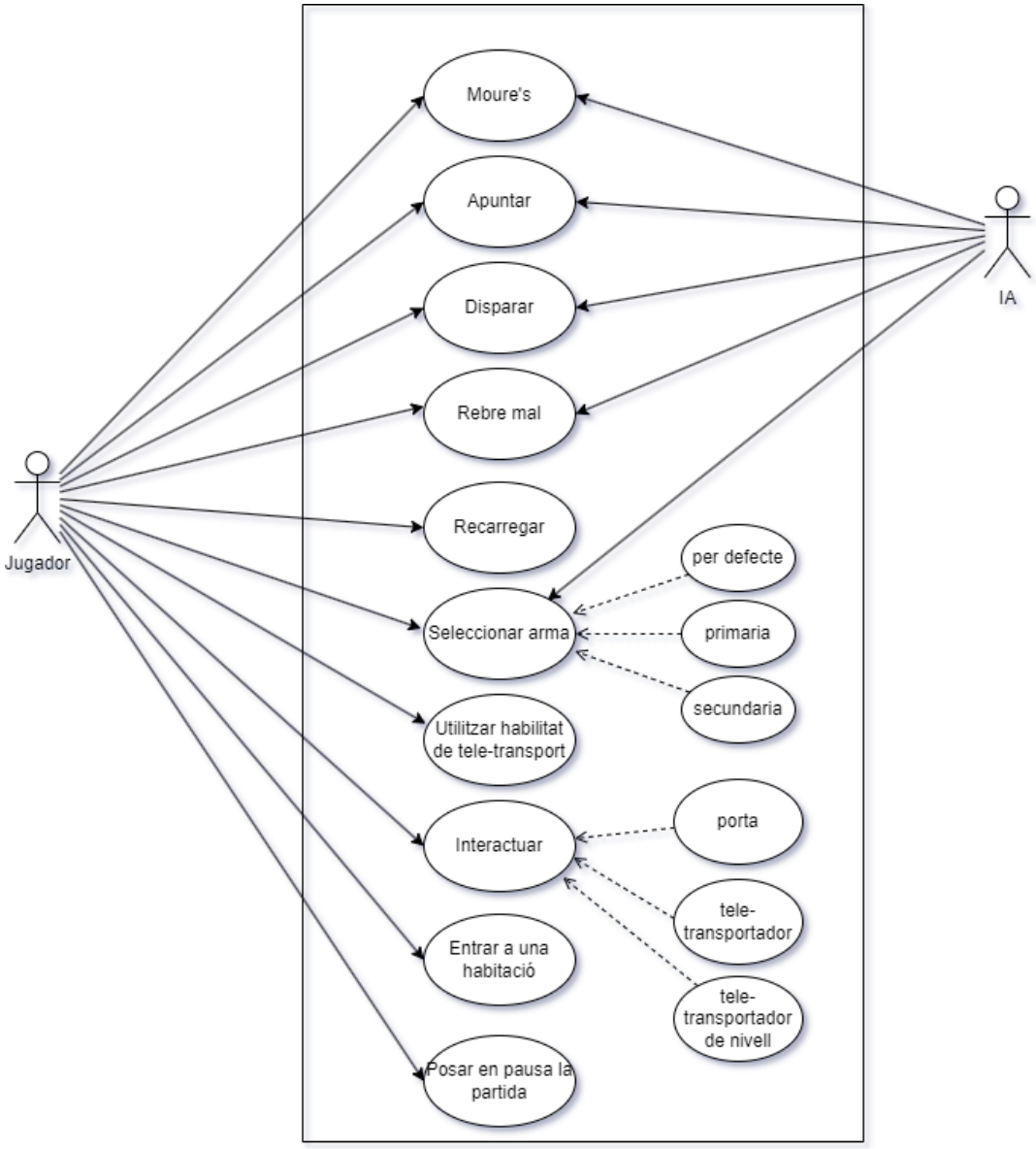

FIGURA 8.13: Diagrama de casos d'ús de la partida.

8.5.2 Fitxes

TAULA 8.1: Fitxa de cas d'ús: Iniciar Partida

Descripció:	Genera el nivell i prepara tots els elements necessaris per a començar la partida.
Actor:	Jugador.
Precondició:	Estar en un dels menús.
Flux Principal:	1. Escollir l'opció d'iniciar partida. 2. Mostrar pantalla de carrega. 3. Carregar l'escena de joc. 4. Generar l'estructura lògica de sales. 5. Per cada sala: 5.1. Generar interior. 5.2. Instanciar blocs. 6. Instanciar el jugador a la sala inicial. 7. Amagar pantalla de carrega.
Flux Alternatiu:	Cap.
Postcondició:	Comença la partida.

TAULA 8.2: Fitxa de cas d'ús: Sortir del joc

Descripció:	Finalitza l'execució del joc.
Actor:	Jugador.
Precondició:	Estar en un dels menús.
Flux Principal:	1. Escollir l'opció de sortir del joc. 2. Finalitza l'execució del joc.
Flux Alternatiu:	Cap.
Postcondició:	Joc finalitzat.

TAULA 8.3: Fitxa de cas d'ús: Tornar al menú principal

Descripció:	Torna al menú principal del joc.
Actor:	Jugador.
Precondició:	Estar en el menú de pausa o de fi de partida.
Flux Principal:	1. Escollir l'opció de tornar al menú principal. 2. Torna al menú principal.
Flux Alternatiu:	Cap.
Postcondició:	S'ha tornat al menú principal.

TAULA 8.4: Fitxa de cas d'ús: Continuar partida

Descripció:	Reprèn la partida en el punt on havia estat pausada.
Actor:	Jugador.
Precondició:	Estar en el menú de pausa
Flux Principal:	1. Escollir l'opció de continuar partida. 2. Tenca el menú de pausa. 3. Reprèn la partida.
Flux Alternatiu:	Cap.
Postcondició:	S'ha reprès la partida.

TAULA 8.5: Fitxa de cas d'ús: Mourse's

Descripció:	Desplaça el jugador.
Actor:	Jugador.
Precondició:	Estar en la partida viu i sense el moviment restringit.
Flux Principal:	1. Prémer un o més botons de moviment. 2. Desplaça el personatge cap a la direcció indicada.
Flux Alternatiu:	Cap.
Postcondició:	S'ha realitzat el moviment.

TAULA 8.6: Fitxa de cas d'ús: Apuntar

Descripció:	Canvia la direcció en la que apunta el personatge.
Actor:	Jugador i IA.
Precondició:	Estar en la partida viu i sense l'apuntat restringit.
Flux Principal:	1. Indicar la direcció d'apuntat. 2. Fer apuntar el personatge cap a la direcció indicada.
Flux Alternatiu:	Cap.
Postcondició:	S'ha canviat la direcció en la que apunta el personatge.

TAULA 8.7: Fitxa de cas d'ús: Disparar

Descripció:	Realitza un disparar amb l'arma seleccionada, si es possible.
Actor:	Jugador i IA.
Precondició:	Estar en la partida viu i sense l'atac restringit.
Flux Principal:	1. Prémer el boto de disparar. 2. Si el personatge esta controlar per el jugador: 2.1. Si l'arma seleccionada té munició: 2.1.1. Cancel·la el procés de l'arma seleccionada en execució. 2.1.2. Si l'arma seleccionada es l'arma primaria: 2.1.2.1. Mentre es mantingui premut el botó, carregar. 2.1.2.2. Realitzar una rafega de dispars únicament si la carrega ha finalitzat, mentre quedi munició i es mantingui premut el boto. 2.1.3. Si l'arma seleccionada es l'arma secundaria: 2.1.3.1. Mentre es mantingui premut el botó, carregar. 2.1.3.2. Disparar únicament si la carrega ha finalitzat i s'ha deixat de prémer el boto. 2.1.4. Altrament (arma per defecte): 2.1.4.1. Mentre es mantingui premut el botó, disparar cada cert interval de temps mentre quedi munició. 2.2. Altrament 2.2.1. Recarregar. 3. Altrament 3.1. Realitzar un dispar sense restriccions.
Flux Alternatiu:	Cap.
Postcondició:	Si ha estat possible, s'ha realitza un disparar amb l'arma seleccionada.

TAULA 8.8: Fitxa de cas d'ús: Rebre mal

Descripció:	Es resta una quantitat determinada de vida al personatge que ocasiona la seva mort en arribar a 0.
Actor:	Jugador i IA.
Precondició:	Estar en la partida viu.
Flux Principal:	1. El personatge ha estat impactat per una bala. 2. Es sostreu de la vida del personatge la quantitat de mal que infligeix la bala. 3. Si la vida ha arribat a 0: 3.1. Morir. 4. Si el personatge esta controlat pel jugador: 4.1. Finalitzar la partida com a derrota. 5. Altrament, si l'enemic era l'últim de l'habitació: 5.1. Desbloquejar les portes de l'habitació.
Flux Alternatiu:	Cap.
Postcondició:	El personatge ha rebut mal i mort si era el cas.

TAULA 8.9: Fitxa de cas d'ús: Recarregar

Descripció:	Recarrega l'arma seleccionada, si és possible.
Actor:	Jugador.
Precondició:	Estar en la partida viu i sense recarregar restringit.
Flux Principal:	1. Prémer el boto de disparar. 2. Si l'arma pot ser recarregada (arma per defecte i secundària) i la munició carregada en l'arma no es troba en el seu màxim: 2.1. Cancel·la el procés de l'arma seleccionada en execució. 2.2. Esperar el temps de recarrega. 2.3. Sostreure munició de recàrrega fins a posar la munició carregada en l'arma igual al seu màxim o fins que no quedi prou munició per a la recàrrega.
Flux Alternatiu:	Cap.
Postcondició:	Si ha estat possible, s'ha recarregat l'arma seleccionada.

TAULA 8.10: Fitxa de cas d'ús: Seleccionar arma

Descripció:	Selecciona l'arma indicada.
Actor:	Jugador i IA.
Precondició:	Estar en la partida viu i sense canvi d'arma restringit.
Flux Principal:	1. Prémer un dels botons de canviar d'arma. 2. Cancel·la el procés de l'arma seleccionada en execució. 3. Equipar l'arma indicada (per defecte, primària o secundaria)
Flux Alternatiu:	Cap.
Postcondició:	S'ha seleccionat l'arma indicada.

TAULA 8.11: Fitxa de cas d'ús: Utilitzar habilitat de tele-transport

Descripció:	Procés de càrrega, on el jugador es pot moure i apuntar, en el que es va incrementant un rang davant del personatge marcat amb un indicador (no travessa obstacles). El jugador pot deixar de prémer el botó en qualsevol moment per tal d'aparèixer en el lloc de l'indicador.
Actor:	Jugador.
Precondició:	Estar en la partida viu i sense l'ús d'habilitats restringit.
Flux Principal:	1. Mantenir premut el botó de l'habilitat. 2. Cancel·la el procés de l'arma seleccionada en execució. 3. Fer aparèixer l'indicador de telentransport. 4. Mentre es mantingui premut el botó: 4.1. Anar Incrementant la distancia de teletransport fins a arribar al màxim. 4.2. Posar l'indicador en la distància de teletransport en la direcció que mira el jugador sempre que no travessi una paret. 5. Fer desaparèixer el personatge i l'indicador. 6. Posar el personatge a la posició de l'indicador. 7. Fer aparèixer el personatge.
Flux Alternatiu:	Cap.
Postcondició:	El jugador s'ha tele-transportat al lloc indicat.

TAULA 8.12: Fitxa de cas d'ús: Interactuar

Descripció:	El jugador interactua amb un objecte fent que aquest faci unes accions.
Actor:	Jugador.
Precondició:	Estar en la partida viu i sense la interacció restringida.
Flux Principal:	1. Prémer el boto d'interaccionar. 2. Si l'objecte que envolta el personatge o el que es troba immediatament davant seu és interactuable: 2.1. Cancel·la el procés de l'arma seleccionada en execució. 2.2. Si l'objecte interactuat és una porta i el nivell d'accés del jugador més un és igual o superior al nivell d'accés indicat per la porta: 2.2.1. S'obre la porta i la porta adjacent. 2.3. Altrament si l'objecte interactuat és una tele-transportador i el nivell d'accés del jugador més un és igual o superior al nivell d'accés indicat pel tele-transportador: 2.3.1. Fer desaparèixer el personatge 2.3.2. Posar el personatge a la posició del tele-transportador vinculat amb el tele-transportador interactuat. 2.3.3. Fer aparèixer el personatge. 2.4. Altrament si l'objecte interactuat és una tele-transportador de nivell: 2.4.1. Fer desaparèixer el personatge 2.4.2. Finalitzar la partida com a victòria.
Flux Alternatiu:	Cap.
Postcondició:	L'objecte interactuat ha realitzat les seves accions.

TAULA 8.13: Fitxa de cas d'ús: Entrar a una habitació

Descripció:	El jugador entra en una sala i si aquesta no es trobava superada apareixen els enemics i el jugador queda tancat a l'habitació.
Actor:	Jugador.
Precondició:	Estar en la partida viu.
Flux Principal:	1. Entrar a l'habitació. 2. Si l'habitació no ha estat superada: 2.1. Bloquejar les portes de l'habitació 2.2. Fer apareixre els enemics.
Flux Alternatiu:	Cap.
Postcondició:	El jugador ha entrat a l'habitació i si era el cas han aparegut els enemics i ha quedat tancat dins.

TAULA 8.14: Fitxa de cas d'ús: Posar en pausa la partida

Descripció:	Es pausa la partida i mostra el menú de pausa.
Actor:	Jugador.
Precondició:	Estar en la partida viu.
Flux Principal:	1. Prémer el boto de pausa. 2. Posar en pausa la partida en el punt actual. 3. Mostrar el menú de pausa.
Flux Alternatiu:	Cap.
Postcondició:	S'ha pausat la partida mostrant el menú de pausa.

8.6 Classes i interfícies

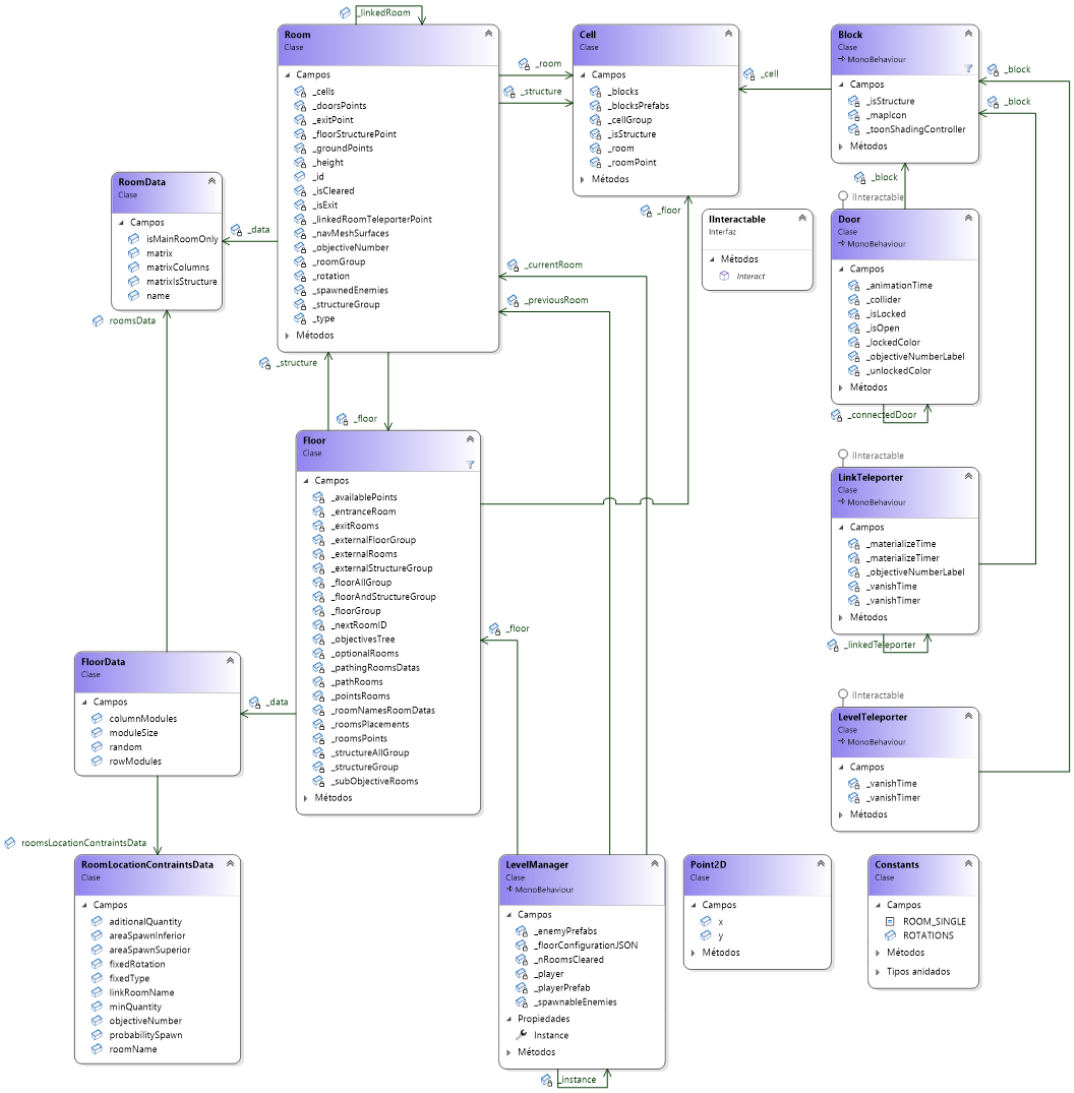

FIGURA 8.14: Diagrama de classes centrat en la generació del pis.

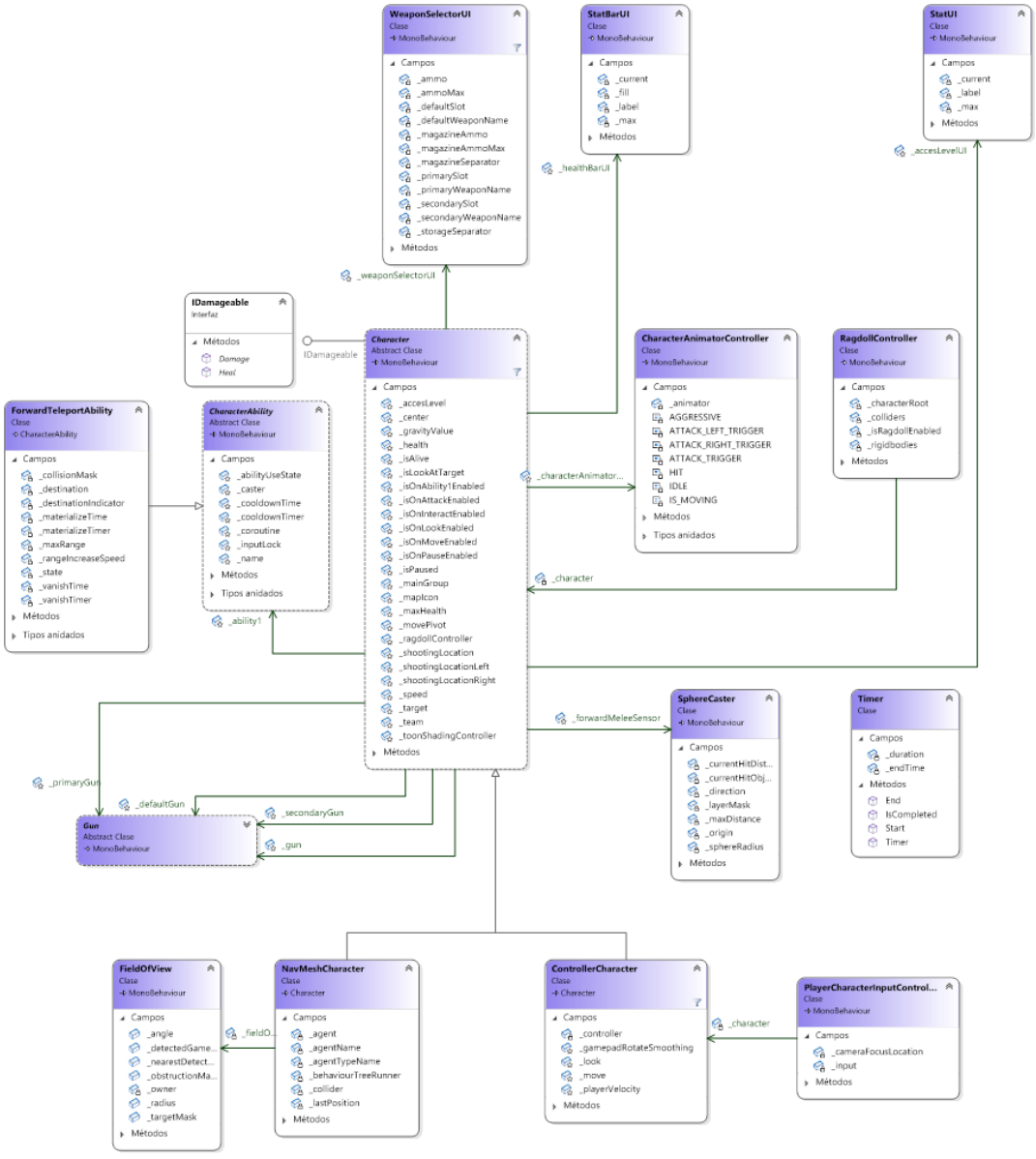

FIGURA 8.15: Diagrama de classes centrar en el personatge.

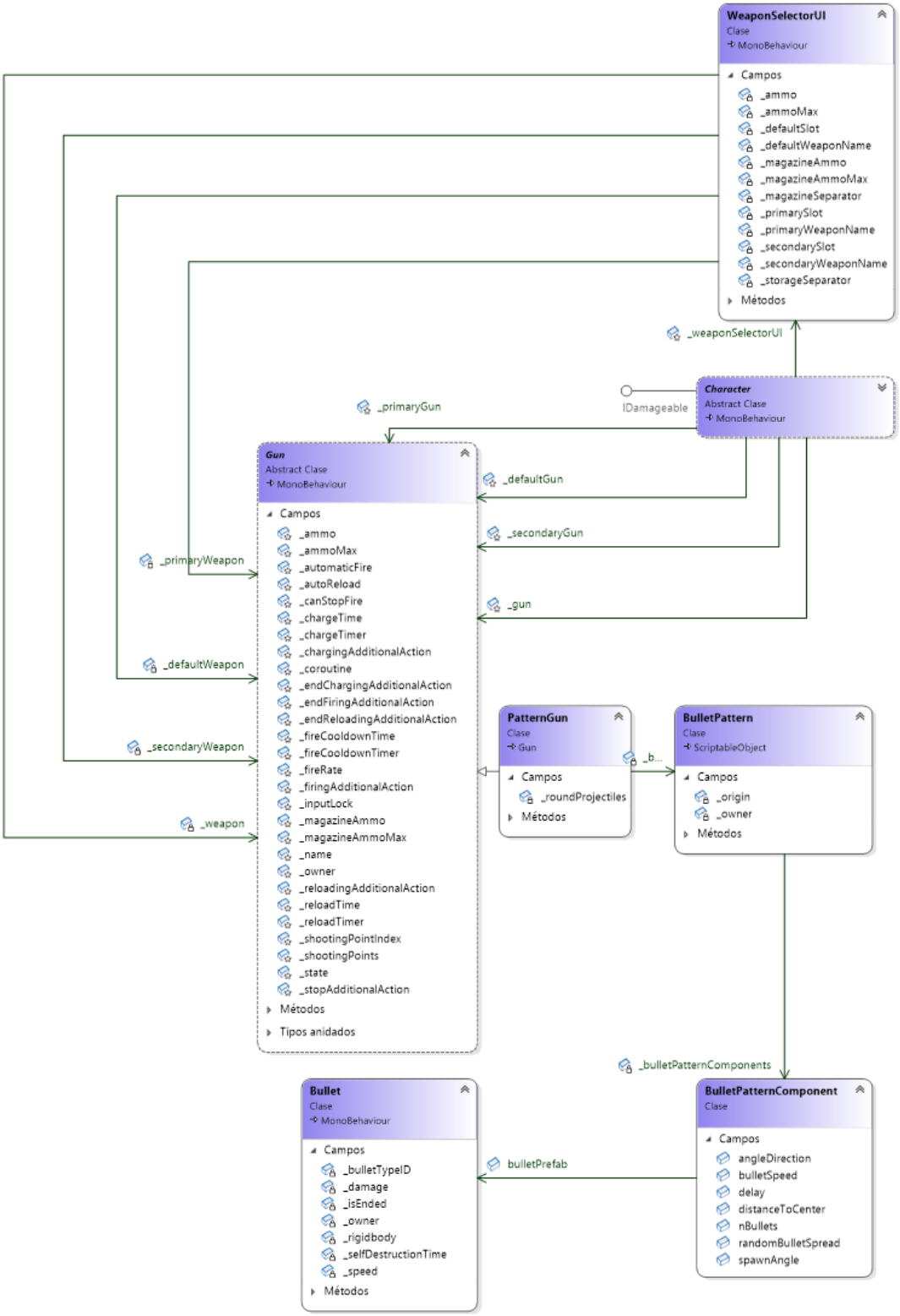

FIGURA 8.16: Diagrama de classes centrat en l'arma.

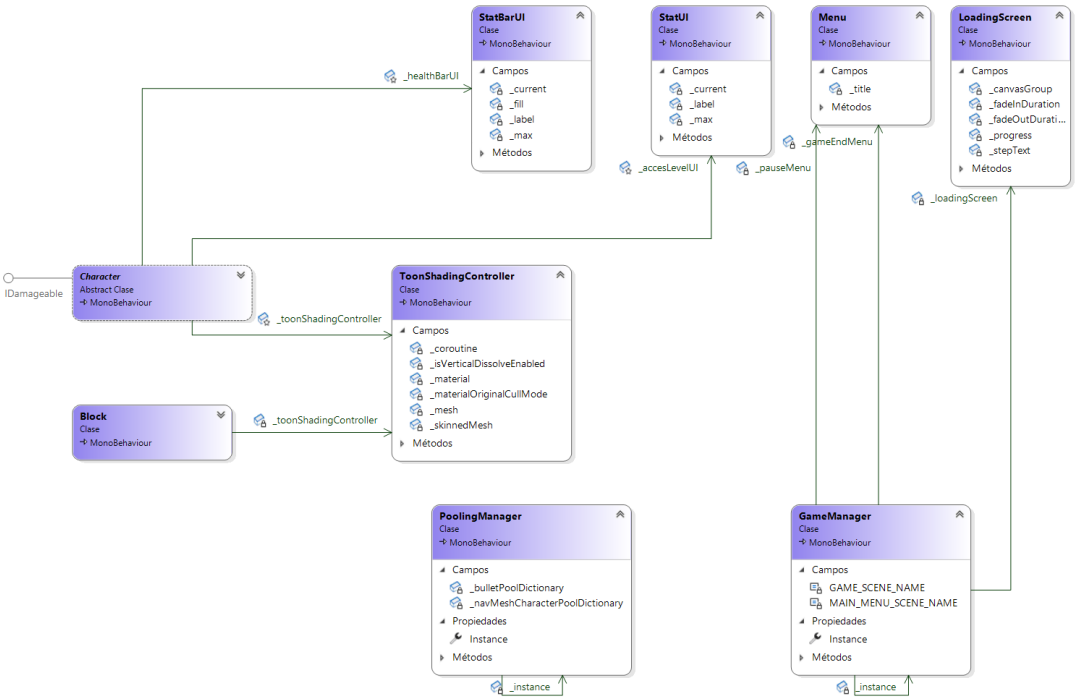

FIGURA 8.17: Diagrama de classes centrat en els elements restants.

8.6.1 FloorData

Consisteix en la classe "serialitzable" per a la lectura del fitxer *JSON* que defineix com ha de construir-se el pis.

Atributs:

- **public bool random**: Si és cert indica que la construcció de l'estructura del pis pot ser aleatòria.
- **public int rowModules**: Nombre de files de l'estructura.
- **public int columnModules**: Nombre de columnes de l'estructura.
- **public int moduleSize**: Dimensions en blocs de cada cella de l'estructura (mòdul).
- **public RoomData[] roomsData**: Informació sobre la generació interna de cada sala
- **public RoomLocationContraintsData[] roomsLocationContraintsData**: Informació sobre les restriccions a l'hora de col·locar cada sala en l'estructura del nivell.

8.6.2 RoomData

Consisteix en la classe "serialitzable" per a la lectura del fitxer *JSON* que defineix com ha de ser una sala.

Atributs:

- **public string name**: Nom de la sala.
- **public bool isMainRoomOnly**: Si és cert indica que la sala només pot ser utilitzada per a ser col·locada mitjançant les restriccions de col·locació de sales del pis.
- **public bool matrixIsStructure**: Si és cert indica que *matrix* defineix la forma de la sala per a la construcció de l'estructura del pis. No s'ha implementat l'alternativa de fals per a poder definir l'estructura interna de la sala.
- **public int matrixColumns**: Indica el nombre de particions de *matrix*.
- **public int[] matrix**: Defineix com ha de ser la sala.

8.6.3 RoomLocationContraintsData

Consisteix en la classe "serialitzable" per a la lectura del fitxer *JSON* que defineix quines restriccions de col·locació mantindrà un tipus de sala.

Atributs:

- **public string roomName**: Nom de la sala a col·locar.

- **public string linkRoomName**: Nom de la sala que reemplaçarà la sala amb *roomName* en l'estructura del nivell (connectades amb un tele-transportador).
- **public string fixedType**: Tipus de sala ("normal", "entrance", "exit", "optional"). Només una restricció ha de tenir el tipus "entrance" i com a mínim i ha d'haver-n'hi una amb el tipus "exit".
- **public int fixedRotation**: Indica com ha d'estar girada la sala en graus (0, 90, 180, 270). Un -1 indica aleatori.
- **public int objectiveNumber**: Indica el nivell d'accés de la sala. Un -1 serveix per a indicar el nivell d'accés més alt, un 0 indica sempre accessible i un 1 es troba reservat per a l'entrada.
- **public int minQuantity**: Indica el nombre mínim de sales amb *roomName* a col·locar. Un -1 indica aleatori.
- **public int aditionalQuantity**: Indica el nombre adicional de sales amb *roomName* a col·locar. Un -1 indica aleatori.
- **public int probabilitySpawn**: És la probabilitat, per cada sala indicada amb *aditionalQuantity*, de ser col·locada.
- **public int[] areaSpawnSuperior**: Cel·la que marca el límit superior esquerra en la matriu de l'estructura del nivell on la sala pot ser col·locada.
- **public int[] areaSpawnInferior**: Cel·la que marca el límit inferior dret en la matriu de l'estructura del nivell on la sala pot ser col·locada.

8.6.4 Floor

És la classe principal per a la generació aleatòria de nivells. Permet crear instàncies amb *_floorData* per a generar automàticament un pis seguint els paràmetres indicats. Aquest pis es generat de manera lògica i cal instanciar-lo a l'escena de *Unity* amb mètodes que aquesta mateixa classe proporciona.

Atributs:

- **FloorData _data**: Informació sobre la generació del pis.
- **GameObject _floorAndStructureGroup**: Grup en l'escena que conté la totalitat l'estructura del pis i el propi pis.
- **GameObject _structureAllGroup**: Grup en l'escena que conté la totalitat de l'estructura del pis.
- **GameObject _structureGroup**: Grup en l'escena que conté l'estructura del pis.
- **GameObject _externalStructureGroup**: Grup en l'escena que conté l'estructura externa del pis.
- **GameObject _floorAllGroup**: Grup en l'escena que conté la totalitat del pis.
- **GameObject _floorGroup**: Grup en l'escena que conté el pis

- **GameObject _externalFloorGroup**: Grup en l'escena que conté els elements externs del pis.
- **Dictionary<string, RoomData> _roomNamesRoomDatas**: Associació de cada nom d'habitació amb la seva informació de generació.
- **List<RoomData> _pathingRoomsDatas**: Llista amb totes les informacions de generació d'habitació que poden ser utilitzades per a generar camins entre habitacions.
- **Cell[,] _floor**: Matriu amb totes les *Cell* que constitueixen el pis.
- **Room[,] _structure**: Matriu amb la col·locació de les habitacions en el pis.
- **Hashtable _roomsPoints**: Associació de cada habitació amb les posicions que ocupa en *_structure*.
- **Hashtable _roomsPlacements**: Associació de cada habitació amb la posició (punt superior esquerra) que ocupa en *_structure*.
- **Hashtable _pointsRooms**: Associació de cada punt en *_structure* amb la habitació que contenen.
- **List<Point2D> _availablePoints**: Llistat de les posicions no ocupades per una habitació en *_structure*
- **Room _entranceRoom**: Referencia a l'habitació d'entrada.
- **HashSet<Room> _exitRooms**: Referencies a les habitacions de sortida.
- **HashSet<Room> _subObjectiveRooms**: Referencies a les habitacions amb un nivell d'accés superior a 1.
- **HashSet<Room> _pathRooms**: Referencies a les habitacions utilitzades per connectar habitacions amb un nivell d'acces superior a 1.
- **HashSet<Room> _optionalRooms**: Referències a les habitacions opcionals (habitacions que no tenen per què ser accedides amb un ordre determinat).
- **HashSet<Room> _externalRooms**: Referències a les habitacions que no estan ubicades a l'estructura del pis, sinó que es troben isolades.
- **SortedDictionary<int, HashSet<Room» _objectivesTree**: Cada nivell d'accés per ordre amb les seves habitacions.
- **_nextRoomID**: Identificador a utilitzar per a la següent habitació que es col·loqui.

Mètodes:

- **public Floor(FloorData data)**: Constructor amb un fitxer d'informació del pis.
- **public void EraseAllFromScene()**: Borra tots els grups instanciats a l'escena de *Unity*.

- **public GameObject SpawnStructure(Vector3 scenePosition)**: Instància l'estructura a l'escena de *Unity* en la posició indicada. Retorna una referencia al grup de l'escena.

- **public GameObject SpawnExternalStructure(Vector3 scenePosition, Vector3 increments)**: Instància l'estructura externa a l'escena de *Unity* en la posició i separació entre sales indicades. Retorna una referencia al grup de l'escena.

- **public GameObject SpawnFloor(Vector3 scenePosition)**: Instància el pis en l'escena a *Unity* en la posició indicada. Retorna una referencia al grup de l'escena.

- **public GameObject SpawnExternalFloor(Vector3 scenePosition, Vector3 increments)**: Instància les sales externes al pis a l'escena de *Unity* en la posició i separació entre sales indicades. Retorna una referencia al grup de l'escena.

- **public GameObject SpawnFloorAll(Vector3 scenePosition, Vector3 increments)**: Instància el pis i les sales externes a l'escena de *Unity* en la posició i separació entre sales externes indicades. Retorna una referencia al grup de l'escena.

- **public GameObject SpawnStructureAll(Vector3 scenePosition, Vector3 increments)**: Instància l'estructura del pis i de les sales externes a l'escena de *Unity* en la posició i separació entre sales externes indicades. Retorna una referencia al grup de l'escena.

- **SpawnFloorAndStructureGroup(Vector3 scenePosition, Vector3 increments)**: Instància tota l'estructura del pis i el pis (també de les sales externes) a *Unity* en la posició i separació entre sales externes indicades. Retorna una referencia al grup de l'escena.

- **public Point2D GetStructurePointToFloorPoint(Point2D structurePoint)**: Tradueix una posició de *_structure* a una posició en *_floor*.

- **public Point2D GetFloorPointToStructurePoint(Point2D floorPoint)**: Tradueix una posició de *_floor* a una posició en *_structure*.

- **void GenerateStructure()**: Si és possible, crea l'estructura del pis intentant seguir les indicacions de *_data*. Si no es pot complir alguna restricció, s'aplica una correcció (canviar la rotació, la posició o externalitzar la sala amb una sala vinculada). Només en el cas que s'hagin definit més habitacions principals, que posicions en *_structure* s'avortarà l'execució.

- **bool MainRoomsPathChecker(bool pathCorrection = false, bool allowNullCellMovement = false)**: Comprova si les habitacions principals (habitacions amb un nivell d'accés superior a 1) són accessibles respectant el seu ordre. Aquesta funció permet habilitar una correcció (*pathCorrection*) externalitzant habitacions que creen un bloqueig, substituint l'habitació conflictiva per l'habitació més senzilla de *_pathRooms* (habitació d'un sol mòdul) amb una teletransportador cap a ella. Addicionalment permet indicar si aquesta comprovació pot passar lliurement per posicions sense sala (*allowNullCellMovement*).

- **public void ExternalizeRoom(Room room, Room linkRoom, Point2D linkRoomPoint)**: Substitueix una habitació per una altra, deixant-les vinculades.

- **void MainRoomsPlacements()**: Col·loca totes les habitacions principals (habitacions amb restriccions de col·locació) en la *_structure* del pis seguint les indicacions de *_data*.

- **void NonRandomGeneration()**: Si és possible, crea l'estructura del pis seguint les indicacions de *_data* sense aplicar aleatorietat, altrament avorta l'execució.

- **void ConnectRooms(int directionMultiplier = 2)**: Connecta les habitacions principals, col·locant habitacions de *_pathRooms* en *_structure*, seguint l'ordre d' *_objectivesTree*.

- **Point2D RandomMoveAndPlaceRoom(Point2D currentPoint, Room destinationRoom = null, bool ignoreObjectiveRequirement = true, int directionMultiplier = 2)**: Donada una posició realitza un moviment pseudoaleatori vertical o horitzontal (si hi ha una habitació de destí especificada existirà més possibilitat d'anar cap a ella). Si la nova posició no conté habitació, col·loca una habitació de *_pathRooms* aleatòria.

- **public void PlaceRandomRoomsInPath(List<Vector2Int> path)**: Donada una llista de posicions col·loca habitacions de *_pathRooms* en les posicions que no continguin habitació.

- **void OptionalRoomsPlacements()**: Col·loca les habitacions opcionals en el pis de manera aleatòria però adjacents a altres habitacions.

- **HashSet<Point2D> GetRoomAdjacencies(Room room)**: Donada una habitació en el pis retorna una llista de posicions de *_structure* amb les seves adjacències.

- **HashSet<Point2D> GetPositionRoomAdjacencies(Point2D point)**: Donada una posició retorna les habitacions adjacents a ella.

- **bool NeighbourRooms(Room roomA, Room roomB)**: Donades dues habitacions retorna si són adjacents o no.

- **bool RandomRoomPlacement(RoomData roomData, bool onlyRoomAdjacent = false, Point2D superiorPoint = null, Point2D inferiorPoint = null, string fixedType = "subObjective", int fixedRotation = -1, int objectiveNumber = 0, RoomData linkRoomData = null, bool placeLinkRoomDirectly = false)**: Donat un seguit de paràmetres col·loca una habitació en una ubicació aleatòria. Retorna cert si ha estat possible.

- **bool RoomPlacement(Room room, Point2D placement, Room linkRoom = null, bool ignoreObjectiveRequirement = false, bool onlyRoomAdjacent = false)**: Donat un seguit de paràmetres col·loca una habitació en una ubicació determinada. Retorna cert si ha estat possible.

- **void AssignRoomId(Room room)**: Donada una habitació li assigna com a identificador el *_nextRoomID* i l'augmenta en 1.

- **Tuple<bool, List<Point2D» CanPlaceRoom(Room room, Point2D placement, bool onlyRoomAdjacent = false)**: Retorna si una habitació pot ser col·locada en una posició determinada i també en retorna una llista amb les posicions que

ocuparia.

- **bool[][] GetWalkableRoomsMatrix(Room[,] matrix, int objectivesAcces, bool allowNullCellMovement = true)**: Donada una matriu d'habitacions i un nivell d'accés construeix una matriu amb cert si la posició és accessible i fals altrament.
- **void GenerateRooms()**: Genera l'interior de totes les habitacions.
- **public List<Tuple<Point2D, List<Constants.DIRECTION»> GetRoomDoorsPlacements(Room room)**: Donada una habitació en *_structure* retorna una llista amb les posicions on hi ha d'haver una porta i les seves direccions.
- **public void DoorsPlacements()**: Col·loca totes les portes i tele-transportadors del pis.
- **public void ActualizeDoorsConnections()**: Actualitza totes les portes i tele-transportadors de l'habitació amb les connexions que presenten.
- **public bool IsExternalRoom(Room room)**: Donada una habitació retorna cert si és externa.

8.6.5 Room

Constitueix una classe essencial per a la generació aleatòria de nivells. Aquesta classe és la que *Floor* utilitza per crear instanciar que s'encarreguin de la gestió i generació interna de les sales que organitza. També conte diversos mètodes de control per controlar els accessos de l'habitació instanciada en l'escena de *Unity* així com per a poder col·locar-hi enemics.

Atributs:

- **Floor _floor**: Referencia al pis.
- **RoomData _data**: Informació sobre la generació interna de la sala.
- **Point2D _floorStructurePoint**: Posició de l'habitació en l'estructura del pis.
- **GameObject _structureGroup**: Grup dins l'escena de *Unity* on es troba instanciada l'estructura de l'habitació.
- **GameObject _roomGroup**: Grup dins l'escena de *Unity* on es troba instanciada l'habitació.
- **Constants.ROTATION _rotation**: Rotació amb la qual es troba col·locada l'habitació.
- **Cell[,] _structure**: Matriu amb totes les *Cell* que constitueixen l'estructura de l'habitació.
- **Cell[,] _room**: Matriu amb totes les *Cell* que constitueixen l'habitació.
- **string _type**: Identificador del tipus de l'habitació ("normal", "entrance", "exit", "optional").

- **int _objectiveNumber**: Nivell d'accés de l'habitació.
- **public int _id**: Identificador de la l'habitació.
- **float _height**: Alçada en la qual l'habitació es troba instanciada (serveix per a les habitacions externes).
- **public Room _linkedRoom**: Habitació amb la qual aquesta habitació es troba vinculada.
- **Point2D _linkedRoomTeleporterPoint**: Posició en la qual es troba el tele-transportador cap a *_linkedRoom*, si aquesta habitació en té.
- **bool _isExit**: Indica si l'habitació és de sortida i, per tant, ha de tenir un tele-transportador de nivell.
- **Point2D _exitPoint**: Posició en la qual es troba el tele-transportador de nivell, si aquesta habitació en té.
- **HashSet<Cell> _cells**: Contenidor amb totes les cel·les de l'habitació.
- **HashSet<Point2D> _doorsPoints**: Contenidor amb totes les posicions que contenen una porta.
- **HashSet<Point2D> _groundPoints**: Contenidor amb totes les posicions que constitueixen el terra de l'habitació.
- **bool _isCleared**: Indica si els objectius de l'habitació han estat completats (el jugador ha eliminat tots els enemics).
- **HashSet<Character> _spawnedEnemies**: Contenidor amb tots els enemics instanciats en l'habitació.
- **Dictionary<string, NavMeshSurface> _navMeshSurfaces**: Per cada tipus d'agent (en base les seves dimensions) el seu *NavMesh* en l'habitació.

Mètodes:

- **public Room(Floor floor, RoomData data, Point2D floorStructurePoint, Constants.ROTATION rotation, string type, int objectiveNumber)**: Constructor amb un fitxer d'informació de l'habitació i diversos paràmetres que constitueixen el context en la que es col·loca en el pis.
- **public Room(Room room)**: Constructor de copia.
- **public GameObject SpawnStructure(Vector3 scenePosition)**: Grup en l'escena de *Unity* que conté l'estructura de l'habitació.
- **public bool IsStructureAcces(Point2D position)**: Retorna cert si la posició és un accés a l'habitació.
- **public void LinkRoom(Room room)**: Crea un amb l'habitació indicada.

- **private void GenerateStructure()**: Crea l'estructura de l'habitació seguint les indicacions de *_data*.
- **private void Rotate(Constants.ROTATION rotation)**: Gira les matrius que constitueixen l'estructura de l'habitació i la mateixa habitació segons la rotació indicada.
- **private Cell[,] Rotate90Clock(Cell[,] matrix)**: Gira 90 graus en sentit horari les matrius que constitueixen l'estructura de l'habitació i la mateixa habitació.
- **private Cell[,] Rotate90counterClock(Cell[,] matrix)**: Gira 90 graus en sentit anti horari les matrius que constitueixen l'estructura de l'habitació i la mateixa habitació.
- **private Cell[,] Rotate180(Cell[,] matrix)**: Gira 180 graus en sentit horari les matrius que constitueixen l'estructura de l'habitació i la mateixa habitació.
- **public void Generate()**: Crea l'interior de l'habitació seguint les indicacions de *_data*.
- **public GameObject Spawn(Vector3 scenePosition)**: Instància l'habitació en l'escena de *Unity* i en retorna el grup.
- **public void SetDoors(List<Tuple<Point2D, List<Constants.DIRECTION»> doorsPlacements)**: Donat un llistat de posicions i direccions, assigna com s'han de col·locar les portes.
- **private void BasicGeneration()**: Genera l'interior de l'habitació de manera bàsica, posant portes o parets al seu contorn, i terra a tots els blocs interiors.
- **private bool IsModuleLimit(Point2D point)**: Donada una posició retorna si està al límit d'un mòdul.
- **private bool IsModuleMid(Point2D point)**: Donada una posició retorna si està al mitg d'un mòdul.
- **private bool PointInStructure(Point2D point)**: Donada una posició retorna si aquesta pertany a l'estructura de l'habitació.
- **private bool PointInRoom(Point2D point)**: Donada una posició retorna si aquesta pertany a l'habitació.
- **public void ActualizeDoorsConnections()**: Actualitza totes les portes i teletransportadors de l'habitació amb les connexions que presenten.
- **void CreateAllAgentTypesNavMesh()**: Per cada tipus existent d'agent (segons les seves dimensions), genera el *NavMesh* de la sala.
- **public void BakeAgentNavMesh(string agentTypeName = null)**: Donat un tipus d'agent (segons les seves dimensions), genera el *NavMesh* de la sala.
- **private int? GetNavMeshAgentTypeID(string name)**: Donat un tipus d'agent retorna el seu identificador.

- **public KeyValuePair<Point2D, Vector3> GetRandomNavMeshPosition()**:Retorna la posició lògica i la posició en escena de *Unity* d'un punt aleatori en el *NavMesh* de la sala.
- **public Point2D GetRandomAccesiblePoint()**: Retorna una posició aleatòria dins de *_groundPoints*.
- **public void SpawnEnemies(Dictionary<NavMeshCharacter, int> enemiesToSpawn)**: Instància el tipus i nombre d'enemics indicats dins l'habitació de l'escena de *Unity*.
- **public void SetDoorsClosed(bool closed)**: Segons la indicació tenca o obre totes les portes.
- **public void SetDoorsLocked(bool locked)**: Segons la indicació, tenca i bloqueja l'obertura de totes les portes. També fa desaparèixer o aparèixer els teletransportadors.
- **public void RemoveEnemy(Character enemy)**: Elimina un enemic dins l'habitació de la llista de *_spawnedEnemies*.
- **public void ActualizeIsClear()**: Comprova si queden enemics dins de *_spawnedEnemies*. Si no en queden, ho notifica al *GameManager*, marca l'habitació com a superada, desbloqueja totes les portes i fa aparèixer tots els teletransportadors.

8.6.6 Cell

Consisteix en una estructura lògica que manté organitzat els diversos blocs que conte cada posició d'una habitació i es la que s'ocupa d'instanciar-los a l'escena.

Atributs:

- **Room _room**: Referència a l'habitació a la qual pertany.
- **Point2D _roomPoint**: Posició logica que ocupa dins l'habitació.
- **GameObject _cellGroup**: Grup dins l'escena de *Unity* on es troba instanciada.
- **List<GameObject> _blocksPrefabs**: Llista dels *prefabs* de tots els blocs que constitueixen la cel·la (en forma de pila).
- **List<GameObject> _blocks**: Llista de tots els blocs instanciats en l'escena de *Unity* constitueixen la cel·la (en forma de pila).
- **bool _isStructure**: Indica si la cel·la és d'estructura del pis.

Mètodes:

- **public Cell(Room room, Point2D roomPoint, List<GameObject> tilesPrefabs, bool isStructure = false)**: Construeix una cel·la amb els paràmetres indicats.
- **public GameObject Spawn(Vector3 scenePosition)**: Instància tots els blocs de

la cel·la de manera ascendent en la posició indicada de l'escena de *Unity* en un grup. Retorna el grup.

- **public void AddTopBlockPrefab(GameObject blockPrefab)**: Afegeix un *prefab* de bloc al final de la llista *_blocksPrefabs*.

8.6.7 Block

Són els elements resultants de la generació del nivell dins de l'escena de *Unity* en forma de component. Totes les instàncies mantindran un accés a la seva representació lògica mitjançant una referència a la seva *Cell*.

Atributs:

- **GameObject _mapIcon**: Referència a la icona que substituirà aquest objecte en el minimapa.
- **Cell _cell**: Referència a la cel·la a la qual pertany el bloc dins la representació lògica del pis i l'habitació.
- **bool _isStructure**: Determina si forma part de la construcció de l'estructura.
- **ToonShadingController _toonShadingController**: Referència al component de control del seu material.

Mètodes:

- **void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir.
- **public void Init(Cell cell, bool isStructure = false)**: Estableix la vinculació amb la seva representació lògica.
- **public void Materialize(float time = 1)**: Inicia una interpolació del material que fa aparèixer l'objecte.
- **public void Dematerialize(float time = 1)**: Inicia una interpolació del material que fa desaparèixer l'objecte.

8.6.8 Door

Consisteix en un component que es combina amb un *Block* en l'escena de *Unity* per a fer d'accés entre dues habitacions adjacents. Una porta es pot travessar mentre es troba tancada o bloquejada.

Atributs:

- **Color _unlockedColor**: Color per a la porta desbloquejada.
- **Color _lockedColor**: Color per a la porta bloquejada.
- **float _animationTime**: Temps de l'animació d'obertura i tencament de la porta.

- **bool _isOpen**: Indica si la porta es troba oberta.
- **bool _isLocked**: Indica si la porta es troba bloquejada.
- **TextMesh _objectiveNumberLabel**: Indica el nivell d'accés de l'habitació adjacent.
- **Block _block**: Referència al component de *Block*.
- **Collider _collider**: Referència al component de *Collider*.
- **Door _connectedDoor**: Referència a la porta adjacent.

Mètodes:

- **void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir.
- **public void ActualizeConnectedDoor()**: Actualitza la porta adjacent.
- **public void Interact(GameObject interactor = null)**: Obre la porta si la porta està tancada, no està bloquejada i qui interactua amb ella té el nivell d'accés necessari.
- **public void Open(bool openConnectedDoor = false)**: Obre la porta, i si s'ha indicat, també la porta adjacent.
- **public void Close(bool closeConnectedDoor = false)**: Tanca la porta, i si s'ha indicat, també la porta adjacent.
- **private void OnTriggerExit(Collider other)**: Tanca la porta travessada i l'adjacent indicant al *LevelManager* que s'ha produït una transició d'habitacions, si el jugador ha travessat les dues portes que separen una sala (havien d'estar prèviament obertes).

8.6.9 LinkTeleporter

Consisteix en un component que es combina amb un *Block* en l'escena de *Unity* per a fer d'accés entre dues habitacions vinculades a distància tele-transportant al personatge.

Atributs:

- **float _vanishTime**: Duració de l'animació de desaparició del personatge.
- **float _materializeTime**: Duració de l'animació d'aparició del personatge.
- **TextMesh _objectiveNumberLabel**: Indica el nivell d'accés de l'habitació vinculada de manera visual en l'escena de *Unity*.
- **Block _block**: Referència al component de *Block*.
- **LinkTeleporter _linkedTeleporter**: Referència al tele-transportador vinculat.

- **Timer _vanishTimer**: Temporitzador de l'animació de desaparició del personatge.
- **Timer _materializeTimer**: Temporitzador de l'animació d'aparició del personatge.

Mètodes:

- **void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir.
- **void Start()**: Abans de començar amb el *Update* inicialitza els temporitzadors, obté el tele-transportador vinculat i estableix el text de nivell d'accés a mostrar.
- **public void Interact(GameObject interactor)**: Inicia el procés de tele-transportació del personatge si aquest té el nivell d'accés requerit.
- **IEnumerator Teleport(GameObject gameObject)**: Executa la tele-transportació del personatge en el temps. Comença amb la desaparició del personatge durant el temps indicat per *_vanishTime*. Passat aquest temps tele-transporta el personatge al tele-transportador destí i comença l'aparició del personatge amb el temps indicat per *_materializeTime*.

8.6.10 LevelTeleporter

Consisteix en un component que es combina amb un *Block* en l'escena de *Unity* per a fer de fi del pis, finalitzant la partida quan el jugador hi interactuï.

Atributs:

- **float _vanishTime**: Duració de l'animació de desaparició del personatge.
- **Block _block**: Referència al component de *Block*.
- **Timer _vanishTimer**: Temporitzador de l'animació de desaparició del personatge.

Mètodes:

- **void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir.
- **void Start()**: Abans de començar amb el *Update* inicialitza els temporitzadors-
- **public void Interact(GameObject interactor)**: Inicia el procés de desaparició del personatge.
- **IEnumerator Teleport(GameObject gameObject)**: Executa la desaparició del personatge en el temps. Comença amb la desaparició del personatge durant el temps indicat per *_vanishTime*. Passat aquest temps comunica al *GameManager* la finalització de la partida.

8.6.11 IInteractable

Interfície utilitzada per a poder detectar si un *GameObject* pot ser interactuat.

Mètodes:

- **void Interact(GameObject interactor = null)**: Mètode a implementar per a interactuar amb l'objecte indicant-li qui interactua amb ell.

8.6.12 Point2D

Classe que serveix per a representar un punt 2D.

Atributs:

- **public int x**:
- **public int y**:

Mètodes:

- **public Point2D(int x, int y)**: Constructor amb dos nombres enters.
- **public Point2D(float x, float y)**: Constructor amb dos nombres decimals.
- **public Point2D(Vector3 v3)**: Constructor amb un *Vector3*. Com que la coordenada *y* a *Unity* apunta cap amunt es prenen els valors *x* i *z* de *v3*.
- **public static bool operator == (Point2D b1, Point2D b2)**: Operador d'equivalència.
- **public static bool operator != (Point2D b1, Point2D b2)**: Operador de no equivalència.
- **public static Point2D operator + (Point2D b1, Point2D b2)**: Operador de suma.
- **public override bool Equals(object obj)**: Determina si es equivalent a *obj*.

8.6.13 Timer

Classe que serveix com a temporitzador.

Atributs:

- **float _duration**: Duració del temporitzador.
- **float _endTime**: Temps de finalització del temporitzador.

Mètodes:

- **public Timer(float duration = 0.01f)**: Construeix un temporitzador amb el temps indicat.

- **public bool IsCompleted()**: Determina si el temporitzador ha finalitzat comprovant si el temps actual és superior a *_endTime*.
- **public void Start()**: Inicialitza la temporització guardant el temps de fi (el temps actual més *_duration*) a *_endTime*.
- **public void End()**: Finalitza la temporització de manera prematura posant *_endTime* a 0.

8.6.14 Constants

Classe per a organitzar diverses constants i mètodes que treballen amb elles.

Atributs:

- **public const string ROOM_SINGLE**: Constant que indica el nom de l'habitació més bàsica i essencial (habitació d'un únic mòdul o cel·la en l'estructura) per a la construcció d'un pis. L'aparició de la definició d'almenys una habitació d'aquest tipus amb la que es puguin generar camins en el JSON de generació, és un requisit mínim. Aquest tipus d'habitacions són les que es fan servir per a substituir una sala principal quan aquesta no pot ser col·locada directament.
- **public enum DIRECTION { NONE, TOP, RIGHT, BOT, LEFT }**: Grup de constants per a indicar una direcció.
- **public enum ROTATION { NONE, CLOCK90, CLOCK180, CLOCK270 }**: Grup de constants per a indicar una rotació.
- **public static List<ROTATION> ROTATIONS**: Llista amb totes les constants de *ROTATION*.

Mètodes:

- **public static ROTATION intToRotation(int number)**: Tradueix EL nombre enter indicat a *ROTATION* si aquesta es troba definida. Altrament retorna *ROTATION.NONE*.

8.6.15 Character

Consisteix en una classe abstracta amb els atributs i mètodes que compartiran els components per al personatge controlat pel jugador i els personatges controlats per les IA.

Atributs:

- **protected GameObject _mainGroup**: Grup en l'escena de *Unity* en la que es troba el personatge.
- **protected GameObject _mapIcon**: Referència a la icona que substituirà aquest objecte en el minimapa.
- **protected int _maxHealth**: Vida Màxima.
- **protected int _health**: Vida actual.

- **protected float _speed**: Velocitat.
- **protected float _gravityValue**: Gravetat que afecta el personatge.
- **protected int _team**: Equip al qual pertany.
- **protected int _accesLevel**: Nivell d'accés.
- **protected Transform _center**: Referència del centre del personatge en l'escena de *Unity*.
- **protected Transform _shootingLocation**: Referència al punt de dispar central del personatge en l'escena de *Unity*.
- **protected Transform _shootingLocationLeft**: Referència al punt de dispar esquerra del personatge en l'escena de *Unity*.
- **protected Transform _shootingLocationRight**: Referència al punt de dispar dret del personatge en l'escena de *Unity*.
- **protected Transform _movePivot**: Referència al punt pivot del moviment del personatge en l'escena de *Unity*. Aquest punt es fa servir per a rotar l'esquelet del personatge en el moviment.
- **protected Transform _target**: Referència a l'objectiu.
- **protected CharacterAbility _ability1**: Referència a l'habilitat.
- **protected Gun _defaultGun**: Referència a l'arma per defecte.
- **protected Gun _primaryGun**: Referència a l'arma primària.
- **protected Gun _secondaryGun**: Referència a l'arma secundària.
- **protected bool _isOnMoveEnabled**: Indica si el moviment està habilitat.
- **protected bool _isOnLookEnabled**: Indica si l'apuntat està habilitat.
- **protected bool _isOnAttackEnabled**: Indica si l'atac està habilitat.
- **protected bool _isOnAbility1Enabled**: Indica si utilitzar l'habilitat està habilitada.
- **protected bool _isOnInteractEnabled**: Indica si utilitzar la interacció està habilitada.
- **protected bool _isOnPauseEnabled**: Indica si posar en pausa la partida està habilitada.
- **protected SphereCaster _forwardMeleeSensor**:Sensor que detecta objectes propers davant del personatge.
- **protected WeaponSelectorUI _weaponSelectorUI**: Referència a la interfície de selecció de l'arma.

- **protected StatBarUI _healthBarUI**: Referència a la interfície de la barra de vida.
- **protected StatUI _accesLevelUI**: Referència a la interfície d'informació sobre el nivell d'accés.
- **protected RagdollController _ragdollController**: Referència al controlador de l'animació dinàmica de mort del personatge.
- **protected ToonShadingController _toonShadingController**: Referència al controlador del material del personatge.
- **protected CharacterAnimatorController _characterAnimatorController**: Referència al controlador d'animacions del personatge.
- **protected Gun _gun**: Referència a l'arma seleccionada.
- **protected bool _isAlive**: Indica si el personatge està viu.
- **protected bool _isPaused**: Indica si el personatge està pausat.
- **protected bool _isLookAtTarget**: Indica si el personatge està apuntant a l'objectiu.

Mètodes:

- **virtual protected void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir, associa les animacions del personatge amb les armes, inicialitza l'estat del personatge i actualitza la informació a mostrar en la interfície.
- **virtual protected void Initialize()**: Inicialitza l'estat del personatge i actualitza la informació a mostrar en la interfície.
- **public void BindGun(Gun gun, int gunSlot = 0)**: Associa una arma al personatge a la posició indicada (per defecte, primària o secundària)
- **private void BindCharacterToGun(Gun gun)**: Associa a l'arma les animacions del personatge.
- **public void Damage(int damage)**: Resta la quantitat de vida indicada al personatge. Si la vida resultant és inferior i igual a 0, executa la funció *Die()*.
- **public void Heal(int health)**: Suma la quantitat de vida indicada al personatge sense sobrepassar-ne el màxim.
- **virtual public void Die()**: Desactiva els controls del personatge, cancel·la totes les accions en curs i inicia l'animació de mort.
- **virtual public void Resurrect()**: Posa la vida actual igual a la vida màxima, inicia l'animació d'aparició i habilita els controls del personatge.
- **virtual public void Spawn()**: Fa aparèixer el personatge en el punt on es trobava abans de desaparèixer.

- **virtual public void Despawn()**: Per fer des-aparèixer el personatge (no hi ha implementació base)
- **protected void LookAt(Vector3 lookPoint)**: Fa que el personatge apunti a la posició indicada.
- **protected Vector2 CartesianToPolar(Vector3 point)**: Retorna la traducció d'una posició cartesiana a polar.
- **protected Vector3 PolarToCartesian(Vector2 polar)**: Retorna la traducció d'una posició polar a cartesiana.
- **public GameObject GetOverlapGameObject()**: Retorna l'objecte que envolta el personatge (serveix per a interactuar amb *Colliders* del tipus *IsTrigger*).

8.6.16 IDamageable

Interfície utilitzada per a poder detectar si un *GameObject* pot rebre mal.

Mètodes:

- **public void Damage(int damage)**: Mètode a implementar per a rebre mal segons el valor indicat.
- **public void Heal(int health)**: Mètode a implementar per a ser curat segons el valor indicat.

8.6.17 ControllerCharacter

Component derivat de *Character* per a personatges controlats pel jugador mitjançant un *CharacterController*.

Atributs:

- **protected float _gamepadRotateSmoothing**: Valor de suavització de la rotació d'apuntat per a quan es fa servir un controlador.
- **CharacterController _controller**: Referència al controlador del personatge.
- **protected Vector3 _playerVelocity**: Velocitat actual.
- **protected Vector2 _look**: Últim canvi de direcció d'apuntat realitzat.
- **protected Vector2 _move**: Últim canvi de direcció de moviment realitzat.

Mètodes:

- **override protected void Awake()**: Extensió del mètode *Awake()* de *Character* en la que es crea una referència al component *CharacterController* de l'objecte.
- **void Update()**: Actualitza a cada "frame" el moviment i l'apuntat si el personatge no esta mort ni pausat.

- **public void Ability1()**: Cancel·la l'acció actual de l'arma i executa el mètode *Trigger()* de l'habilitat si el personatge no està mort ni pausat i l'habilitat es troba habilitada.
- **public void Ability1Release()**: Executa el mètode *ReleaseTrigger()* de l'habilitat si el personatge no està mort ni pausat i l'habilitat es troba habilitada.
- **public void Attack()**: Executa el mètode *Trigger()* de l'arma seleccionada si aquesta no és nul·la, el personatge no està mort ni pausat i l'atac està habilitat.
- **public void AttackRelease()**: Executa el mètode *ReleaseTrigger()* de l'arma seleccionada si el personatge no està mort ni pausat i l'atac està habilitat.
- **public void Reload()**: Executa el mètode *Reload()* de l'arma seleccionada si el personatge no està mort ni pausat i l'atac està habilitat.
- **public void SelectDefaultWeapon()**: Selecciona l'arma per defecte si aquesta no és nul·la, no està seleccionada i el personatge no està mort ni pausat i l'atac està habilitat.
- **public void SelectPrimaryWeapon()**: Selecciona l'arma primària si aquesta no és nul·la i no està seleccionada, el personatge no està mort ni pausat i l'atac està habilitat.
- **public void SelectSecondaryWeapon()**: Selecciona l'arma secundària si aquesta no és nul·la, no està seleccionada i el personatge no està mort ni pausat i l'atac està habilitat.
- **public void Interact()**: Interactua amb l'objecte que envolta el personatge o amb un objecte proper al davant seu si aquest existeix, és interactuable, el personatge no està mort ni pausat i la interacció està habilitada.
- **public void Menu()**: Obre el menú de pausa i pausa la partida si la pausa està habilitada. Si la partida ja es trobava pausada, l'execució d'aquest mètode tenca el menú i reprèn la partida.
- **public void Move(Vector2 move)**: Actualitza el valor de *_move* amb el valor indicat.
- **public void Look(Vector2 look)**: Actualitza el valor de *_look* amb el valor indicat.
- **void HandleMove()**: Realitza el moviment si aquest està habilitat. També gestiona l'animació de moviment dinàmica (segons la direcció on es mou el personatge) i s'encarrega de posar el personatge en "idle'" (animació de personatge quiet) si aquest no s'està movent.
- **void HandleLook()**: Realitza el canvi d'apuntat si aquest està habilitat. Si el personatge té un *_target* l'apunta automàticament en comptes d'utilitzar els valors de *_look*.
- **public override void Die()**: Extensió del mètode *Die()* de *Character* en la que

es reinicien *_move* i *_look* i s'invoca de manera retardada la funció *DelayedEndGame(float delay)*.

- **IEnumerator DelayedEndGame(float delay)**: Espera el temps indicat i seguidament notifica al *GameManager* la finalització del joc com a derrota.

8.6.18 NavMeshCharacter

Component derivat de *Character* per a personatges controlats per la IA mitjançant la combinació del *NavMeshCharacter* i el *BehaviourTreeRunner*.

Atributs:

- **string _agentName**: Identificador del personatge.
- **string _agentTypeName**: Identificador del tipus de l'agent (és el nom associat a la configuració de dimensions).
- **FieldOfView _fieldOfView**: Referencia al camp de visió.
- **NavMeshAgent _agent**: Referencia al component de *NavMeshAgent*.
- **Collider _collider**: Referència al component de *Collider*.
- **BehaviourTreeRunner _behaviourTreeRunner**: Referència al component de *BehaviourTreeRunner* per a l'execució de la IA.
- **Vector3 _lastPosition**: Ultima posició visitada en l'escena de *Unity*.

Mètodes:

- **override protected void Awake()**: Extensió del mètode *Awake()* de *Character* en la que es creen referenciés a components incorporats en aquesta classe.
- **protected override void Initialize()**: Extensió del mètode *Initialize()* de *Character* en la que s'estableix com a *_lastPosition* la posició actual i s'iguala la velocitat del *NavMeshAgent* amb la del personatge.
- **void Update()**: Actualitza a cada "frame" el moviment i l'apuntat si el personatge no esta mort ni pausat.
- **void HandleMove()**: Gestiona l'animació de moviment dinàmica (segons la direcció on es mou el personatge) i s'encarrega de posar el personatge en "idle'" (animació de personatge quiet) si aquest no s'està movent.
- **void HandleLook()**: Gestiona els canvis d'apuntat segons si aquesta està habilitat. Si el personatge té un *_target* l'apunta automàticament.
- **public override void Die()**: Extensió del mètode *Die()* de *Character* en la que es deshabilita el *_collider*, notifica al *LevelManager* la mort del personatge i s'invoca de manera retardada la funció *Despawn()*.
- **public override void Resurrect()**: Extensió del mètode *Resurrect()* de *Character* en la que s'habilita el *_collider*.

- **override public void Despawn()**: Extensió del mètode *Die()* de *Character* en el que es notifica al *PoolingManager* l'alliberació d'aquest personatge perquè pugui ser reutilitzat.
- **public void Attack()**: Executa el mètode *DirectFire()* de l'arma seleccionada si aquesta no és nul·la, el personatge no està mort ni pausat i l'atac està habilitat.

8.6.19 PlayerCharacterInputController

Component de connexió entre *PlayerInput* i *ControllerCharacter* per tal de captar els inputs del jugador, processar-los i controlar el personatge amb ells. Els mètodes precedits per *On* són executats segons l'input captat per *PlayerInput*.

Atributs:

- **Transform _cameraFocusLocation**: Referència al punt dins l'escena de *Unity* on la càmera apunta.
- **PlayerInput _input**: Referència al *PlayerInput*.
- **ControllerCharacter _character**: Referència al *ControllerCharacter*.

Mètodes:

- **void Awake()**: En instanciar-se, crea una referència als components als quals es vol poder accedir.
- **public void OnAbility1()**: Executa el mètode *Ability1* de *_character*.
- **public void OnAbility1Release()**: Executa el mètode *Ability1Release()* de *_character*.
- **public void OnAttack()**: Executa el mètode *Attack()* de *_character*.
- **public void OnAttackRelease()**: Executa el mètode *AttackRelease* de *_character*.
- **public void OnReload()**: Executa el mètode *Reload()* de *_character*.
- **public void OnSelectDefaultWeapon()**: Executa el mètode *SelectDefaultWeapon()* de *_character*.
- **public void OnSelectPrimaryWeapon()**: Executa el mètode *SelectPrimaryWeapon()* de *_character*.
- **public void OnSelectSecondaryWeapon()**: Executa el mètode *SelectSecondaryWeapon()* de *_character*.
- **public void OnInteract()**: Executa el mètode *Interact()* de *_character*.
- **public void OnMenu()**: Executa el mètode *Menu()* de *_character*.
- **public void OnMove(InputValue value)**: Captura el valor de l'input, el processa i executa el mètode *Move* de *_character* amb aquest valor.

- **public void OnLook(InputValue value)**: Captura el valor de l'input, el processa i executa el mètode *Look* de *_character* amb aquest valor.

8.6.20 FieldOfView

Component de detecció d'objectes en un angle i distància determinada, simulant un camp de visió. Es fa servir perquè la IA pugui detectar al jugador quan aquest estigui davant.

Atributs:

- **GameObject _owner**: Referència al propietari d'aquest camp de visió.
- **public float _radius**: Distancia del camp de visió.
- **public float _angle**: Angle del camp de visió
- **public GameObject _nearestDetectedGameObject**: Objecte més proper detectat.
- **public HashSet<GameObject> _detectedGameObjects**: Objectes detectats.
- **public LayerMask _targetMask**: *Tags* dels objectes a detectar.
- **public LayerMask _obstructionMask**: textitTags dels objectes que bloquegen el camp de visió.

Mètodes:

- **private void Awake()**: En instanciar-se, crea una referència al propietari i inicialitza el contenidor d'objectes detectats.
- **private void Update()**: Actualitza a cada "frame" els objectes detectats per el camp de visió.
- **private void FieldOfViewCheck()**: Actualitza els contenidors d'objectes dins del camp de visió.
- **public bool HasCleanSight(GameObject target, float sphereRadius)**: Indica si l'objecte indicat es pot veure en línia recta sense que hi hagi cap col·lisió amb una esfera del radi indicat.

8.6.21 SphereCaster

Component de detecció d'objectes amb la projecció lineal d'una esfera que s'atura amb el primer objecte col·lidit (objecte detectat).

Atributs:

- **GameObject _currentHitObject**: Referència a l'objecte detectat actualment.
- **float _sphereRadius**: Radi de l'esfera de detecció.
- **float _maxDistance**: Distància màxima de detecció.

- **LayerMask _layerMask**: textitTags dels objectes que s'ignoren en la detecció.
- **private Vector3 _origin**: Punt en l'escena de *Unity* en la que comença la projecció de l'esfera.
- **private Vector3 _direction**: Direcció que pren la projecció de l'esfera.
- **private float _currentHitDistance**: Distància en la qual l'esfera ha col·lidit amb l'objecte detectat actualment.

Mètodes:

- **void Update()**: Actualitza a cada "frame" l'objecte detectat per l'esfera.

8.6.22 CharacterAnimatorController

Component que serveix per a controlar un *Animator* amb les accions específiques dels personatges d'aquest joc.

Atributs:

- **const string IS_MOVING**: Constant amb el valor per a indicar al *_animator* que el personatge es troba en moviment.
- **const string ATTACK_TRIGGER**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **const string ATTACK_LEFT_TRIGGER**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **const string ATTACK_RIGHT_TRIGGER**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **const string AGGRESSIVE**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **const string IDLE**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **const string HIT**: Constant amb el valor per a activar l'animació de moviment del *_animator*.
- **Animator _animator**: Referència al *Animator* que controla.

Mètodes:

- **public void TriggerAttack()**: Indica al *_animator* que executi l'animació d'atac.
- **public void TriggerAttackLeft()**: Indica al *_animator* que executi l'animació d'atac esquerra.
- **public void TriggerAttackRight()**: Indica al *_animator* que executi l'animació d'atac dret.

- **public void TriggerAggressive()**: Indica al *_animator* que executi l'animació d'aggresivitat.
- **public void TriggerIdle()**: Indica al *_animator* que executi l'animació d'"idle".
- **public void TriggerHit()**: Indica al *_animator* que executi l'animació de rebre mal.
- **public void ClearLoop()**: Indica al *_animator* que aturi l'execució d'una animació en bucle.
- **public void LoopChanneling()**: Indica al *_animator* que executi en bucle l'animació de canalització.
- **public void LoopReloading()**: Indica al *_animator* que executi en bucle l'animació de recàrrega.
- **public void LoopAttackReady()**: Indica al *_animator* que executi en bucle l'animació d'atac preparat.
- **public void LoopCharging()**: Indica al *_animator* que executi en bucle l'animació de carregar un atac.
- **public void LoopContinuousAttack()**: Indica al *_animator* que executi en bucle l'animació d'atac continu.

8.6.23 RagdollController

Component que serveix per a crear una animació dinàmica de mort del personatge. Això s'aconsegueix aplicant físiques de gravetat i col·lisions als ossos del personatge de manera que es puguin fer caure a terra, simulant que aquest ha perdut la vida.

Atributs:

- **Transform _characterRoot**: Referència a l'os que encapçala la jerarquia d'ossos de l'esquelet del personatge.
- **bool _isRagdollEnabled**: Indica si l'animació de mort està aplicada.
- **Character _character**: Referència al personatge.
- **Rigidbody[] _rigidbodies**: Llista amb tots els *Rigidbody* de l'esquelet del personatge.
- **Collider[] _colliders**: Llista amb tots els *Collider* de l'esquelet del personatge.

Mètodes:

- **private void Start()**: Abans de realitzar el primer *Update()*, crea una referència als components als quals es vol poder accedir.
- **public void EnableRagdoll()**: Activa l'animació de mort, activant tots els *Collider*, desactivant el *IsKinematic* dels *Rigidbody* i desactivant el *Animator* i sistema de control del personatge.

- **public void DisableRagdoll()**: Desactiva l'animació de mort, desactivant tots els *_colliders*, activant el *IsKinematic* dels *_rigidbodies* i activant el *Animator* i sistema de control del personatge.

8.6.24 ToonShadingController

Component que serveix per a controlar un material amb el *Shader* utilitzat per a tots els models d'aquest projecte. Això es vol per a poder fer interpolacions d'aparició i desaparició dels models que tenen aplicats materials amb aquest *Shader*.

Atributs:

- **SkinnedMeshRenderer _skinnedMesh**: Referència al *SkinnedMeshRenderer* que te aplicat el material.
- **MeshRenderer _mesh**: Referència al *MeshRenderer* que té aplicat el material.
- **bool _isVerticalDissolveEnabled**: Indica si l'efecte de dissolució s'ha d'aplicar verticalment.
- **Material _material**: Referència al material controlat.
- **float _materialOriginalCullMode**: Valor que indica del mode original de renderitzat del material (doble cara o cara frontal).
- **IEnumerator _coroutine**: Referència al procés en execució actual (dissolució, materialització o nul).

Mètodes:

- **private void Awake()**: En instanciar-se, crea una referència al material el qual busca en el *_skinnedMesh* o *_mesh* segons si l'objecte és deformable o no. També guarda el mode de renderitzat (doble cara o cara frontal) original del material a *_materialOriginalCullMode*.
- **public void StartDissolve(float time = 1f, bool isVertical = false)**: Inicia la interpolació de desaparició del model amb els paràmetres indicats.
- **public void StartUndissolve(float time = 1f, bool isVertical = false)**: Inicia la interpolació d'aparició del model amb els paràmetres indicats.
- **IEnumerator Dissolve(float time = 1f, bool isVertical = false)**: Executa la desaparició del model en el temps amb els paràmetres indicats. En iniciar posa el mode de renderitzat a doble cara i en acabar el posa amb el valor de *_materialOriginalCullMode*.
- **IEnumerator Undissolve(float time = 1f, bool isVertical = false)**: Executa l'aparició del model en el temps amb els paràmetres indicats. En iniciar posa el mode de renderitzat a doble cara i en acabar el posa amb el valor de *_materialOriginalCullMode*.

8.6.25 Gun

Consisteix en una classe abstracta amb els atributs i mètodes que compartiran els diferents tipus d'armes de foc que s'implementin en el joc. La seva filosofia és la de simular una arma de foc real pel que fa a prémer i deixar de prémer el disparador, permeten comportaments diferents segons els paràmetres i tipus d'arma però oferint els mateixos mètodes d'interacció. També es caracteritza pels següents estats: esperant, disparant, recarregant, carregant i esperant disparar.

Atributs:

- **protected string _name**: Nom de l'arma.
- **protected GameObject _owner**: Referència al propietari.
- **protected Transform[] _shootingPoints**: Llista de punts per on pot disparar.
- **protected float _fireCooldownTime**: Temps d'espera abans de poder tornar a fer una comanda de disparar.
- **protected bool _automaticFire**:Indica si l'arma dispara automàticament en transcorra el temps de *_fireCooldownTime* i no haver donat una comanda de fi de disparar. En el cas d'una arma de càrrega índica si s'ha d'executar el tret automàticament en finalitzar la càrrega.
- **protected bool _canStopFire**: Indica si es pot atura una rafega de trets donant una comanda de fi de disparar.
- **protected float _fireRate**: Temps entre trets d'una rafega de trets.
- **protected float _chargeTime**: Temps de càrrega de l'arma. Un valor inferior a 0 indica que l'arma no és de càrrega.
- **protected float _reloadTime**: Temps de recarrega de l'arma. Un valor inferior a 0 indica que l'arma no és pot recarregar.
- **protected bool _autoReload**: Indica si l'arma és recàrrega automàticament en donar una comanda de tret sense munició a *_magazineAmmo*.
- **protected int _ammoMax**: Munició maxima amb la que es pot recarregar l'arma. Un valor inferior a 0 indica que es infinita.
- **protected int _magazineAmmoMax**: Munició màxima amb la qual es pot disparar abans de recarregar. Un valor inferior a 0 indica que l'arma no es recarrega i consumeix munició directament de *_ammo*.
- **protected int _ammo**: Munició que queda per recarregar.
- **protected int _magazineAmmo**: Munició que queda abans de recarregar.
- **protected int _shootingPointIndex**:Índex del punt de tret actual dins de *_shootingPoints*.
- **protected Timer _fireCooldownTimer**: Temporitzador del temps d'espera entre comandes de disparar.

- **protected Timer _chargeTimer**: Temporitzador del temps d'espera per a la finalització de la càrrega.
- **protected Timer _reloadTimer**: Temporitzador del temps d'espera per a la finalització de la recàrrega.
- **protected readonly Object _inputLock**: Objecte auxiliar per a regular el flux de comandes a l'arma.
- **protected GunState _state**: Estat actual de l'arma (*WAITING*, *FIRING*, *RELOADING*, *CHARGING*, *WAITING_FIRE*).
- **protected IEnumerator _coroutine**: Referència al procés en execució actual.
- **protected UnityAction _firingAdditionalAction**: Acció dinàmica que s'executa en realitzar un tret. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _endFiringAdditionalAction**: Acció dinàmica que s'executa en finalitzar el procés de disparar. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _reloadingAdditionalAction**: Acció dinàmica que s'executa en iniciar la recàrrega. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _endReloadingAdditionalAction**: Acció dinàmica que s'executa en finalitzar el procés de recàrrega. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _chargingAdditionalAction**: Acció dinàmica que s'executa en iniciar la càrrega. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _endChargingAdditionalAction**: Acció dinàmica que s'executa en finalitzar el procés de càrrega. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.
- **protected UnityAction _stopAdditionalAction**:Acció dinàmica que s'executa en avortar el procés actual. S'utilitza per a cridar l'animació corresponent del personatge que utilitza l'arma.

Mètodes:

- **private void Start()**: Abans de realitzar el primer *Update()* inicialitza l'arma.
- **virtual protected void Initialize()**: Inicialitza l'arma fent que la munició actual sigui igual a la màxima, inicialitzant els temporitzadors i posant el *_state* a *WAITING*.
- **virtual public void Trigger()**: Serveix per simular prémer un disparador d'una arma de foc (no hi ha implementació base).

- **virtual public void ReleaseTrigger()**: Serveix per simular deixar de prémer un disparador d'una arma de foc (no hi ha implementació base).
- **virtual public void Reload()**: Mètode sense implementar que serveix per simular la recarrega d'una arma de foc.
- **virtual public void CancelAction()**: Cancel·la el procés actual en *_coroutine*, invoca la funció dinàmica guardada en *_stopAdditionalAction* i posa el *_state* a *WAITING*.
- **virtual public void DirectFire(bool unlimited = true)**: Mètode sense implementar que serveix per a realitzar un sol tret. Serveix per a fer més senzilla la interacció de la IA dels personatges amb les armes.
- **public bool IsAmmoLeft()**: Indica si queda prou munició per disparar.

8.6.26 PatternGun

Component derivat de *Gun* per a armes que instancien patrons de bales.

Atributs:

- **BulletPattern _bulletPattern**: Referència al *Prefab* del patró de bales que l'arma instància en cada tret.
- **int _roundProjectiles**: Nombre de *_bulletPattern* que s'instanciaran per iteració en el procés de *Firing()*.

Mètodes:

- **override public void Trigger()**: Segons la configuració de l'arma i la munició actual si *_state* és *WAITING*, pot executar el procés *Firing()*, *Reloading()* o *Charging()*. Altrament si *_state* és *RELOADING* i la configuració de l'arma o permet, cancel·la aquest procés i executa *Charging()* o *Firing()*.
- **override public void ReleaseTrigger()**: Si *_state* és *FIRING* O *CHARGING* executa *CancelAction()*. Altrament si *_state* és *WAITING_FIRE* i queda munició executa el procés de *Firing()*.
- **override public void Reload()**: Segons la configuració de l'arma i la munició actual si *_state* és *WAITING* o *WAITING_FIRE* executa el procés de *Reloading()*. Si *_state* és *FIRING* i es pot aturar el foc, atura el procés actual i executa el procés de *Reloading()*. Altrament si *_state* és *CHARGING* atura el procés actual i executa el procés de *Reloading()*.
- **override public void DirectFire(bool unlimited = true)**: Executa *Fire* indicant-li si ha de tenir en compte la munició i s'invoca *_endFiringAdditionalAction*.
- **IEnumerator Firing()**: Procés en el temps en el qual es posa *_state* a *FIRING* i s'executa *Fire* fins a arribar *_roundProjectiles*. Al final s'invoca *_endFiringAdditionalAction* i es posa *_state* a *WAITING*.
- **public void Fire(bool unlimited = false)**: Realitza un únic tret sense importar la configuració de l'arma i s'invoca *_firingAdditionalAction*. Es pot indicar si

aquest tret te en compte la munició.

- **IEnumerator Reloading()**: Procés d'espera fins a assolir el *_reloadTime* en el que s'invoca *_reloadingAdditionalAction* i es posa *_state* a *RELOADING*. Un cop assolit l'arma queda carregada i s'invoca *_endReloadingAdditionalAction* i es posa *_state* a *WAITING*.

- **IEnumerator Charging()**: Procés d'espera fins a assolir el *_chargeTime* en el que s'invoca *_chargingAdditionalAction* i es posa *_state* a *CHARGING*. Un cop assolit, si es *_automaticFire* posa *_state* a *FIRING* i invoca *Fire*. Altrament s'invoca *_endReloadingAdditionalAction* i es posa *_state* a *WAITING_FIRE*.

8.6.27 BulletPattern

Consisteix en una classe que hereta de *ScriptableObject* i que indica com han de ser instanciades un seguit de bales en forma d'un conjunt de patrons radials i permet instanciar-les amb una sola crida.

Atributs:

- **BulletPatternComponent[] _bulletPatternComponents**: Llistat de totes les parts que componen el patró.

- **GameObject _owner**: Referència al propietari d'aquest patró. S'utilitza per vincular-li l'arma que l'ha instanciat.

- **Transform _origin**: Referència al punt en l'escena de *Unity* on s'origina el patró.

Mètodes:

- **public void Shoot()**: Instància cada element en *_bulletPatternComponents*.

- **void ShootComponent(BulletPatternComponent component)**: Instància la part del patró indicada segons els paràmetres que defineix.

8.6.28 BulletPatternComponent

Consisteix en una classe "serialitzable" que guarda la informació d'un patró distanciació radial de bales i que utilitza la classe *BulletPattern* per a instanciar-les.

Atributs:

- **public Bullet bulletPrefab**: Referència al *Prefab* de la bala a instanciar.

- **public bool randomBulletSpread**: Indica si les bales s'instancien de manera aleatòria per l'angle indicat.

- **public float bulletSpeed**: Velocitat de les bales.

- **public int nBullets**: Nombre de bales a instanciar.

- **public float spawnAngle**:Angle s'obre el que s'instancien les bales.

- **public float angleDirection**: Direcció en la qual apunta el centre de l'angle.
- **public float distanceToCenter**: Distància al centre en la que s'instancien les bales.
- **public float delay**: Temps de retard abans d'instanciar.

8.6.29 Bullet

Component que serveix per a fer que un objecte es desplaci cap endavant fins que sigui destruït en tocar un obstacle. Aquest component està preparat per tal que, en ser instanciat un primer cop, pugui ser reutilitzat mitjançant els mètodes que ofereix sense necessitat de ser instanciat repetidament. Això es fa per qüestions de rendiment, ja que les bales per naturalesa haurien de ser instanciades constantment.

Atributs:

- **string _bulletTypeID**: Identificador del tipus de bala.
- **int _damage**: Valor que resta als objectes, que implementin *IDamageable*, tocats.
- **float _selfDestructionTime**: Temps que triga a finalitzar la bala sense haver impactat amb objecte.
- **float _speed**: Velocitat de la bala.
- **GameObject _owner**: Referència a qui ha instanciat la bala.
- **Rigidbody _rigidbody**: Referencia al *Rigidbody*.
- **bool _isEnded**: Indica si la bala ha finalitzat.

Mètodes:

- **private void Awake()**: En instanciar-se, crea la referencia al *Rigidbody*, indica que la bala ha finalitzat i desactiva l'objecte.
- **void OnTriggerEnter(Collider other)**: En col·lisionar amb un objecte, si aquest té el *Tag* de *Character* es comprova que el personatge que ha provocat l'instanciació d'aquesta bala no sigui el mateix objecte tocat. Llavors s'accedeix al component *IDamageable* de l'objecte collisionat per tal d'aplicar-li el mal indicat per *_damage* i finalitzar la bala.
- **void SelfDestruct()**: Mètode que és invocat en acabar el temps de *_selfDestructionTime* i el qual executa *End()*.
- **public void Setup(Vector3 position, Quaternion rotation, float speed, GameObject owner = null, float delay = 0)**: Actualitza la bala amb els paràmetres indicats i l'inicialitza segons el retard indicat.
- **void Initialize()**: Inicialitza la bala, activant el *IsKinematic* de *_rigidbody* i aplicant-li la *_speed* com a una velocitat. També posa a fals *_isEnded* i prepara la invocació amb el retard indicat per *_selfDestructionTime* de *SelfDestruct()*.

- **public void End()**: Finalitza la bala, desactivant el *IsKinematic* de *_rigidbody*. També cancel·la la invocació amb retard de *SelfDestruct()* i posa *_isEnded* a cert.

8.6.30 CharacterAbility

Consisteix en una classe abstracta amb els atributs i mètodes que compartiran els diferents tipus d'habilitats que s'implementin en el joc. Una habilitat només pot ser utilitzada per un personatge, el qual es el seu invocador, i la seva filosofia és la de controlar-la a través de dos mètodes *Trigger()* i *ReleaseTrigger()*, permeten comportaments diferents segons els paràmetres i tipus d'habilitat però oferint els mateixos mètodes d'interacció. També es caracteritza pels següents estats: a punt, activa, en temps d'espera.

Atributs:

- **protected string _name**: Nom de l'habilitat
- **protected Character _caster**: Personatge que utilitza l'habilitat.
- **protected float _cooldownTime**: Temps d'espera entre usos de l'habilitat.
- **protected readonly Object _inputLock**: Objecte auxiliar per a regular el flux de comandes a l'habilitat.
- **protected Timer _cooldownTimer**: Temporitzador del temps d'espera per a reutilitzar l'habilitat.
- **protected IEnumerator _coroutine**: Referència al procés en execució actual.
- **protected AbilityUseState _abilityUseState**: Estat actual de l'habilitat (*READY, ACTIVE, COOLDOWN*).

Mètodes:

- **private void Start()**: Abans de realitzar el primer *Update()* inicialitza l'habilitat.
- **virtual protected void Initialize()**: Inicialitza el temporitzador *_cooldownTimer* amb *_cooldownTime*.
- **public virtual void Trigger()**: És el mètode que s'executa en prémer el botó de l'habilitat (no hi ha implementació base).
- **public virtual void ReleaseTrigger()**: És el mètode que s'executa en deixar de prémer el botó de l'habilitat (no hi ha implementació base).
- **public virtual void CancelAction()**: Cancel·la el procés actual en *_coroutine*, fa que *_caster* executi l'animació d'"idle" i posa el *_state* a *READY*.

8.6.31 ForwardTeleportAbility

Component derivat de *CharacterAbility* que consisteix en un teletransport cap endavant del personatge que la utilitza. Consisteix en dues fases, una primera fase en la qual el personatge canalitza l'habilitat incrementant el seu rang en el temps i podent-se moure i apuntar, i una segona fase en la qual el personatge decideix començar la

teleportació. Es caracteritza per uns estats que són completament independents dels de la classe pare i el seu funcionament és únicament intern: esperant, canalitzant i teleportant.

Atributs:

- **GameObject _destinationIndicator**: Referència a l'objecte que fa d'indicador en l'escena per a veure a on es tele-transportarà el personatge.
- **float _maxRange**: Distància màxima de teletransport.
- **float _rangeIncreaseSpeed**: Velocitat en la qual creix el rang de teletransport.
- **float _vanishTime**: Duració de l'animació de desaparició del personatge.
- **float _materializeTime**: Duració de l'animació d'animació del personatge.
- **LayerMask _collisionMask**: *Tags* dels objectes que no es poden sobrepassar en el teletransport.
- **Vector3 _destination**: Posició destí calculada en l'escena de *Unity*.
- **Timer _vanishTimer**: Temporitzador del temps de desaparició del personatge.
- **Timer _materializeTimer**: Temporitzador del temps d'aparició del personatge.
- **State _state**: Estat actual de l'habilitat (*WAITING*, *CHANNELING*, *TELEPORTING*). Aquest estat és intern i independent a la classe pare

Mètodes:

- **void Start()**: Abans de realitzar el primer *Update()* inicialitza l'habilitat.
- **override protected void Initialize()**: Extensió del mètode *Initialize()* de *CharacterAbility* en la que s'inicialitzen els temporitzadors, es posa *_state* a *WAITING* i es prepara l'indicador de destinació del personatge.
- **public override void Trigger()**: Si *_state* és *WAITING* i el temporitzador de *_cooldonwTimer* ha finalitzat, posa *_state* a *CHANNELING*, *_abilityUseState* a *ACTIVE* i executa el procés de *Channeling()*.
- **public override void ReleaseTrigger()**: Si *_state* és *CHANNELING*, atura el procés de *Channeling()*, posa *_state* a *TELEPORTING*, *_abilityUseState* a *ACTIVE* i executa el procés de *Teleporting()*.
- **public override void CancelAction()**: Extensió del mètode *CancelAction()* de *CharacterAbility* en la que si *_state* és *CHANNELING*, es desactiva *_destinationIndicator* i s'habilita l'ús d'atac, habilitat i interacció del personatge. Altrament, si *_state* és *TELEPORTING* es desactiva *_destinationIndicator* i s'habiliten els controls del personatge. Finalment, es posa *_state* a *WAITING*.
- **IEnumerator Channeling()**: Procés en el temps en el qual s'activa *_destinationIndicator*, es deshabilita la capacitat d'atacar, utilitzar altres habilitats i interactuar del personatge, s'executa l'animació de canalització del personatge i es va

incrementant el rang de teletransport en la direcció en la qual s'apunta fins a arribar al màxim, actualitzan *_destination*. El procés perdura fins que aquest finalitza per la seva cancel·lació o la confirmació de teletransport.

- **IEnumerator Teleporting()**: Procés en el temps en el qual es desactiva *_destinationIndicator*, es deshabiliten els controls del personatge i es comença el teletransport del personatge fent-lo desaparèixer en el temps indicat per *_vanishTime*. Un cop finalitzat aquest temps, es trasllada el personatge a *_destination*, es retornen els controls del personatge, i s'inicia l'animació d'aparició durant el temps indicat per *_materializeTime*. Finalitzat l'ús de l'habilitat entra en temps d'espera i es posen *_state* a *WAITING* i *_abilityUseState* a *COOLDOWN*.

8.6.32 WeaponSelectorUI

Component que serveix per mostrar la selecció de l'arma d'un personatge amb els seus estats en la interfície de *Unity*. Ofereix mètodes per tal d'actualitzar la informació quan aquesta canvia. S'utilitza per a mostrar els estats de l'arma seleccionada pel personatge controlat pel jugador.

Atributs:

- **TextMeshProUGUI _defaultWeaponName**: Referencia al text en la interfície on es mostra el nom de l'arma per defecte.
- **TextMeshProUGUI _primaryWeaponName**: Referencia al text en la interfície on es mostra el nom de l'arma primària.
- **TextMeshProUGUI _secondaryWeaponName**: Referencia al text en la interfície on es mostra el nom de l'arma secundària.
- **GameObject _defaultSlot**: Referència al panell en la interfície per a l'arma per defecte.
- **GameObject _primarySlot**: Referència al panell en la interfície per a l'arma primària.
- **GameObject _secondarySlot**: Referència al panell en la interfície per a l'arma secundària.
- **TextMeshProUGUI _ammoMax**: Referencia al text en la interfície on es mostra la munició maxima de l'arma seleccionada.
- **TextMeshProUGUI _magazineAmmoMax**: Referència al text en la interfície on es mostra la munició màxima del cartutx de l'arma seleccionada.
- **TextMeshProUGUI _ammo**: Referencia al text en la interfície on es mostra la munició de l'arma seleccionada.
- **TextMeshProUGUI _magazineAmmo**: Referència al text en la interfície on es mostra la munició carregada en el cartutx de l'arma seleccionada.
- **TextMeshProUGUI _magazineSeparator**: Referència al text que fa de separació en la munició.

- **TextMeshProUGUI _storageSeparator**: Referència al text que fa de separació en la munició de cartutx.
- **Gun _weapon**: Referència a l'arma seleccionada.
- **Gun _defaultWeapon**: Referència a l'arma per defecte.
- **Gun _primaryWeapon**: Referència a l'arma primària.
- **Gun _secondaryWeapon**: Referència a l'arma secundària.

Mètodes:

- **public void ActualizeStats()**: Actualitza els estats a mostrar segons l'arma seleccionada.
- **public void SelectDefaultWeapon()**: Mostra el panell de l'arma per defecte amb els seus estats actualitzats.
- **public void SelectPrimaryWeapon()**: Mostra el panell de l'arma primària amb els seus estats actualitzats.
- **public void SelectSecondaryWeapon()**: Mostra el panell de l'arma secundària amb els seus estats actualitzats.

8.6.33 StatBarUI

Component que serveix per a mostrar un valor actual sobre un valor màxim en format de barra en la interfície de *Unity*. Ofereix un mètode per a actualitzar-la amb els valors desitjats. S'utilitza per a mostrar la barra de vida del personatge controlat pel jugador.

Atributs:

- **TextMeshProUGUI _label**: Referència al text en la interfície on es mostra l'etiqueta que dona nom a la barra.
- **TextMeshProUGUI _current**: Referència al text en la interfície on es mostra el valor actual de la barra.
- **TextMeshProUGUI _max**: Referència al text en la interfície on es mostra el valor màxim de la barra.
- **Image _fill**: Referència a la imatge que s'utilitza per a mostrar el percentatge del valor de *_current* sobre *_max* de manera visual en forma de barra.

Mètodes:

- **public void ActualizeStats(string label, int current, int max)**: Actualitza els valors de la barra amb els valors indicats.

8.6.34 StatUI

Component que serveix per a mostrar un valor actual sobre un valor màxim en la interfície de *Unity*. Ofereix un mètode per tal d'actualitzar-la amb els valors desitjats. S'utilitza per a mostrar el nivell d'accés del personatge controlat pel jugador sobre el nivell d'accés màxim del pis generat.

Atributs:

- **TextMeshProUGUI _label**: Referència al text en la interfície on es mostra l'etiqueta que dona nom a l'estat.
- **TextMeshProUGUI _current**: Referència al text en la interfície on es mostra el valor actual de l'estat.
- **TextMeshProUGUI _max**: Referència al text en la interfície on es mostra el valor màxim de l'estat.

Mètodes:

- **public void ActualizeStats(string label, int current, int max)**: Actualitza els valors de l'estat amb els valors indicats.

8.6.35 Menu

Component que serveix per a enllaçar botons de la interfície de *Unity*, dins la jerarquia de l'objecte que conté aquests component, a un seguit de funcions del *GameManager* per així gestionar la navegació entre escenes del joc.

Atributs:

- **TextMeshProUGUI _title**: Referència al text en la interfície on es mostra el títol del menú.

Mètodes:

- **void Start()**: Abans de realitzar el primer *Update()* selecciona el primer botó del menú.
- **private void OnEnable()**: Quan s'activa l'objecte que conté aquest component selecciona el primer botó del menú.
- **public void ChangeTitleText(string text, Color color)**: Canvia el text i color mostrat per *_title* segons els paràmetres indicats.
- **public void ExitGame()**: Executa el mètode *ExitGame()* de *GameManager*.
- **public void StartGame**: Executa el mètode *StartGame()* de *GameManager*.
- **public void GoMainMenu**: Executa el mètode *GoMainMenu()* de *GameManager*.

8.6.36 LoadingScreen

Component que serveix per a gestionar una pantalla estàtica que permet tapar l'escena actual de Unity. Això es vol per a poder deixar de mostrar l'escena de *Unity* quan aquest es troba realitzant un procés de càrrega o de construcció del nivell i només tornar a mostrar-la quan aquest procés hagi finalitzat.

Atributs:

- **float _fadeInDuration**: Duració de l'animació d'aparició de la pantalla.
- **float _fadeOutDuration**: Duració de l'animació de desaparició de la pantalla.
- **CanvasGroup _canvasGroup**: Referència al *CanvasGroup* que permet modificar la transparència per fer que la pantalla aparegui i desaparegui.

Mètodes:

- **public void Awake()**: En instanciar-se crea la referència amb el *CanvasGroup* i li posa l'opacitat a 0 per tal que la pantalla no sigui visible.
- **public IEnumerator FadeIn()**: Inicia una interpolació d'aparició de la pantalla en el temps indicat per *_fadeInDuration*.
- **public IEnumerator FadeOut**: Inicia una interpolació de desaparició de la pantalla en el temps indicat per *_fadeInDuration*.

8.6.37 PoolingManager

Consisteix en un component que converteix l'objecte en un *Singleton* que perdura entre escenes de *Unity*. Aquest serveix per a gestionar la instanciació d'elements que d'altra manera serien destruïts i creats constantment. Això ho fa utilitzant contenidors del tipus *ObjecPool*, els quals permeten obtenir instàncies que s'han deixat de fer servir (alliberades) i els quals només instanciaran noves instàncies quan totes les instàncies actuals es trobin en ús. Aquesta classe es fa servir, en comptes dels mètodes *Instantiate*, quan es vol instanciar una bala o un enemic en el joc.

Atributs:

- **private static PoolingManager _instance**: Referència a la instància d'aquest objecte.
- **public static PoolingManager Instance**: Referència pública a la instància d'aquest objecte.
- **static Dictionary<string, ObjectPool<Bullet» _bulletPoolDictionary**: Diccionari que associa els noms de les bales amb un contenidor *ObjectPool* amb bales que tinguin aquest nom.
- **static Dictionary<string, ObjectPool<NavMeshCharacter» _navMeshCharacterPoolDictionary**: Diccionari que associa els noms dels personatges controlats per la IA amb un contenidor *ObjectPool* amb *NavMeshCharacter* que tinguin aquest nom.

Mètodes:

- **private void Awake()**: En instanciar-se s'assegura que només existeixi una instància de l'objecte que conté aquest component i que aquest no es destrueixi en carregar una nova escena.
- **public Bullet GetBullet(Bullet prefab)**: Retorna una bala instanciada en l'escena de *Unity* que sigui instància del *Prefab* indicat. Si no existia prèviament cap instància del *Prefab* indicat, crea un contenidor per a ella en *_bulletPoolDictionary*.
- **public void ReleaseBullet(Bullet instance)**: Allibera la bala instanciada indicada en l'escena de *Unity*. Si no existia prèviament cap instància del *Prefab* d'aquesta bala, crea un contenidor per a ella en *_bulletPoolDictionary*.
- **public ObjectPool<Bullet> CreateBulletPool(Bullet prefab)**: Crea un *ObjectPool* per al *Prefab* de bala indicat i l'afegeix a *_bulletPoolDictionary*.
- **public NavMeshCharacter GetNavMeshCharacter(NavMeshCharacter prefab)**: Retorna una personatge controlat per la IA instanciat en l'escena de *Unity* que sigui instància del *Prefab* indicat. Si no existia prèviament cap instància del *Prefab* indicat, crea un contenidor per a ell en *_navMeshCharacterPoolDictionary*.
- **public void ReleaseCharacter(NavMeshCharacter instance)**: Allibera el personatge controlat per la IA instanciat instanciat indicat en l'escena de *Unity*. Si no existia prèviament cap instància del *Prefab* d'aquest personatge, crea un contenidor per a ell en *_bulletPoolDictionary*.
- **public ObjectPool<NavMeshCharacter> CreateNavMeshCharacterPool (NavMeshCharacter prefab)**: Crea un *ObjectPool* per al *Prefab* del personatge controlat per la IA indicat i l'afegeix a *_navMeshCharacterPoolDictionary*.
- **public void DestroyAll()**: Destrueix totes les instàncies que aquesta classe ha instanciat i reinicia els diccionaris.
- **void DestroyAllChildrenObjects()**: Destrueix totes les instàncies que aquesta classe ha instanciat.

8.6.38 LevelManager

Consisteix en un component que converteix l'objecte en un *Singleton* que perdura entre escenes de *Unity*. Aquest serveix per a iniciar la generació del pis i per a mantenir l'estat del joc sobre ell (com el jugador va visitant les habitacions que el componen).

Atributs:

- **private static LevelManager _instance**: Referència a la instància d'aquest objecte.
- **public static LevelManager Instance**: Referència pública a la instància d'aquest objecte.
- **TextAsset _floorConfigurationJSON**: Referència al fitxer en formar JSON que defineix com ha de ser construït el pis.

- **GameObject _playerPrefab**: Referencia al *Prefab* a instanciar que consisteix en el personatge controlar per el jugador durant la partida.
- **GameObject[] _enemyPrefabs**: Referència als *Prefabs* a instanciar com a enemics.
- **ControllerCharacter _player**: Referència al personatge controlar pel jugador en l'escena de *Unity*.
- **Floor _floor**: Referència al pis generat.
- **Room _previousRoom**: Referència a l'ultima habitació visitada pel jugador.
- **Room _currentRoom**: Referència a l'habitació en es troba el jugador actualment.
- **List<NavMeshCharacter> _spawnableEnemies**: Llista amb els *Prefabs* dels enemics instanciables en el pis.
- **int _nRoomsCleared**: Nombre d'habitacions superades (habitacions en les quals s'ha eliminat tots els enemics).

Mètodes:

- **void AwakeAdditional()**: Abans del primer *Update()* inicia la generació del pis.
- **void Start()**: Assigna a *_player* el nivell d'accés de l'habitació actual.
- **void CleanLevel()**: Desactiva el *_player* i el destrueix, també indica al *PoolingManager* que ha de destruir totes les instàncies que ha creat (enemics i bales) i destrueix tots els elements del pis.
- **void GenerateLevel()**: Executa *CleanLevel()* si el pis ja estava generat, construeix un pis amb el fitxer *_floorConfigurationJSON*, prepara *_spawnableEnemies* que poden ser instanciats i instància el personatge del jugador.
- **void SpawnPlayer()**: Instància el jugador en la primera sala del pis i reinicia el nombre d'habitacions superades.
- **void SpawnEnemiesInCurrentRoom()**: Indica a l'habitació actual que instància de manera pseudoaleatòria un conjunt d'enemics. Per a l'aleatorietat es fa servir una funció que té tendència a posar més enemics com més sales hagin estat superades.
- **public void TriggerRoomTransition()**: Indica que el jugador ha canviat d'-habitació. Si la nova habitació no havia estat superada, s'executa el mètode *SpawnEnemiesInCurrentRoom()* i es bloquegen les portes de l'habitació perquè el jugador no pugui sortir fins que aquesta hagi estat superada.
- **public void NotifyCurrentRoomCleared()**: Incrementa el nombre d'habitacions superades i actualitza el nivell d'accés del jugador.

- **public void UpdateAccesLevel()**: Assigna el nivell d'accés de l'habitació al nivell d'accés del jugador, si aquest tenia un nivell d'accés inferior.

8.6.39 GameManager

Consisteix en un component que converteix l'objecte en un *Singleton* que perdura entre escenes de *Unity*. Aquesta serveix per a controlar la navegació entre escenes dins del joc.

Atributs:

- **private static GameManager _instance**: Referència a la instància d'aquest objecte.
- **public static GameManager Instance**: Referència pública a la instància d'aquest objecte.
- **const string GAME_SCENE_NAME**: Constant que manté el nom de l'escena on es juga la partida.
- **const string MAIN_MENU_SCENE_NAME**: Constant que manté el nom de l'escena del menú principal.
- **LoadingScreen _loadingScreen**: Referència a la pantalla de carrega.
- **Menu _pauseMenu**: Referència al menú de pausa.
- **Menu _gameEndMenu**: Referència al menú de fi de partida.

Mètodes:

- **void Start()**: Inicia la generació de la *Seed* utilitzada per a l'aleatorietat de la partida.
- **IEnumerator LoadSceneAsyncCoroutine(string sceneName)**: Procés en el qual es carrega l'escena indicada mostrant una pantalla de càrrega. Al final d'aquest procés la pantalla de càrrega s'oculta.
- **void GenerateSeed()**: GGenera una *Seed* amb el temps actual del sistema i l'assigna al sistema d'aleatorietat *Random.InitState*.
- **public void PauseGame()**: Pausa la partida i mostra el menú de pausa.
- **public void UnPauseGame()**: Reprèn la partida i oculta el menú de pausa.
- **public void EndGame(bool victory = false)**: Mostra el menú de derrota o victòria en funció del valor indicat.
- **public void ResumeGame()**: Executa *UnPuaseGame()*.
- **public void ExitGame()**: Atura l'execució del joc.
- **public void StartGame()**: Executa *UnPuaseGame()* i inicia la càrrega de l'escena on es du a terme la partida.

- **public void GoMainMenu()**: Executa *UnPuaseGame()* i inicia la càrrega de l'escena del menú principal.

Capítol 9

Implementació i proves

9.1 Generació del pis

Un pis es genera a través de les indicacions d'un fitxer *JSON* del qual podem distingir-ne 3 parts: la informació general, la informació de les habitacions que el componen i les restriccions sobre com han de col·locar-se. Havent carregat aquesta informació, podem començar amb la generació del pis, la qual a gran escala està dividida de la següent manera:

```
algoritme Generacio_pis()
    informacio = Llegir_Fitxer(fitxerJSON)
    pis_estructura =
    CrearMatriu<Habitacio>(informacio.moduls_per_fila,
    informacio.moduls_per_columna)
    * informacio.mida_modul)

    si informacio.generacio_aleatoria
        Colocar_habitacions_principals()
        Comprovar_cami_habitacions_principals(correcio=cert,
    posicions_nules_valides=cert)
        Connectar_habitacions()
        Colocar_habitacions_opcionals()
    altrament
        Generacio_no_aleatoria()
    fsi

    Colocar_portes()
    Generar_interior_habitacions()

    Instanciar_pis()
falgoritme
```

Seguidament, veurem amb més detall les diferents parts de l'algoritme presentat:

9.1.1 Execució aleatòria

És la secció més crítica de la generació del pis, ja que el pes de crear un pis solucionable, recau completament en els algoritmes. Quan parlem d'un pis solucionable, ens referim que totes les habitacions que han estat marcades com a obligatòries han d'aparèixer en el joc, han de poder ser accedides en l'ordre indicat i també ha d'existir una única entrada i com a mínim una sortida.

Col·locació de les habitacions principals

Aquesta etapa s'encarrega de col·locar totes les habitacions no opcionals descrites en la informació. Pel que fa a la col·locació de les habitacions mínimes, en un primer moment intenta col·locar-les de la manera d'escrita per la restricció, però si no ho aconsegueix les col·loca allà on pugui, inclús si ha de substituir-la per una habitació d'una sola posició en l'estructura que comuniqui amb l'habitació corresponent a la restricció (externalització de l'habitació). Si això no es pot realitzar, a causa d'haver definit més habitacions que posicions en l'estructura, el programa avorta.

La col·locació de les habitacions addicionals es duu a terme com a intents amb una probabilitat determinada, sense importar si aquestes arriben a ser col·locades o no.

```
algoritme Colocacio_habitacions_principals()
    per cada restriccio en informacio.restriccions
        si restriccio.tipus_habitacio != "opcional"
            per i=0 fins i=restriccio.habitacions_minimes - 1 pas 1
                si no Colocar_habitacio(restriccio)
                    error
                fsi
            fper

            per i=0 fins i=restriccio.habitacions_addicionals
                si Aleatorietat(restriccio.probabilitat)
                    Colocar_habitacio(restriccio)
                fsi
            fper
        fsi
    fper
falgoritme
```

Comprovació i correcció de camins entre les habitacions principals

Un cop havent col·locat les sales que constitueixen les parts principals del joc, és imprescindible assegurar que aquestes no han quedat col·locades generant barreres de nivell que impedeixin traçar un camí que sigui capaç de visitar totes les sales per ordre. Aquesta etapa traça camins mínims entre l'habitació d'entrada i la resta d'habitacions amb un nivell d'accés determinat per la iteració actual. Les habitacions a les quals no s'ha pogut accedir, és perquè existeix algun bloqueig de nivell, el qual es corregeix traçant un camí mínim cap a elles i externalitzant les habitacions que generen el bloqueig. Aquest camí mínim es calcula amb *A** mitjançant la classe *AStar* anteriorment descrita.

```
algoritme Comprovar_cami_habitacions_principals(correcio,
    posicions_nules_valides)
     habitacio_inicial = Obtenir_habitacio_entrada(pis_estructura)
     nivells_habitacions =
    Obtenir_habitacions_per_nivell(pis_estructura)
     nivells_habitacions_pendents = Crear_diccionari<Nombre,
    Habitacio>()

     per cada nivell_habitacions en nivells_habitacions
         per cada habitacio_desti en nivell_habitacions.habitacions
             cami = Cami_minim(habitacio_inicial, habitacio_desti,
    nivell_habitacions.nivell, posicions_nules_valides)
             si no cami.existeix
                 si correcio

    nivell_habitacions_pendents.Afegir(nivell_habitacions.nivell,
    habitacio_desti)
                 altrament
                     retorna fals
                 fsi
             fsi
         fper
     fper

     si correcio
         per cada nivell_habitacions en nivells_habitacions_pendents
             per cada habitacio_desti en
    nivell_habitacions.habitacions
                 cami = Cami_minim(habitacio_inicial,
    habitacio_desti, nivells_habitacions_pendents.nivell_minim,
    posicions_nules_valides)
                 per cada node en cami
                     si node.habitacio.nivell >
    nivell_habitacions.nivell
                         Externalitzar_habitacio(node.habitacio,
    habitacio_simple, node.posicio)
                     fsi
                 fper
             fper
         fper
     fsi

    retorna cert
falgoritme
```

Connexió d'habitacions principals

Aquest procés consisteix en una iteració per nivell d'accés, que connecta cada habitació d'aquest nivell amb cada habitació del superior. Per a fer això, es generen camins d'habitacions entre l'habitació actual fins a l'habitació destí. Aquest camí es construeix avançant aleatòriament cap amunt, esquerra o dreta amb una influència més gran cap a la direcció de l'habitació destí. Quan s'avança a una posició sense habitació es col·loca una habitació aleatòria vàlida d'una llista d'habitacions per aquest propòsit definides en la informació. Aquest procés es fa fins a arribar a l'habitació destí o fins a excedir cert nombre de passos (per reduir el temps d'execució). Amb

aquests passos excedits es traça un camí mínim d'habitacions entre la posició actual i la primera posició de l'habitació destí.

Finalment, es realitzen les connexions de les habitacions definides com a desordenades en la informació. Aquestes simplement es connecten amb un camí d'habitacions, de la mateixa manera que la descrita anteriorment, cap a una altra habitació principal no desordenada.

```
algoritme Connectar_habitacions()
    nivells_habitacions = Obtenir_habitacions_per_nivell(pis_estructura)
    habitacions_previes = Sostreure_nivell_1(nivells_habitacions)
    habitacions_desordenades = Sostreure_nivell_0(nivells_habitacions)

    per cada nivell_habitacions en nivells_havitacions
        per cada habitacio_previa en habitacions_previes
            per cada habitacio_desti en nivell_habitacions.habitacions
                Generar_cami_habitacions(havitacio_previa, habitacio_desti)
            fper
        fper
        habitacions_previes = nivell_habitacions.habitacions
    fper

    per cada habitacio_desordenada en habitacions_desordenades
        habitacio_desti = Obtenir_habitacio_aleatoria_ordenada(pis_estructura)
        Generar_cami_habitacions(havitacio_previa, habitacio_desti)
    fper
falgoritme
```

Col·locació d'habitacions opcionals

Les habitacions opcionals no necessiten poder ser accedides en cap ordre particular, així que només cal col·locar-les de manera molt similar a la col·locació d'habitacions principals, però de manera que aquestes quedin adjacents a una altra habitació prèviament existent.

```
algoritme Colocar_habitacions_opcionals()
    per cada restriccio en informacio.restriccions
        si restriccio.tipus_habitacio == "opcional"
            per i=0 fins i=restriccio.habitacions_minimes - 1 pas 1
                Colocar_habitacio(restriccio, adjacent=cert)
            fper

            per i=0 fins i=restriccio.habitacions_addicionals - 1 pas 1
                si Aleatorietat(restriccio.probabilitat)
                    Colocar_habitacio(restriccio, adjacent=cert)
                fsi
            fper
        fsi
    fper
falgoritme
```

9.1.2 Execució no aleatòria

La idea de l'execució no aleatòria consisteix en simplement col·locar de manera exacta cada una de les sales com si fos la col·locació de sales principals, però avorten l'execució si la informació de posicionament no és possible. Un cop fet això només cal fer la comprovació de camí entre sales anteriorment descrita, però sense permetre el moviment per posicions nul·les i sense aplicar la correcció, avortant el programa en el cas que aquesta no s'hagi pogut realitzar.

9.1.3 Col·locació de portes

Amb tot el pis muntat de manera estructural, ara només cal assignar a cada habitació les posicions de les portes, les quals s'obtenen mirant les adjacències entre sales i per cada posició de la sala en l'estructura (mòdul) adjacent a un altre sala, indicar la direcció en la qual es connecten.

```
algoritme Colocar_portes()
    per cada habitacio en pis_estructura
        habitacio.Assignar_posicions_portes
    (Obenir_posicions_portes(habitacio))
    fper
falgoritme
```

9.1.4 Generació interna de les habitacions

Com a últim pas, de la generació lògica del pis, queda crear l'interior de cada sala i, per tant, l'interior del pis. Això es fa ordenant a les sales, que ara també coneixen la ubicació de les seves portes, que determinin els blocs que constitueixen el seu interior. Cal també indicar que es generi l'interior de totes aquelles habitacions externalitzades que han quedat fora de l'estructura del pis i només poden ser accedides mitjançant els tele-transportadors.

```
algoritme Generar_interior_habitacions()
    pis_interior =
    CrearMatriu<Habitacio>(informacio.moduls_per_fila *
    informacio.mida_modul, informacio.moduls_per_columna *
    informacio.mida_modul)

    per cada habitacio en pis_estructura
        habitacio.Generar_interior()
        si habitacio.habitacio_vinculada != nula
            habitacio.habitacio_vinculada.Generar_interior()
        fsi
    fper
falgoritme
```

En més detall la generació de l'interior de la sala consisteix a iterar per cada posició de cada mòdul no nul que la compon i empilar els blocs segons la posició que ocupen, de manera que totes les posicions contindran un terra, els límits també contindran una paret o porta (segons si hi ha accés amb una altra habitació) i el centre del primer mòdul contindrà el tele-transportador cap a la sala vinculada, si aquesta en tenia. Addicionalment, si la sala és una sortida, es col·loca un tele-transportador de nivell de manera aleatòria sobre una posició amb només terra.

```
algoritme Generar_interior_habitacio()
    habitacio_interior = Matriu<Cela>()

    /*Moduls*/
    per i=0 fins habitacio_estructura.files - 1 pas 1
        per j=0 fins habitacio_estructura.columnes - 1 pas 1
            si habitacio_estructura[i, j] != nul

                /*Interior de modul*/
                per k=0 fins mida_modul - 1 pas 1
                    per l=0 mida_modul - 1 pas 1
                        fila_interior = i * mida_modul + k
                        columna_interior = j * mida_modul + l

                        cela_actual =
    habitacio_interior[fila_interior, columna_interior]

                        cela_actual.afegir("terra")
                        si Es_limit_modul(cela_actual.posicio)
                            si
    posicions_portes.Conte(cela_actual.posicio)
                                cela_actual.afegir("porta")
                            altrament
                                cela_actual.Afegir("paret")
                            fsi
                        altrament si habitacio_vinculada != nul /\
    Es_centre_primer_modul(cela_actual.posicio)

    cela_actual.Afegir("tele-transportador")
                        fsi

                    fper
                fper
            fsi
        fper
    fper

    si tipus_habitacio == "sortida"
        posicio = Obtenir_posicio_transitable_aleatoria()
        habitacio_interior[posicio.x,
    posicio.y].Afegir("tele-transportador-nivell")
    fsi
falgoritme
```

9.1.5 Instanciació del pis

Amb el pis generat internament de manera lògica, només cal instanciar cada habitació que el conforma. Això es fa instanciant cada bloc de les piles (cel·les) guardades en cada una de les seves posicions seguint un ordre *FIFO*.

9.2 Personatge

Els personatges s'han implementat com una composició de components originals de *Unity* i propis amb un component gestor derivat de *Character*, *ControllerCharacter* o

NavMeshCharacter, que s'encarrega de mantenir el seu estat i de servir de interfície per a controlar tota la seva lògica d'accions. Una gestió important que aquest component realitza es l'associació de les animacions del personatge segons l'arma i acció utilitzada. Aixo es du a terme assignant les animacions que s'han de fer servir a accions anònimes, associades a un proces de l'arma (disparar, recarregar, carregar i mantenir preparat un dispar).

9.2.1 Jugador

El personatge controlat pel jugador es caracteritza per moure's a través de moviments indicats pels controls que faci servir, és per això que utilitza la implementació de *Character* amb *ControllerCharacter*. Això es produeix a través d'una cadena on *PlayerInput* capta els "inputs", *PlayerCharacterInputController* els captura, els processa i els passa a *ContollerCharacter*, i finalment *ControllerCharacter* segons l'accio i l'estat, les solicita al *CharacterController*, la *CharacterHability* o la *Weapon* seleccionada.

9.2.2 Enemics

Els personatges controlats per les IAs es caracteritzen per moure's a través d'un *NavMesh*, essent indispensable per a ells que siguin col·locats sobre un i que addicionalment tinguin un component de tipus*NavMeshAgent* que defineixi les seves característiques sobre aquest. És per això que aquests fan servir la implementació de *Character* de *NavMeshCharacter*, ja que aquesta permet controlar-los oferint mètodes de control més adequats perquè la IA pugui treballar amb ells.

És també important destacar que aquests, per a poder detectar el seu entorn tenen un component de tipus *FieldOfView* el qual els proporciona un con de visió de rang determinat i és la manera que tenen de poder detectar el jugador.

Una altra característica rellevant és el component *BehaviourTreeRunner* el qual executa l'arbre de comportament que en determina les seves accions. És important destacar que alguns nodes que conformen aquest arbre treballaran amb les habitacions per tal que aquestes els hi proporcionin posicions del *NavMesh* de l'habitació que ocupen, per tal que així puguin indicar al *NavMeshAgent* quin és el seu destí. Per simplicitat en la programació de la IA aquests s'han simplificat perquè no facin servir habilitats i utilitzin les armes de manera directa, únicament disparant quan així ho indiquin. Seguidament veurem l'implementació de la IA amb mes detall:

Clon IA

El comportament d'aquest enemic, s'implementa amb un "behaviour tree" dividit en 3 comportaments:

- Patrulla: Va demanant posicions aleatòries al *NavMesh* de l'habitació i s'hi desplaça utilitzant el *NavMeshAgent*, fent parades de duració aleatòria en arribar al destí. Paral·lelament, va comprovant si el seu *FieldOfView* detecta el jugador, finalitzant aquest comportament en la seva detecció.

- Persecució: Havent detectat el jugador, aquest el persegueix, actualitzant el destí del *NavMeshAgent* amb la posició del jugador, fins a arribar a una distància determinada d'ell.

- Batalla: Aquest es va movent cap a posicions aleatòries al voltant del jugador mentre l'apunta i el dispara quan té una visió neta del jugador (sense obstacles que puguin obstruir les bales), havent d'esperar un cert temps entre tret i tret. Aquest comportament finalitza i torna a la persecució quan el jugador s'allunya prou.

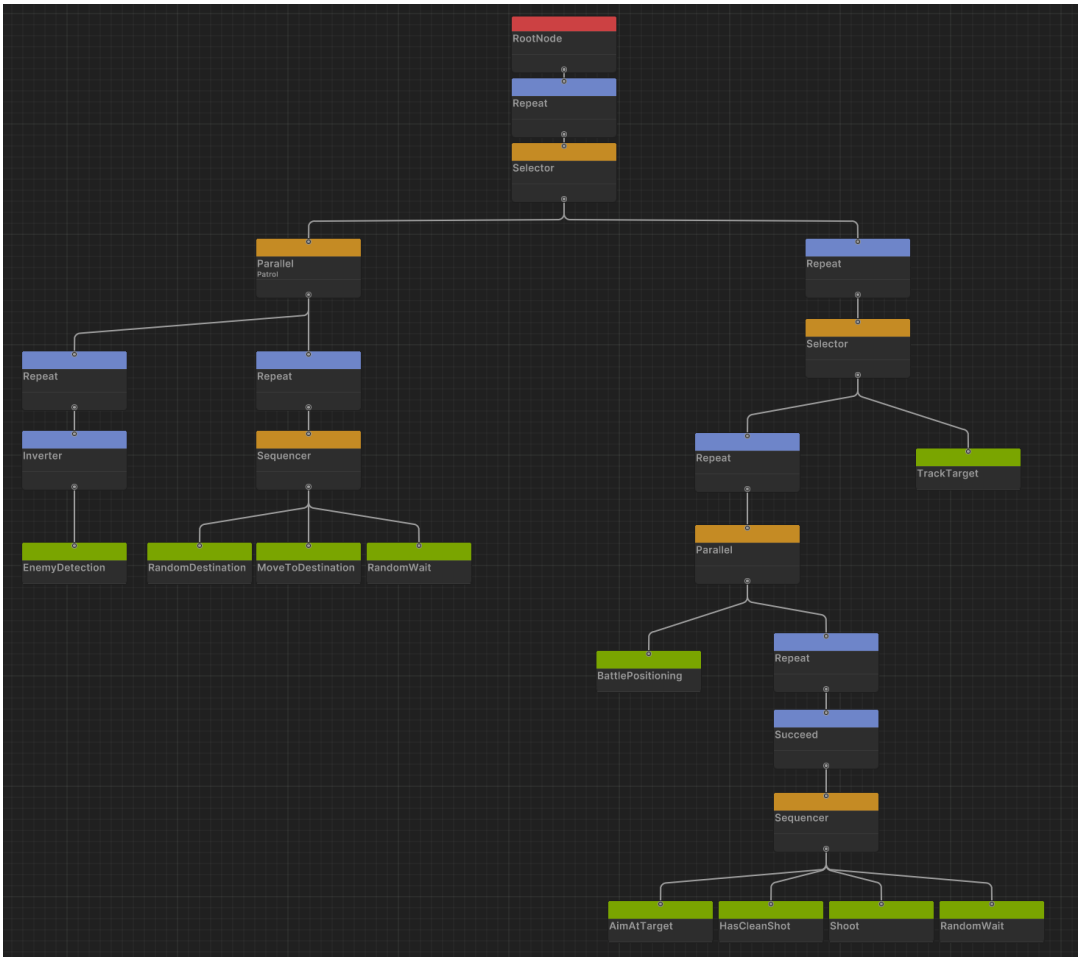

FIGURA 9.1: Captura de pantalla del "behaviour tree" utilitzat per la IA del enemic *Clon*.

HEX-p0p IA

El "behaviour tree" d'aquest enemic, és pràcticament idèntic al del *Clon*, simplement té una ampliació en la seva manera de disparar. Aquest petit canvi, i la modificació de paràmetres com la seva velocitat, vida, les armes que utilitza i la distància de posicionament en la batalla, fan que aquest impliqui un repte diferent a superar.

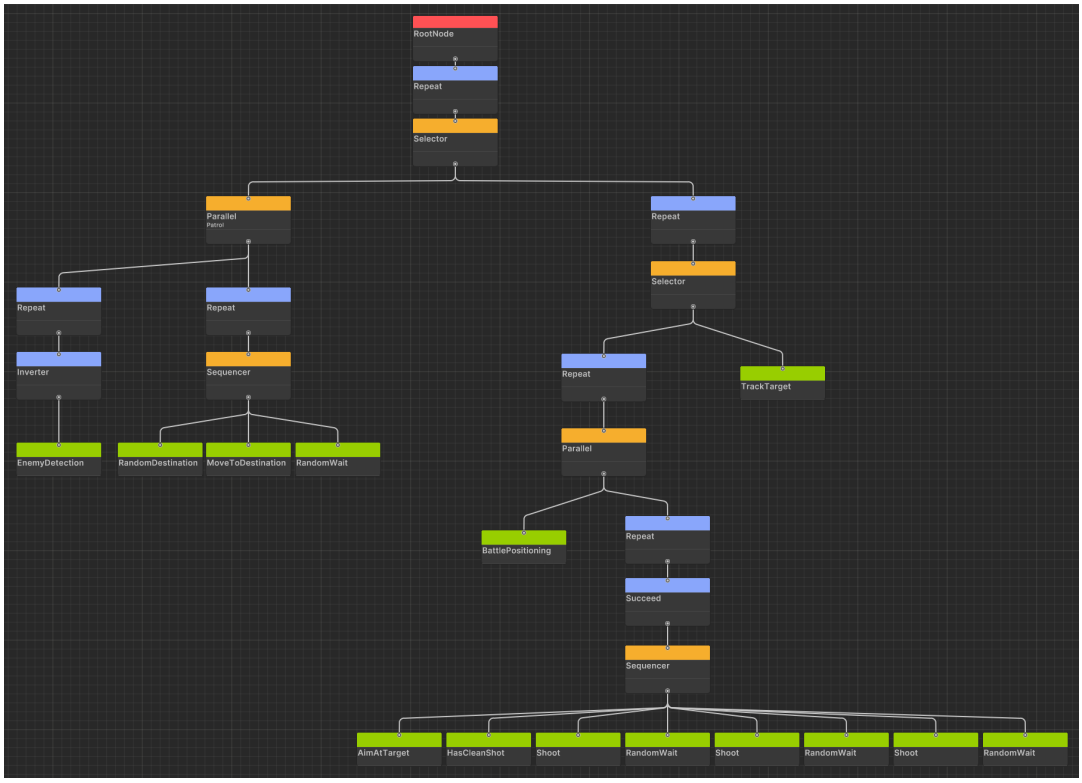

FIGURA 9.2: Captura de pantalla del "behaviour tree" utilitzat per la IA del enemic *HEX-p0p*.

9.3 Armes

Les armes consisteixen en una implementació genèrica i abstracta de *Gun*, que defineix una sèrie de paràmetres, estats (espera, disparant, recarregant, carregant, esperant disparar) i funcions d'interacció (prémer, deixar de prémer, recarregar, cancel·lar procés actual) comunes per a tots els tipus d'armes que s'implementin.

La classe *PatternGun* implementa els mètodes de *Gun* per tal d'obtenir armes que disparin patrons de bales, de manera reminiscent als populars "bullet hell", però que ho facin d'acord amb els paràmetres indicats. Ara veurem, a gran escala, com organitza les transicions d'estat aquest tipus d'arma:

```
algoritme Premer()
    cas estat
        "Esperant" llavors
            Si pot_recarregar /\ recarrega_automaticament
                Recarregant()
            altrament si temps_espera_disparar == 0 /\
    Queda_municio()
                si arma_carrega
                    Carregant()
                altrament
                    Disparant()
                fsi
            fsi
        fllavors

        "Recarregant" llavors
            Si temps_espera_disparar == 0 /\ Queda_municio()
                Cancelar_accio_actual()
                si arma_carrega
                    Carregant()
                altrament
                    Disparant()
                fsi
            fsi
        fllavors
    fcas
falgoritme
```

```
algoritme Deixar_de_premer()
    cas estat
        "Esperant" \/ "Carregant" llavors
            Cancelar_accio_actual()
        fllavors

        "Esperant_disparar" llavors
            si Queda_municio()
                Disparant()
            fsi
        fllavors
    fcas
falgoritme
```

```
algoritme Recarregar()
    si pot_recarregar /\ no Queda_municio()
        cas estat
            "Esperant" \/ "Esperant_disparar" llavors
                Recarregar()
            fllavors

            "Disparant" llavors
                si pot_aturar_disparar
                    Cancelar_accio_actual()
                    Recarregar()
                fsi
            fllavors

            "Carregant" llavors
                Cancelar_accio_actual()
                Recarregar()
            fllavors
        fcas
    fsi
falgoritme
```

Pel que fa a la instanciació de projectils, una *PatternGun*, treballa amb un *BulletPattern*. Aquest *ScriptableObject* defineix un seguit de patrons radials de bales dels quals, quan l'arma ho sol·licita, s'encarrega d'instanciar cada bala en l'orientació indicada pel patró i la posició del punt de tret de l'arma. Cal dir, que la instanciació de les bales, no es fa de manera directa sinó que es demana una instància en desús de la bala en qüestió al *PoolingManager*.

Per acabar, les bales que el patró utilitza, tenen una implementació molt senzilla, la qual simplement consisteix en, un cop activades, desplaçar-se cap endavant fins a col·lidir amb un objecte, sostraient el mal que fan de la vida del personatge, o fins a exhaurir el temps de vida. Un cop hagin finalitzat, però, en comptes de destruir-se com és habitual, aquestes es desactiven i ho notifiquen al *PoolingManager* per tal que puguin ser reutilitzades.

9.4 Shaders

La implementació dels *shaders* s'ha dut a terme mitjançant l'editor visual de nodes de *Unity*, el *ShaderGraph*.

Els *shaders* que s'han fet amb aquesta eina són el *shader* d'estètica "cartoon" que comparteixen tots els models 3D i el *shader* que s'aplica a les partícules que formen les bales.

9.4.1 Toon Shader

La implementació base consisteix en un *shader* que simplifica les ombres projectades sobre els objectes, i els hi afegeix un contorn brillant ("rim light") i finalment carrega la sortida de color en el canal d'emissió en comptes del canal de color base, aconseguint una estètica menys realista i més pròpia a la del 2D. Aquesta implementació

no és pròpia sinó que està extreta íntegrament de la implementació realitzada per *MinionsArt* [18], essent aquesta la que veiem a continuació:

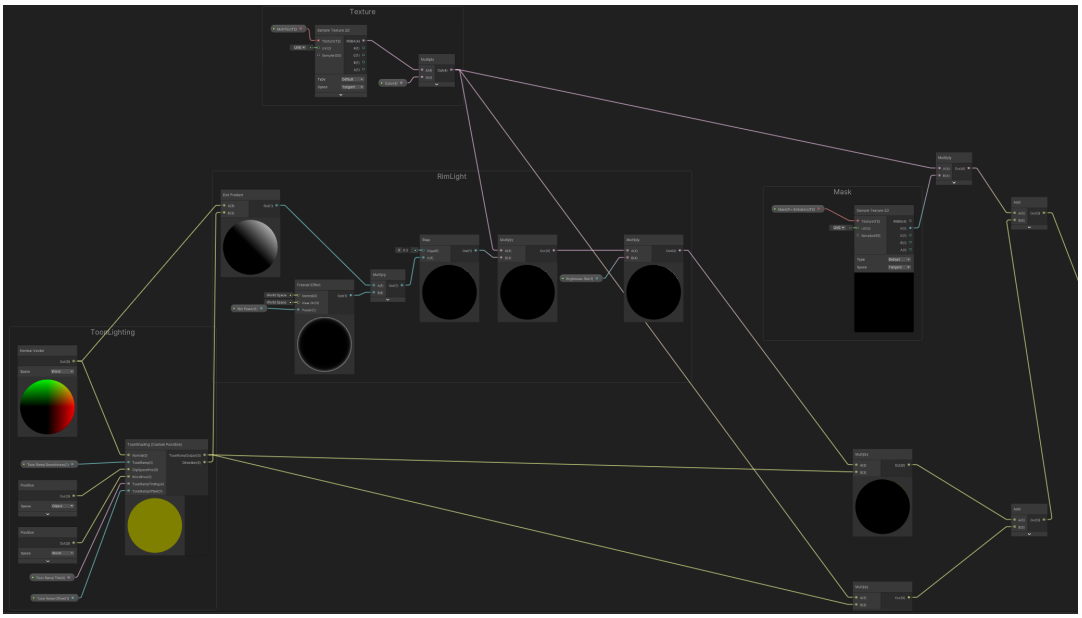

FIGURA 9.3: Captura de pantalla de la implementació base del "toon shader" realitzada per *MinionsArt*.

No obstant això, per a poder fer que els objectes apareguin i desapareguin amb efectes especials s'ha hagut de modificar aquest *shader* de manera que inclogués aquest efecte. Aquest efecte, busca crear la il·lusió de l'objecte dissolent-se i disposa de dos modes. A grans trets, el primer mode, fa un escombrat traçant una línia que marca per on va posant a màxima transparència l'objecte. El segon mode genera una fresa de manera procedimental, que s'aplica sobre l'objecte augmentant progressivament la grandària per a generar cada cop forats més grans fins a fer desaparèixer l'objecte per complet. Aquesta implementació es la següent:

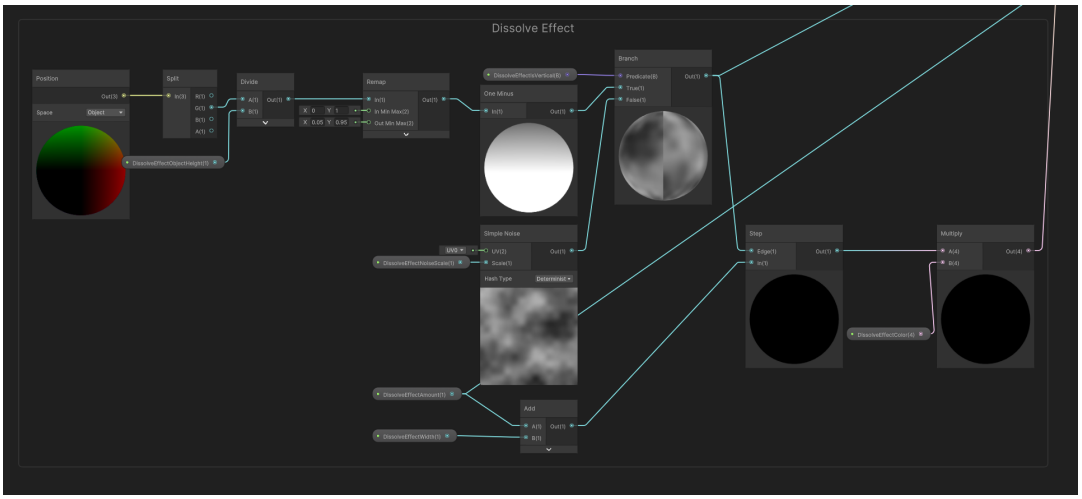

FIGURA 9.4: Captura de pantalla de la implementació de l'efecte de dissolució.

Per acabar, queda pendent unir aquestes dues parts. Això es fa unint les sortides de color de les dues i assignar-les al canal d'emissió. Abans, però, com que tenim

un efecte de dissolució, assignem a les cares internes el color del contorn de l'efecte de dissolució. Això, però no és l'únic, l'efecte de dissolució necessita treballar amb l'opacitat de l'objecte, per tant, és indispensable connectar-lo als canals d'"alpha". La unió resultant és la següent:

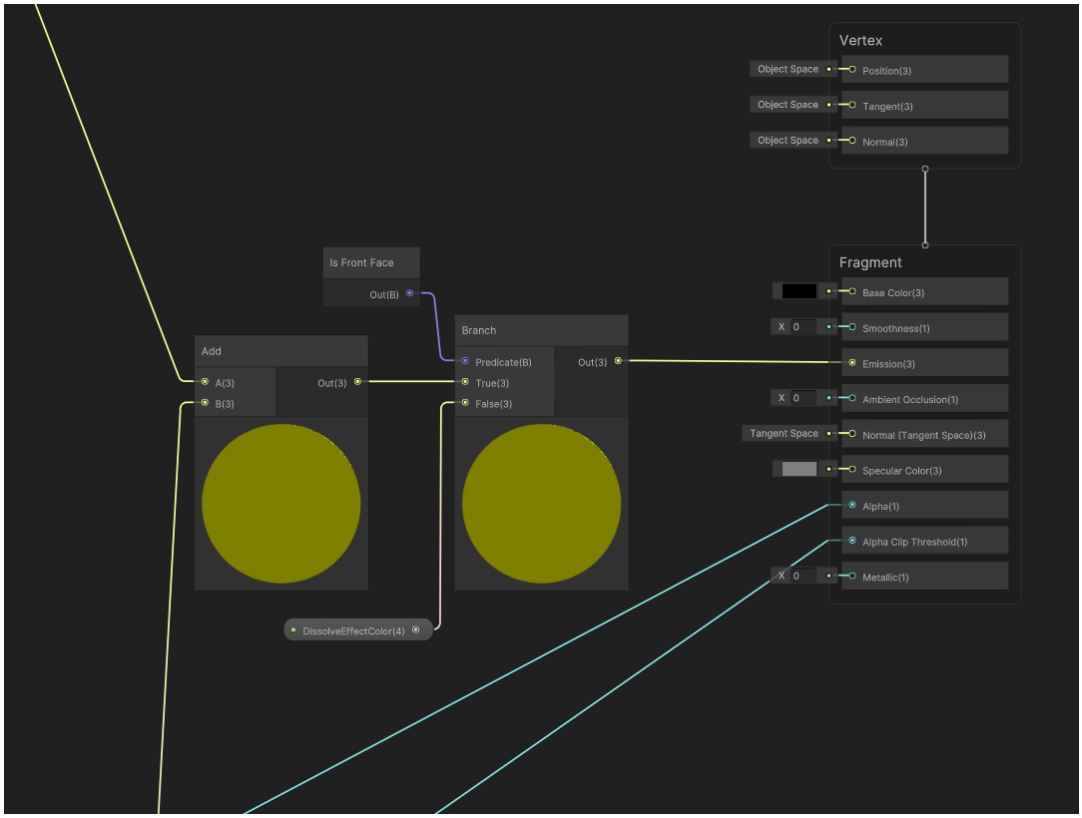

FIGURA 9.5: Captura de pantalla de la unió del "toon shader" amb l'efecte de dissolució

Amb el *shader* finalitzat, només queda assignar-lo a un material per tal de poder ajustar els seus paràmetres i així retocar el resultat final. Per acabar l'efecte de dissolució, però, és necessari un pas més i s'ha implementat el *ToonShadingController* el qual s'encarrega de fer una interpolació en el temps del paràmetre *DissolveEffectAmount* i també activa o desactiva el *DissolveEffectIsVertical* per tal d'alternar entre els dos modes de l'efecte.

9.4.2 Bales

Les bales, pel que fa a gràfics, s'han creat utilitzant el sistema de partícules de *Visual Effect Graph* en combinació amb un *shader* que simula un efecte d'electricitat. Aquesta combinació aconsegueix com a resultat una esfera elèctrica

Pel que fa al *shader* aquest consisteix en la generació de dues textures procedimentals de fresa, que s'interpolen en direccions oposades i s'ajunten, seguidament es processen per a aconseguir únicament línies i s'aplica una màscara radial per aconseguir la forma rodona necessària per a les partícules. El *shader* implementat és el següent:

El sistema de partícules, simplement consisteix a tocar paràmetres, com la vida de les partícules i les dimensions, per tal d'aconseguir que aquestes produeixin un efecte

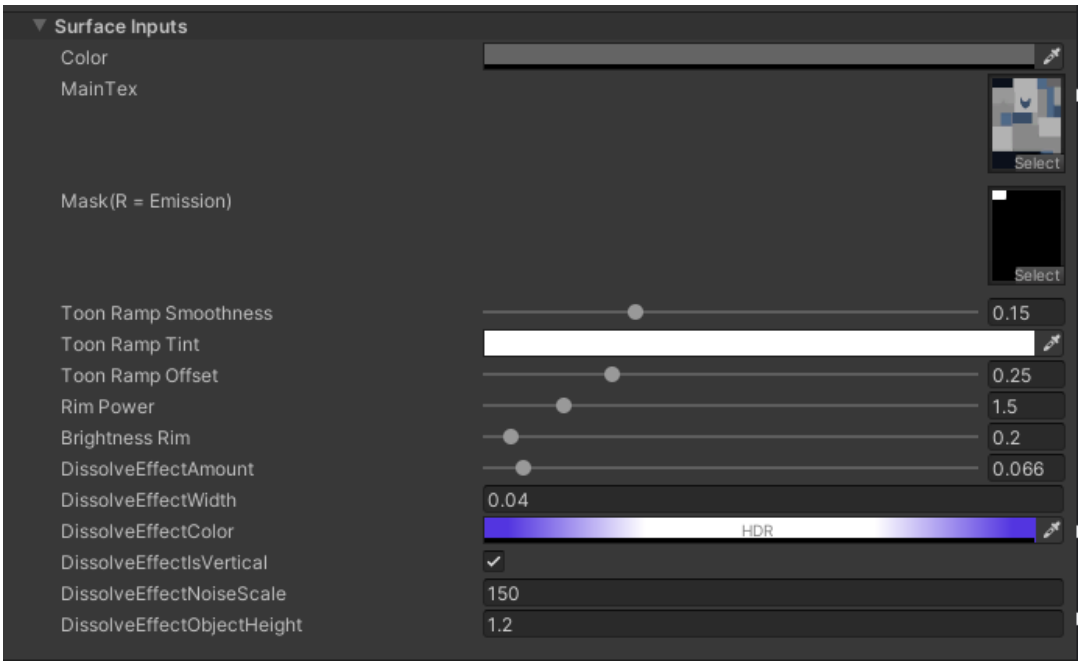

FIGURA 9.6: Captura de pantalla dels paràmetres del material del personatge principal amb el "toon shader" utilitzat en el joc.

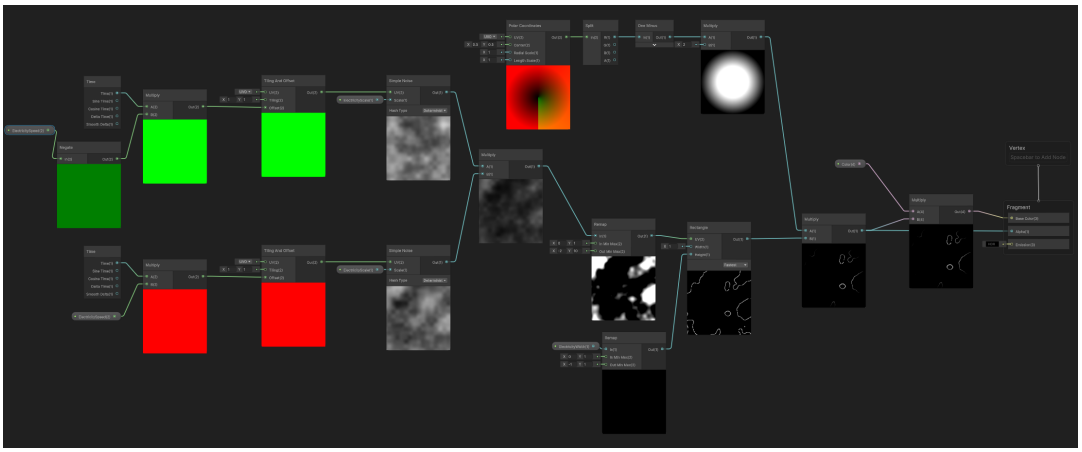

FIGURA 9.7: Captura de pantalla de la implementació del *shader* d'electricitat.

de palpitacions erràtiques. A continuació podem veure com han quedat aquests paràmetres:

9.5 flux de joc

Per tal de gestionar el flux del joc, hem implementat dues classes, el *GameManager* i el *LevelManager*. Tot i la seva naturalesa de gestió, en utilitzar el cicle de vida de *Unity*, aquestes han estat implementades com a components amb un patró de *Singleton* i col·locades en un objecte buit dins l'escena de *Unity*.

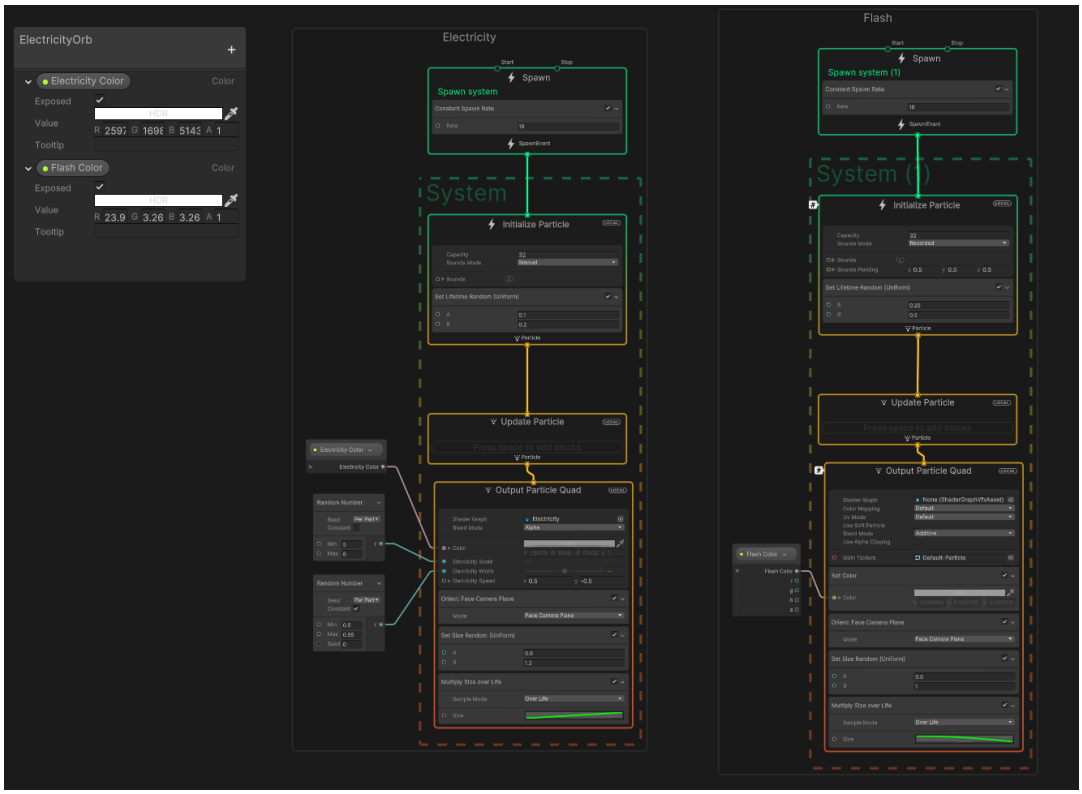

FIGURA 9.8: Captura de pantalla dels paràmetres del sistema de partícules utilitzat per a crear les bales.

9.5.1 GameManager

Pel que fa al codi, es troba implementat com un seguit de mètodes que s'encarreguen d'iniciar la navegació entre les dues escenes (menú principal i joc) i mostrar els menús en la partida (fi de joc i pausa). Aquesta navegació es fa carregant la nova escena de manera asíncrona a través del *SceneManager* mentre es mostra la pantalla de càrrega. També és qui s'encarrega de gestionar la pausa durant la partida, posant el *Time.timeScale* de 1 a 0 per tal d'aturar els*Update()* de tots els objectes de l'escena. Com que es troba implementat com un *singleton*, aquest pot ser fàcilment accedit per a qualsevol objecte sense tenir una referència a ell, i així pot servir les peticions de pausa del personatge controlat pel jugador o les peticions de finalització de partida. Addicionalment, s'ha fet ús de la funció *DontDestroyOnLoad(this)* la qual permet que aquest objecte perduri entre escenes.

Pel que fa a l'objecte que manté aquest component, aquest i els seus fills es converteixen en una entitat que no es destrueix en el canvi d'escenes, per tant, un cop el trobem per primera vegada en el menú principal, aquest objecte ens acompanyarà a la resta d'escenes. Aquest factor s'utilitza per a posar tots els *Canvas* pertinents als menús de la partida i a la pantalla de càrrega dins la jerarquia de l'objecte amb aquest component per tal de poder mantenir una referència a ells i així poder-los mostrar i ocultar sense importar en l'escena en què ens trobem.

9.5.2 LevelManager

Es troba implementat com un objecte buit en l'escena de joc*Unity* que, just després que aquesta ha estat carregada, llegeix el fitxer JSON de la generació del pis, construeix un pis amb aquesta informació utilitzant *Floor*, li demana que faci l'instanciació a la posició *(0,0,0)* de l'escena de *Unity* i també col·loca el personatge a la primera posició del pis construït. Un cop acabada aquesta inicialització aquesta classe implementa mètodes per a controlar els diversos aspectes de la partida, que es redueixen a les transicions entre habitacions.

Quan els *accessos* detecten una transició d'habitacions per part del jugador, notifiquen al *LevelManager* que s'ha produït una transició i aquest, si l'habitació no es trobava superada, fa una assignació aleatòria dels enemics que hauran d'aparèixer i li demana a l'habitació que els col·loqui aquests enemics en posicions aleatòries dins d'ella. Un cop havent fet això, aquest també demana a l'habitació que bloquegi totes les seves portes i queda a l'espera que, aquesta li notifiqui que tots els enemics han mort per tal de desbloquejar-les i actualitzar el nivell d'accés del jugador si l'habitació superada era d'un nivell més elevat que el nivell d'accés del qual aquest disposava. Seguidament, veurem amb més detall com és s'instancien els enemics en una transició:

```
1  algoritme Intanciar_enemics_habitacio_actual()
2      diccionari_enemics = Diccionari<Enemic, Nombre>()
3
4      min_total_enemics = Logaritme(habitacions_superades + 2, 2)
5      max_total_enemics =
       Minim(habitacio.Obtenir_posicions_terra().nombre_elements /
       10, jugador.nivell_acces * 2 + 1)
6      total_enemics = Aleatorietat(minim, maxim)
7
8      enemics_restants = total_enemics
9      mentre enemics_restants > 0
10         max_quantitat_enemic_seleccionat = Minim(enemics_restants,
       total_enemics / diccionari_enemics.nombre_elements)
11         quantitat_enemic_seleccionat(1,
       max_quantitat_enemic_seleccionat + 1)
12         enemic_seleccionat = tipus_enemics[Aleatorietat(0,
       tipus_enemics.nombre_elements)];
13
14         si diccionari_enemics.conte(enemic_seleccionat)
15             diccionari_enemics[enemic_seleccionat] +=
       quantitat_enemic_seleccionat
16         altrament
17             diccionari_enemics.Afegir(enemic_seleccionat,
       quantitat_enemic_seleccionat)
18         fsi
19
20         enemics_restants -= quantitat_enemic_seleccionat
21     fmentre
22
23     habitacio_actual.Instanciar_enemics_aleatoriament
       (diccionari_enemics)
24 falgoritme
```

Capítol 10

Implementació i resultats

En aquest capítol farem un breu incís sobre la legislació i normativa vigent en el marc del videojoc desenvolupat i llavors repassarem amb diverses captures els resultats d'aquesta.

10.1 Legislació i normativa vigent

El videojoc resultant d'aquest projecte no treballa ni recull cap mena de dada de caràcter personal i, per tant, no s'ha tingut en compte la *Llei Orgànica de Protecció de Dades de Caràcter Personal (LOPD)*.

Pel que fa a la seguretat, aquesta s'executa a nivell d'usuari i no presenta cap problema de seguretat.

Per acabar, tampoc s'ha tingut en compte *la Llei de serveis de la societat de la informació i comerç electrònic (LSSICE)*. Tot i que amb el material produït es vulgui acabar construint un producte a comercialitzar com a videojoc, aquest actualment es troba en una fase de prototipatge que es troba molt lluny dels estàndard de qualitat i extensió per a un comercialitzable d'aquestes característiques.

10.2 Captures de pantalla

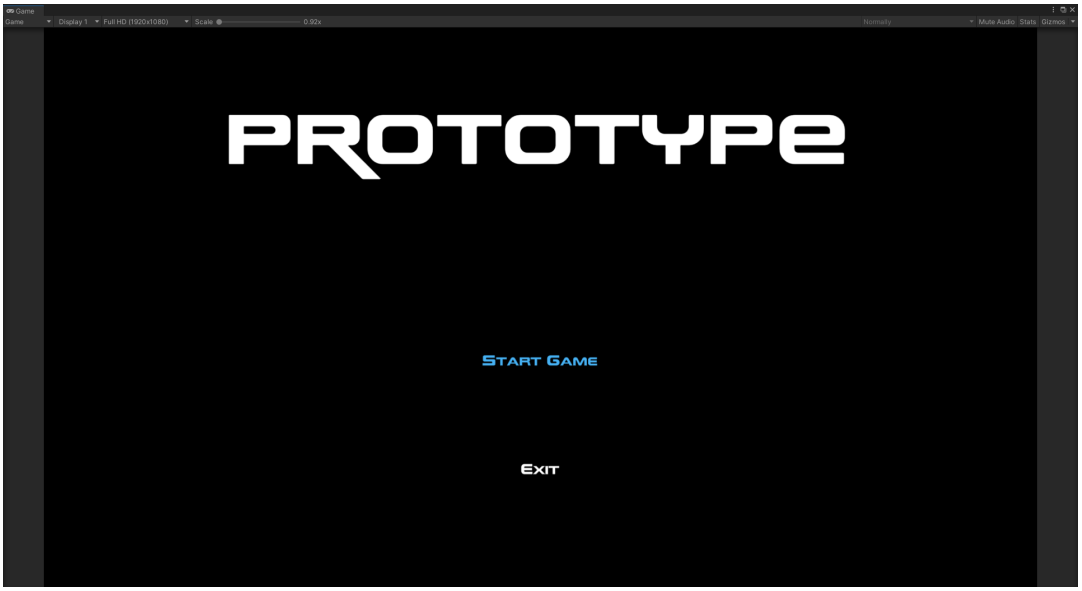

FIGURA 10.1: Captura de pantalla del menú principal.

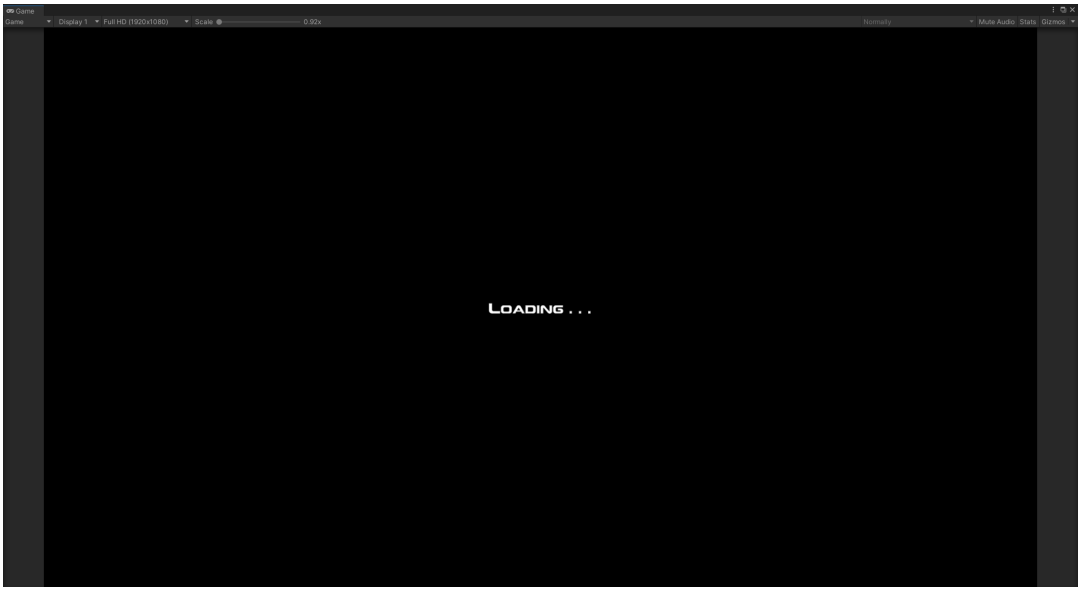

FIGURA 10.2: Captura de pantalla del menú principal.

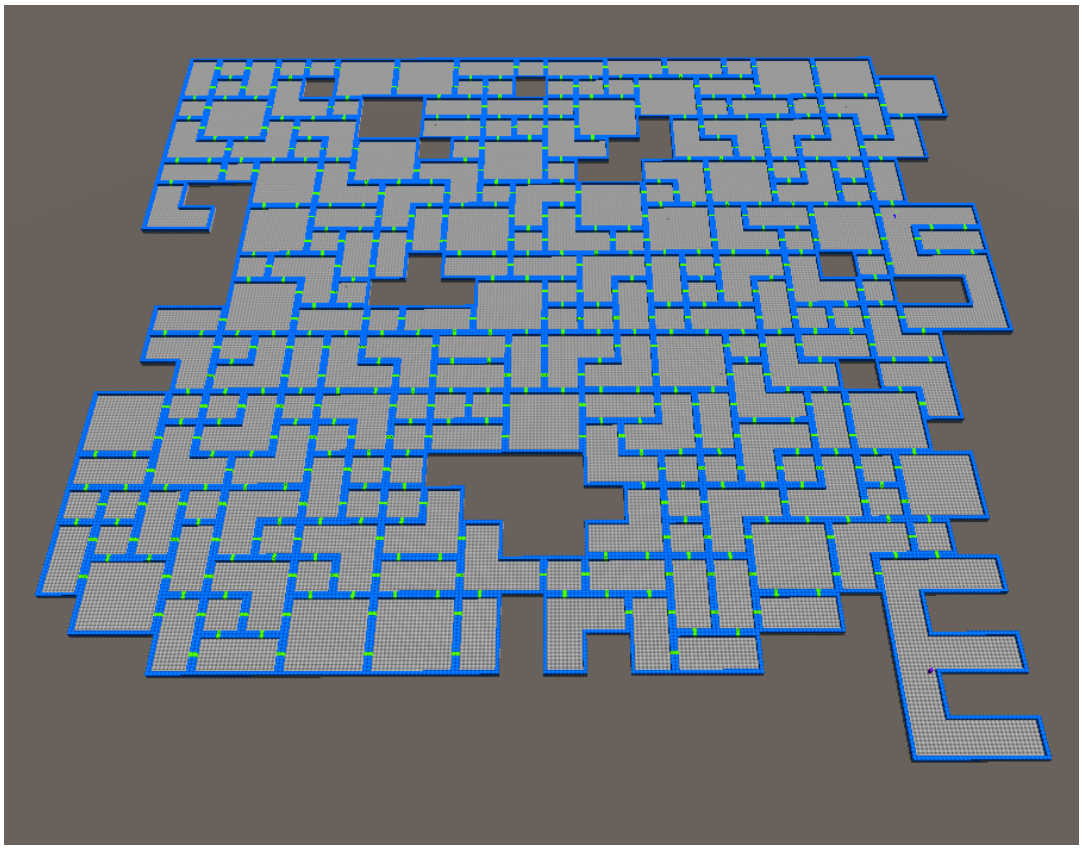

FIGURA 10.3: Captura de pantalla de la generació d'un pis. L'habitació en forma de "S" és l'entrada i l'habitació amb forma de "E" la sortida.

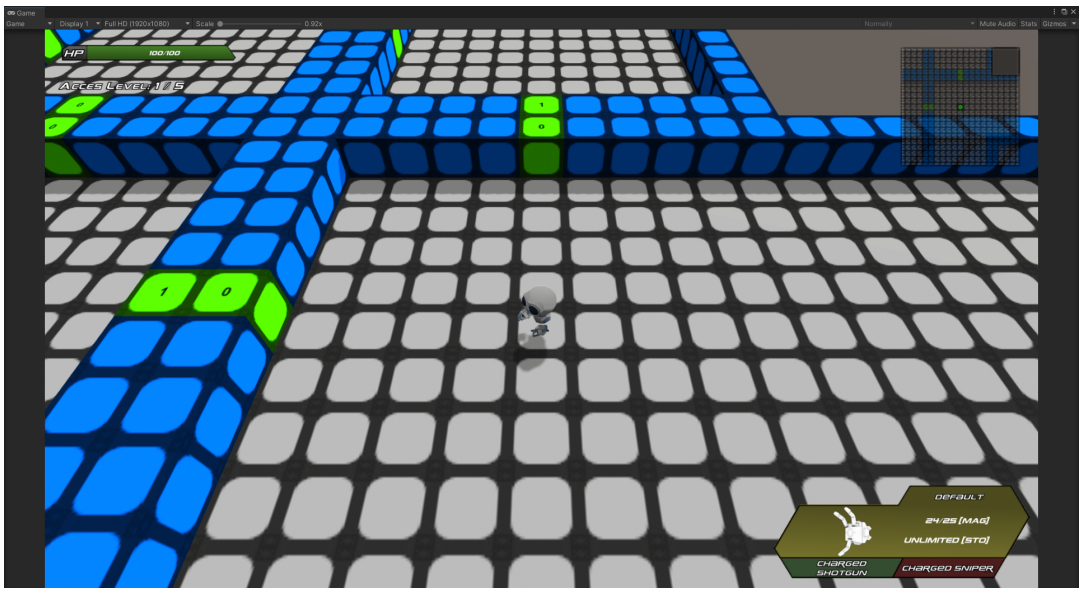

FIGURA 10.4: Captura de pantalla de pantalla on veiem el jugador començant a l'habitació d'entrada.

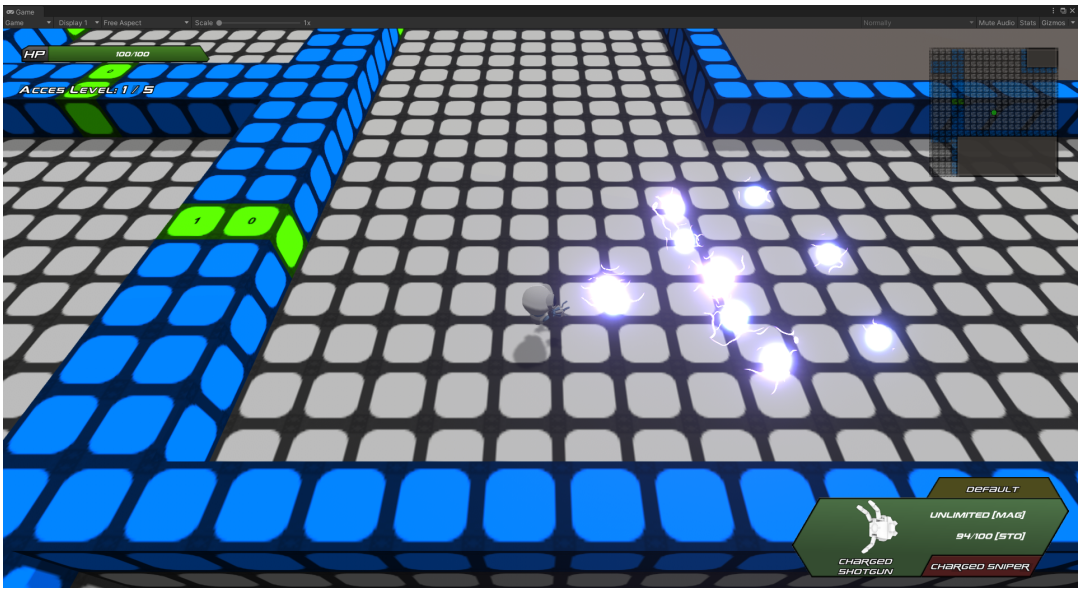

FIGURA 10.5: Captura de pantalla on podem apreciar el patró de bales d'una de les armes del jugador.

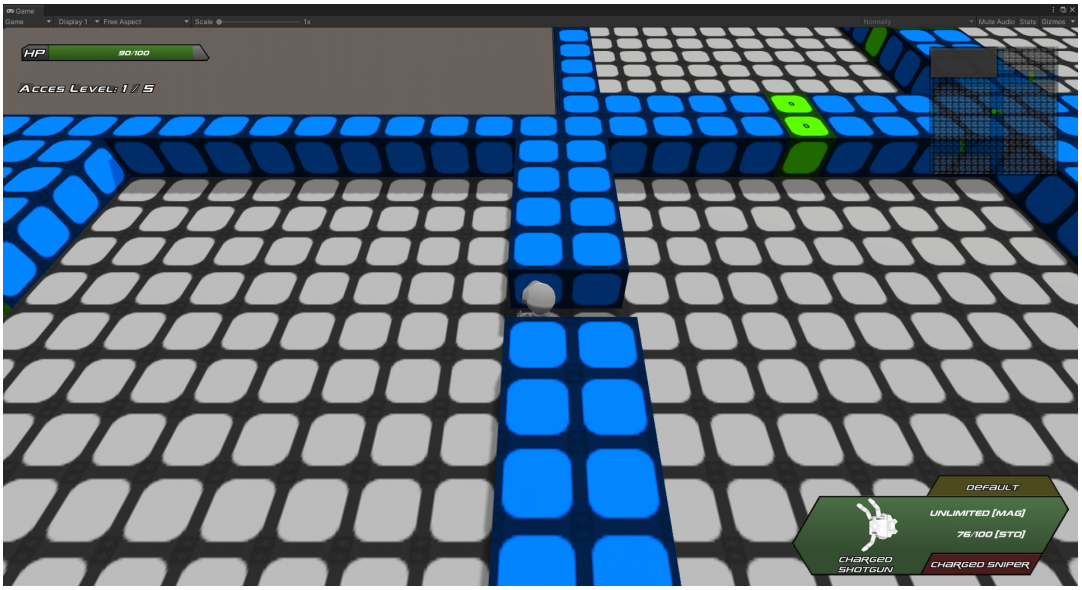

FIGURA 10.6: Captura de pantalla on podem veure el jugador travessant una porta.

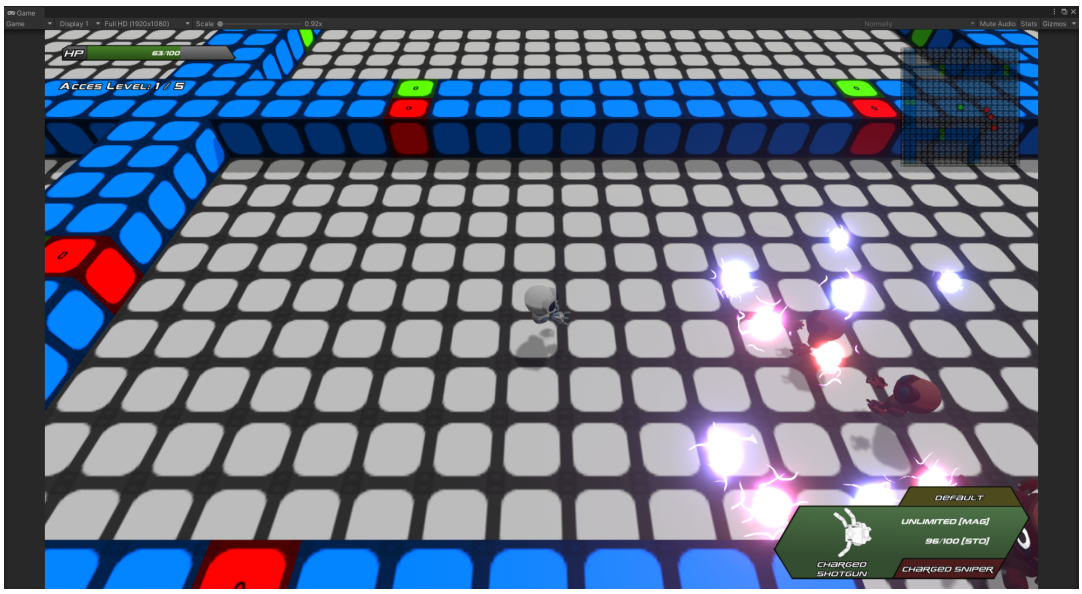

FIGURA 10.7: Captura de pantalla d'un combat (1).

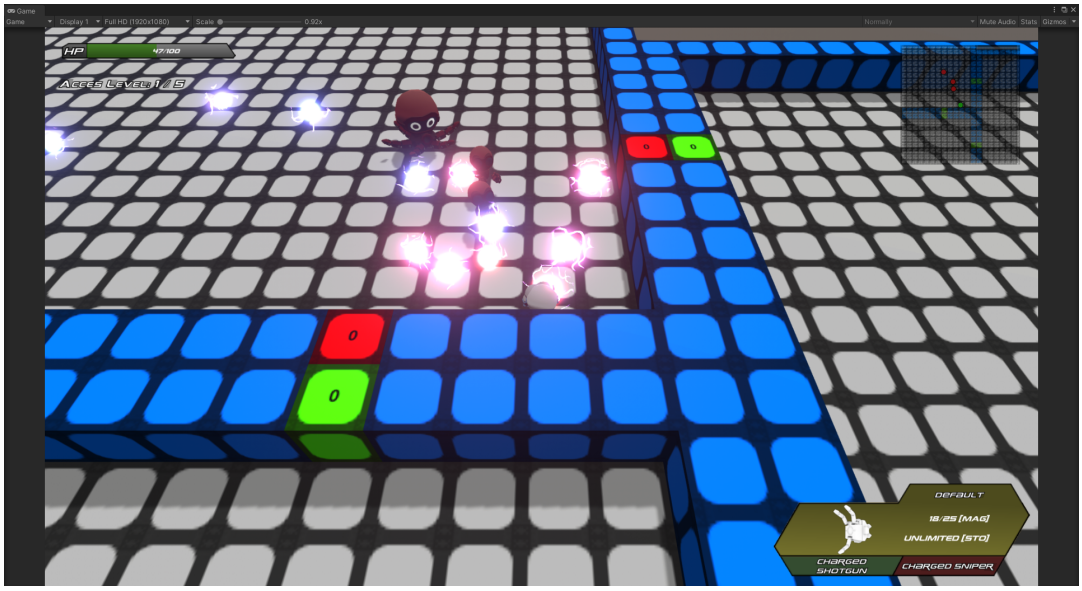

FIGURA 10.8: Captura de pantalla d'un combat (2).

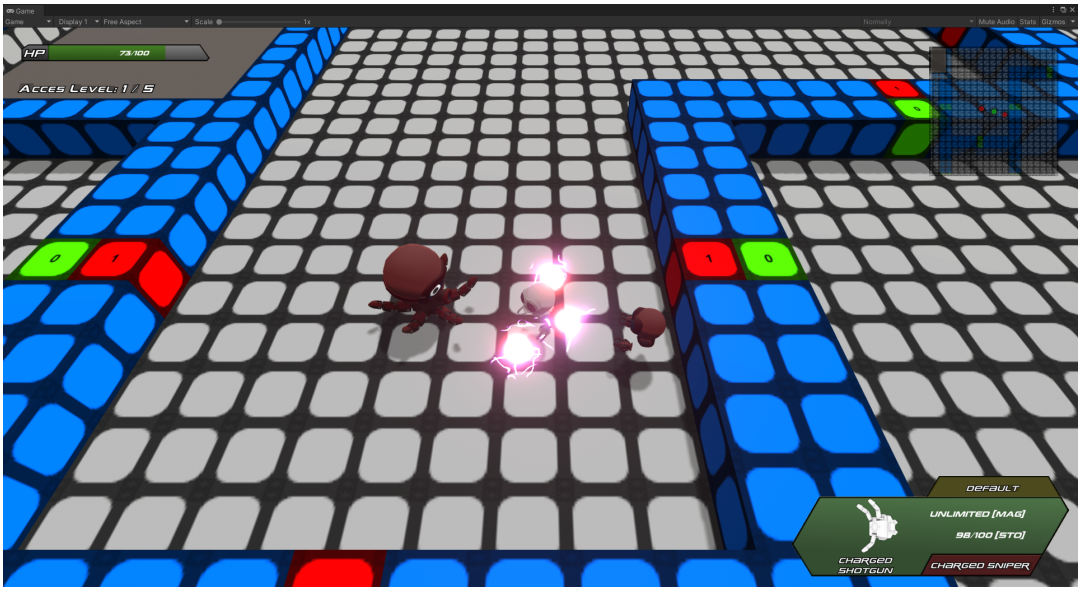

FIGURA 10.9: Captura de pantalla d'un combat (3).

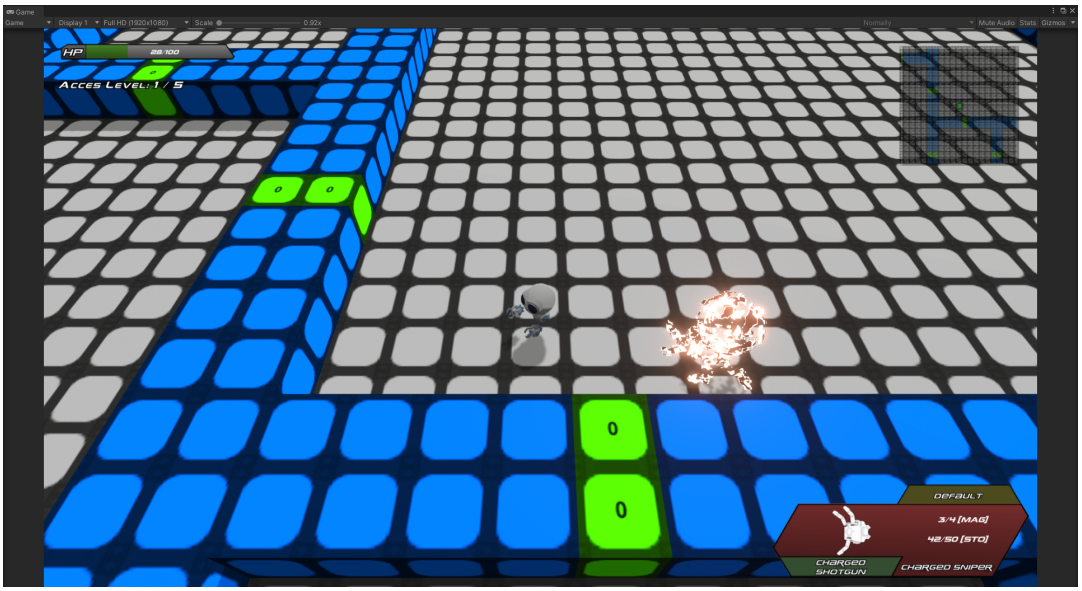

FIGURA 10.10: Captura de pantalla on podem veure la mort d'un enemic.

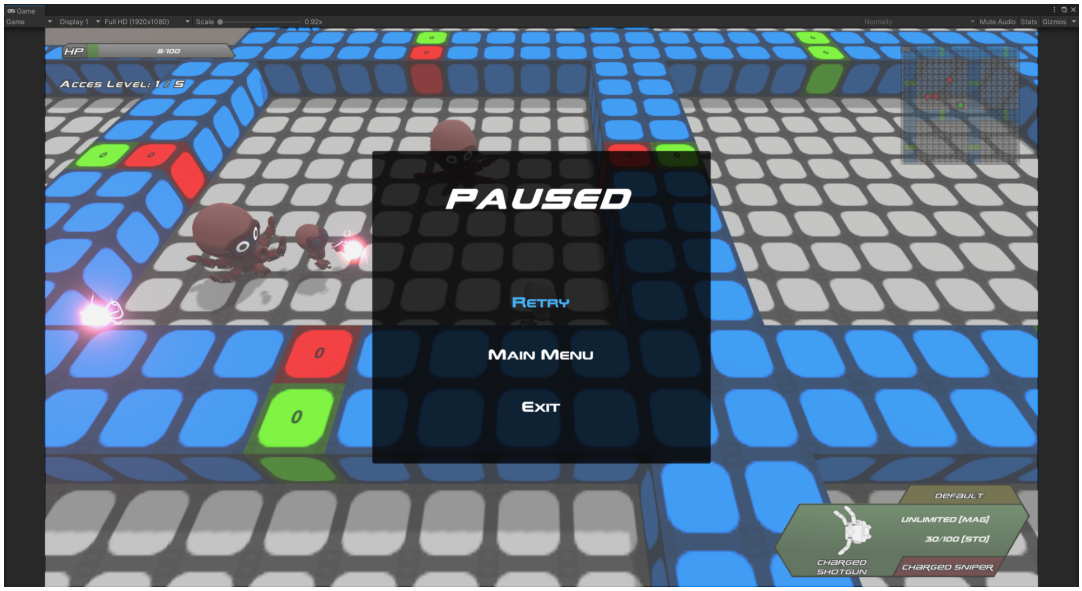

FIGURA 10.11: Captura de pantalla del menú de pausa.

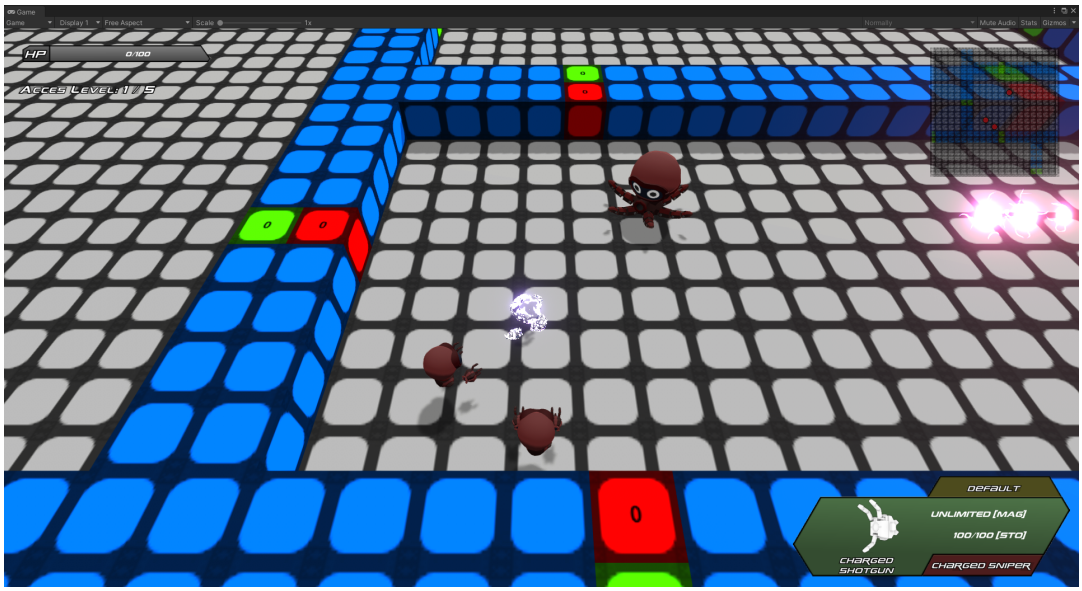

FIGURA 10.12: Captura de pantalla on podem veure la mort del jugador.

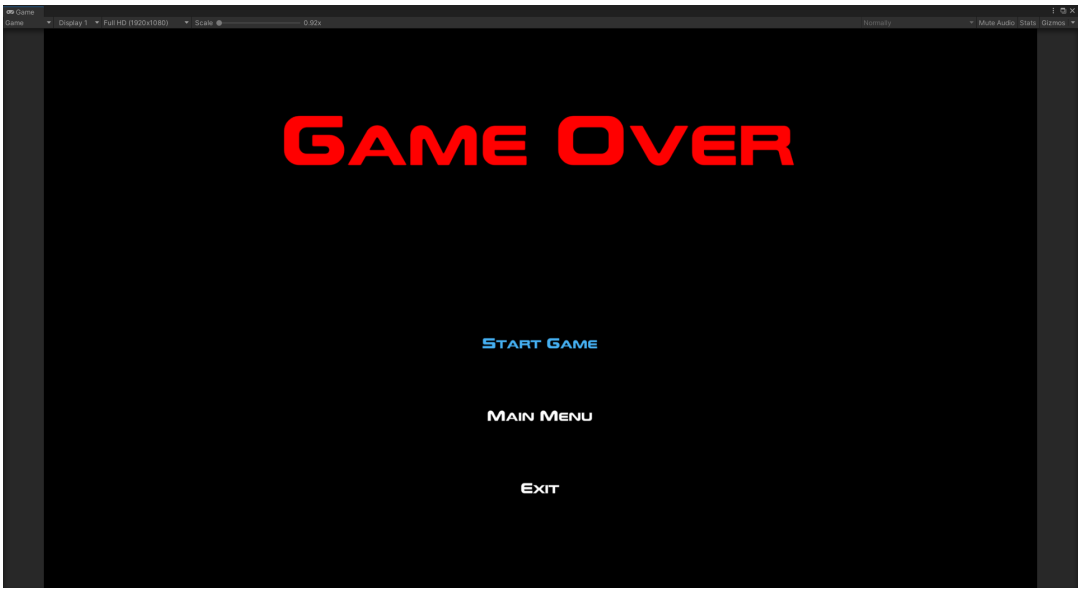

FIGURA 10.13: Captura de pantalla del menú de fi de partida mostrant la derrota del jugador.

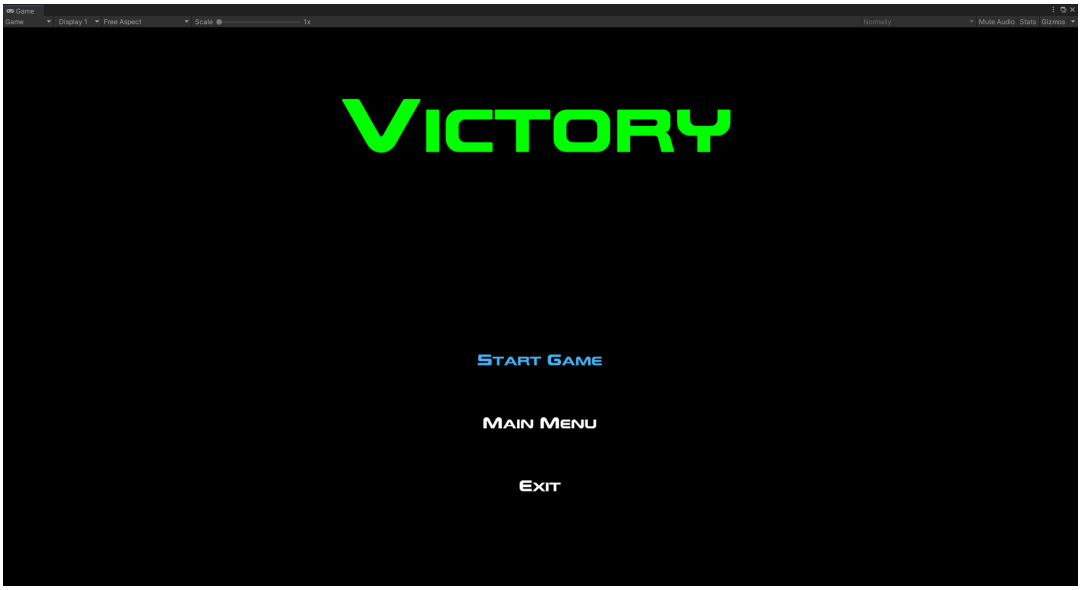

FIGURA 10.14: Captura de pantalla del menú de fi de partida mostrant la victòria del jugador.

Capítol 11

Conclusions

Primer de tot és important aclarir que aquest projecte ha complert tots els objectius que es plantejaven inicialment (capítol 1). Tot i això, considero que un dels aspectes més positius del treball realitzat es la fluïdesa, reactivitat i escalabilitat de tot el que engloba el control del personatge principal i les armes que aquest fa servir.

El desenvolupament d'aquest projecte ha estat sens dubte el repte més gran i multidisciplinari que fins a aquestes dates he tingut l'oportunitat de fer front. Tot i que ja comptava amb coneixements sobre desenvolupament de videojocs, a causa de la meva formació prèvia (CFGS en animació 3D, jocs i entorns interactius) aquest projecte ha suposat el meu major repte tècnic, ja que la complexitat en la programació requerida era molt major als projectes que havia desenvolupat amb anterioritat. També el que volia aconseguir, era molt més personal i específic i no em servien recursos gratuïts de *Unity* que havia fet servir en altres petits projectes.

En el projecte hi ha hagut sobretot, dos aspectes tècnics que han estat els meus majors reptes i contratemps a l'hora de desenvolupar. Aquests són la integració de les mecàniques del joc utilitzant el motor de físiques *Unity*, fet que provocava deteccions de col·lisions inadequades. Així com la filosofia de components i "game objects" en la que el mateix motor es troba basat, la qual em costava especialment combinar amb components encarregats a la gestió o flux de joc.

Dit això, el desenvolupament de tots els diferents sistemes que conformen el videojoc, m'han portat moltes més hores que les previstes amb anterioritat. No obstant això, aquest prototip, constitueix una base prou sòlida per a construir un producte final que sigui comercialitza-ble, objectiu que és el meu propòsit més enllà del desenvolupament d'aquest treball. Considero que els sistemes desenvolupats ajuden moltíssim a la seva reutilització. Cal dir que aquest fet, dona suport a la meva hipòtesi inicial, sobre que la generació per procediments és ideal per a projectes amb pocs recursos humans, tot i que ara veig ideal la incorporació d'una altra persona per a la continuació i finalització del projecte en uns termini d'uns dos anys.

Finalment, però, m'agradaria afegir que, l'aspecte més satisfactori de la realització d'aquest treball, ha estat poder desenvolupar un videojoc i "tornar a aprendre" la programació de videojocs en *Unity*, ara però, des de la perspectiva d'un enginyer informàtic. Els meus coneixements adquirits al llarg de la carrera m'han permès entendre una documentació i conceptes que abans de realitzar aquests estudis era incapaç d'entendre, i això ha estat clau per aconseguir un nivell de polidesa en els controls i mecàniques de joc que amb anterioritat no havia estat capaç d'aconseguir.

Capítol 12

Treball futur

Aquest projecte va ser concebut des de l'inici com a un prototip funcional, que servis sobretot com a base per provar les mecàniques i controls del jugador en un entorn generat de manera aleatòria. No obstant això, es tenien clars certs aspectes essencials del producte final que s'ha pres la decisió deixar de banda. Entre aquests aspectes en podem destacar els següents:

- Parametrització i aleatorietat en la generació interna de les sales: Actualment, el sistema d'arxius JSON no admet la definició interna d'una sala, aquest només permet la definició estructural. La definició de l'interior es faria de dues maneres diferents. El primer mètode consistiria a poder definir cel·la a cel·la els blocs que aquesta conté. El segon, consistiria a definir els mòduls que aquesta hauria de contenir. Aquests mòduls serien conjunts de celles, prèviament definits, amb els respectius blocs, els quals s'intentarien col·locar en la sala seguint un conjunt de restriccions. D'aquesta manera la navegació de la sala, es podria construir com a un repte per si sol, amb un seguit de trampes i obstacles que el jugador hagi de sobreviure.

- Integració del sistema de col·locació en el punt anterior: Per tal de fer més fàcil les qüestions de balanceig, la col·locació d'enemics s'hauria de definir, també per sala, amb un seguit de restriccions.

- Creació de més enemics: El joc requereix múltiples enemics amb comportaments diversos per tal d'obligar el jugador a aprendre diferents patrons. Tot i que, per la part de programació, amb els sistemes utilitzats, no és necessària una gran feina, en l'àmbit de disseny i art es necessitaria bastant feina per fer funcionar cadascun d'ells.

- Creació de trampes: La idea consisteix a crear blocs amb components especials que serien col·locats a les sales i que els facin interactuar amb l'entorn o el jugador d'una manera determinada. Alguns exemples d'aquests blocs serien:

  - Fer mal o restringir els moviments del jugador que hi ha a sobre o al seu voltant.

  - Crear vincles entre blocs del mateix tipus en la mateixa sala. En travessar el vincle entre ells és quan es produiria l'efecte de mal o restricció.

  - Disparar patrons de bales.

- Recol·lecció d'armes en la partida: El sistema d'armes creat és un sistema completament parametrizable que permet crear un infinit nombre d'armes de diferent nivell de poder. Aquesta expansió consistiria a definir prèviament un conjunt d'armes, per tal que aquestes es puguin repartir de manera aleatòria pels diferents nivells del joc. Això faria que el jugador adapti l'estil de joc segons les armes que disposi.

- Sistema d'ítems i cofres: El jugador compta amb vida i un sistema de munició per les armes, però actualment no hi ha res que permeti recuperar la vida o munició gastada. Aquest sistema buscaria expandir el punt anterior, per tal de també repartir pel nivell un seguit d'ítems i cofres que permetrien tant recuperar els recursos gastats com augmentar les estadístiques bases del jugador de manera temporal o permanent. Aquest sistema també permetria obtenir claus que llavors el jugador podria decidir gastar o no per a obrir certes portes o cofres.

- Implementació de botigues en els nivells per a la compra d'ítems: A grans trets consistiria en sales amb un seguit d'ítems seleccionats de manera aleatòria, que es podrien recollir pagant un preu determinat. Les monedes s'inclourien dins del sistema d'ítems i cofres.

- Expansió del sistema d'habilitats: a banda de l'habilitat de tele-transport que inclou aquest prototip, es volen crear més habilitats, tant de desplaçament com d'escut i atac.

- Producció de models i textures per als nivells i ítems: Actualment, no hi ha textures per a crear una ambientació per cadascun dels nivells finals del joc. També caldrien models diferents de cubs perfectes per a blocs especials, per exemple, les portes, els tele-transportadors, les trampes, d'entre altres. Addicionalment, el sistema que es vol implementar d'Ítems recol·lectables, hauria de comptar amb "sprites" o models que els fessin reconeixibles a simple vista.

- Producció i integració de música i efectes de so: El prototip actual no inclou so de cap mena i aquests s'haurien d'afegir tant en concepte de música d'ambient com efectes de so, com per exemple el tret i impacte de bales. De la mateixa manera s'hauria d'incloure en els menús del joc l'opció d'ajustar el volum.

Finalment, a banda d'aquests aspectes, també es considera la possibilitat d'haver de realitzar canvis en el sistema de generació dels nivells per a poder utilitzar diversos fils d'execució. Això és perquè la càrrega del nivell, amb els aspectes anteriorment comentats, seria molt major i podria portar a temps d'espera notablement elevats. També una qüestió interessant és que aquests canvis ajudarien a poder convertir aquest sistema de generació procedimental en un producte independent, en forma de paquet de *Unity*, de manera que aquest pogués ser usat per al desenvolupament d'altres projectes.

Capítol 13

Bibliografia

[1] Edmund McMillen. *The binding of Isaac*. Accedit: 2023/08/15. URL: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/.

[2] Dodge Roll. *Enter the Gungeon*. Accedit: 2023/08/15. URL: https://store.steampowered.com/app/311690/Enter_the_Gungeon/?l=spanish.

[3] Unity Technologies. *Unity*. Accedit: 2023/08/15. URL: https://unity.com/es.

[4] Capcom. *Ghosts'n Goblins*. Accedit: 2023/08/15. URL: https://store.steampowered.com/app/1556690/Capcom_Arcade_StadiumGhosts_n_Goblins/?l=latam.

[5] Michael Toy i Glenn Wichman. *Rogue*. Accedit: 2023/08/15. URL: https://en.wikipedia.org/wiki/Rogue_(video_game)#:~:text=Rogue%20(also%20known%20as%20Rogue,later%20contributions%20by%20Ken%20Arnold..

[6] MediBang Inc. *Medibang*. Accedit: 2023/08/15. URL: https://medibangpaint.com/es/.

[7] Equip Inkscape. *Inkscape*. Accedit: 2023/08/15. URL: https://inkscape.org/es/.

[8] Autodesk. *Maya*. Accedit: 2023/08/15. URL: https://www.autodesk.es/products/maya/overview?term=1-YEAR&tab=subscription.

[9] Microsoft. *Visual Studio*. Accedit: 2023/08/15. URL: https://visualstudio.microsoft.com/es/.

[10] Fundació Blender. *Blender*. Accedit: 2023/08/15. URL: https://www.blender.org/.

[11] Wikipedia. *Desarrollo en cascada*. Accedit: 2023/08/15. URL: https://es.wikipedia.org/wiki/Desarrollo_en_cascada.

[12] Wikipedia. *Desarrollo en espiral*. Accedit: 2023/08/15. URL: https://es.wikipedia.org/wiki/Desarrollo_en_espiral.

[13] Ariel Manzur i la comunidad de Godot Juan Linietsky. *Godot*. Accedit: 2023/08/15. URL: https://godotengine.org/.

[14] Epic Games. *Unreal*. Accedit: 2023/08/15. URL: https://www.unrealengine.com/es-ES.

[15] Unity Technologies. *API de Unity*. Accedit: 2023/08/15. URL: https://docs.unity3d.com/ScriptReference/.

[16] The Kiwi Coder. *Paquet de behaviour trees*. Accedit: 2023/08/15. URL: https://www.thekiwicoder.com/behaviour-tree.

[17] Jun'ya Ota (ZUN). *Touhou*. Accedit: 2023/08/15. URL: https://es.wikipedia.org/wiki/Touhou_Project.

[18] Minions Art. *Toon shader*. Accedit: 2023/08/15. URL: https://www.patreon.com/posts/lit-toon-shader-54740865.

Capítol 14

Annexos

14.1 Fitxer JSON per a la generació del pis

A continuació podem veure el *JSON* utilitzat per a definir el pis aleatori present en el videojoc desenvolupat:

```
{
  "random": true,
  "rowModules": 25,
  "columnModules": 25,
  "moduleSize": 11,
  "roomsData": [
    {
      "name": "single",
      "isMainRoomOnly": false,
      "matrixIsStructure": true,
      "matrixColumns": 1,
      "matrix": [
        1
      ]
    },
    {
      "name": "line",
      "isMainRoomOnly": false,
      "matrixIsStructure": true,
      "matrixColumns": 2,
      "matrix": [
        1,1
      ]
    },
    {
      "name": "corner",
      "isMainRoomOnly": false,
      "matrixIsStructure": true,
      "matrixColumns": 2,
      "matrix": [
        1,0,
        1,1
      ]
    },
    {
      "name": "square",
      "isMainRoomOnly": false,
      "matrixIsStructure": true,
      "matrixColumns": 2,
```

```
      "matrix": [
        1,1,
        1,1
      ]
    },
    {
      "name": "start",
      "isMainRoomOnly": true,
      "matrixIsStructure": true,
      "matrixColumns": 3,
      "matrix": [
        1,1,1,
        1,0,0,
        1,1,1,
        0,0,1,
        1,1,1
      ]
    },
    {
      "name": "end",
      "isMainRoomOnly": true,
      "matrixIsStructure": true,
      "matrixColumns": 3,
      "matrix": [
        1,1,1,
        1,0,0,
        1,1,1,
        1,0,0,
        1,1,1
      ]
    }

  ],
  "roomsLocationContraintsData": [
    {
      "roomName": "start",
      "linkRoomName": "",
      "fixedType": "entrance",
      "fixedRotation": -1,
      "objectiveNumber": 1,
      "minQuantity": 1,
      "aditionalQuantity": 0,
      "probabilitySpawn": 0,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "end",
      "linkRoomName": "",
      "fixedType": "exit",
      "fixedRotation": 0,
      "objectiveNumber": 5,
      "minQuantity": 1,
      "aditionalQuantity": 0,
      "probabilitySpawn": 0,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "line",
      "linkRoomName": "",
      "fixedType": "subObjective",
      "fixedRotation": -1,
```

```
      "objectiveNumber": 2,
      "minQuantity": 1,
      "aditionalQuantity": 2,
      "probabilitySpawn": 50,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "square",
      "linkRoomName": "",
      "fixedType": "subObjective",
      "fixedRotation": -1,
      "objectiveNumber": 3,
      "minQuantity": 1,
      "aditionalQuantity": 0,
      "probabilitySpawn": 50,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "corner",
      "linkRoomName": "",
      "fixedType": "subObjective",
      "fixedRotation": -1,
      "objectiveNumber": 4,
      "minQuantity": 1,
      "aditionalQuantity": 0,
      "probabilitySpawn": 50,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "corner",
      "linkRoomName": "",
      "fixedType": "subObjective",
      "fixedRotation": -1,
      "objectiveNumber": 0,
      "minQuantity": 0,
      "aditionalQuantity": 15,
      "probabilitySpawn": 50,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    },
    {
      "roomName": "line",
      "linkRoomName": "",
      "fixedType": "subObjective",
      "fixedRotation": -1,
      "objectiveNumber": 0,
      "minQuantity": 0,
      "aditionalQuantity": 10,
      "probabilitySpawn": 10,
      "areaSpawnSuperior": [ 0, 0 ],
      "areaSpawnInferior": [ 24, 24 ]
    }
  ]
}
```

Capítol 15

Manual d'usuari i/o instal·lació

15.1 Instal·lació

El videojoc desenvolupat consisteix en un simple executable per a "Windows" el qual no requereix instal·lacions, únicament disposar de l'executable. Per a l'execució s'ha d'iniciar l'executable, escollir les opcions gràfiques i prémer el botó de "Play!". Això obrirà el joc en el menú principal.

15.2 Objectiu del joc

En aquest joc, el jugador controlarà un robot i haurà d'arribar a la sala final d'un nivell laberíntic conformat per sales distribuïdes de manera aleatòria.

Les portes de les sales es podran obrir si el jugador compta amb un nivell d'accés adequat (es podrà accedir aquelles sales que tinguin un nivell igual o inferior que el nivell d'accés del jugador més un). A cada nova sala que s'entri, apareixeran enemics que buscaran al jugador i intentaran matar-lo, fent que així perdi la partida. El jugador haurà de derrotar a tots els enemics per a poder sortir de la sala i continuar avançant.

La partida acabarà quan el jugador interactuï amb el tele-transportador de nivell que contindran les sales finals i podran ser interactuats en completar-les.

15.3 Controls

Aquest videojoc funciona amb dos esquemes de control, teclat i ratolí o controlador, tant per a la navegació dels menús com pel control del personatge. L'esquema de controlador admet qualsevol controlador compatible amb PC.

15.3.1 Menús

Tots els menús consisteixen en un seguit d'opcions distribuïdes en vertical, de les quals únicament se'n pot escollir una.

- Selecció: Permet escollir una opció la qual es ressaltarà canviant de color respecte a la resta.

  "Inputs":

  - Teclat i ratolí: Tecles "W" (amunt) i "S" (avall), Tecles de fletxes direccionals "amunt" i "avall" o posar el cursor sobre l'opció.
  - Controlador: Amunt i avall amb "joystick" esquerra, "joystick" dret o "D-Pad"

- Confirmació: Es confirma la selecció de l'opció seleccionada.

  "Inputs":

  - Teclat i ratolí: Tecla "Enter" o clic esquerra del ratolí sobre l'opció.
  - Controlador: Botó sud.

15.3.2 Personatge

Durant la partida, el jugador controlarà un personatge amb el qual podrà realitzar les següents accions:

- Moviment: Permet moure el personatge en la direcció indicada per l'"input".

  "Inputs":

  - Teclat i ratolí: Tecles "W,A,S,D" (amunt, esquerra, avall i dreta).
  - Controlador: Direcció del "joystick" esquerra.

- Apuntar: Permet fer que el personatge miri en la direcció indicada per '"input".

  "Inputs":

  - Teclat i ratolí: Direcció del cursor del ratolí.
  - Controlador: Direcció del "joystick" dret.

- Seleccionar arma per defecte: Selecciona l'arma per defecte.

  "Inputs":

  - Teclat i ratolí: Tecla "1".
  - Controlador: Botó nord.

- Seleccionar escopeta carregada: Selecciona l'escopeta carregada.

  "Inputs":

  - Teclat i ratolí: Tecla "2".
  - Controlador: Botó sud.

- Seleccionar franctirador carregat: Selecciona el franctirador carregat.

  "Inputs":

  - Teclat i ratolí: Tecla "3".

  - Controlador: Botó est.

- Disparar (arma seleccionada): Segons el tipus seleccionada funcionarà de manera diferent. Aquests són els diferents comportaments:

  - Arma per defecte: S'anirà disparant mentre es mantingui sostingut l'"input" i quedin bales en el cartutx.

  - Escopeta carregada: Mantenir premut l'"input" donarà lloc a un procés de càrrega el qual, un cop completada, dispararà una rafega de bales.

  - Franctirador carregat: Mantenir premut l"'input" donarà lloc a un procés de càrrega el qual, un cop completada, permetrà que es pugui deixar de prémer l'"input" per tal de realitzar el tret.

  "Inputs":

  - Teclat i ratolí: Tecla "space" o clic esquerra del ratolí.

  - Controlador: "Trigger ZR".

- Carregar arma seleccionada: Carrega l'arma seleccionada, si aquesta no disposa de munició de cartutx il·limitada i queda munició de magatzem. L'arma sempre s'intentarà carregar al màxim de la munició de cartutx, sempre que quedi prou munició de magatzem.

  "Inputs":

  - Teclat i ratolí: Tecla "R".

  - Controlador: Botó oest.

- Habilitat de tele-transport: Mantenir premut l"'input" deshabilita els "inputs" relacionats amb l'arma i projecta un indicador cap endavant del personatge fins a una distància limitada (sense travessar parets). Aquest indicador seguirà els moviments i direcció del personatge. Un cop es deixi de prémer l"'input" el personatge es tele-transportarà cap a la posició de l'indicador.

  "Inputs":

  - Teclat i ratolí: Tecla "Q".

  - Controlador: "Trigger ZL".

- Interaccionar: Si el personatge es troba dins d'un element interacciona-ble (un tele-transportador) s'interactuarà amb ell de manera prioritària. Altrament, s'interactuarà amb l'element més proper en la direcció en la qual apunti el personatge (amb una distància molt limitada).

  "Inputs":

  - Teclat i ratolí: Tecla "F".

  - Controlador: Botó "R".

- Pausar la partida: Bloqueja tots els altres "inputs" del personatge, pausa el joc i obre el menú de pausa. Tornar a prémer aquest "input" reverteix els efectes anteriors.

  "Inputs":

  - Teclat i ratolí: Tecla "Escape"'
  - Controlador: Botó "Start".