TecnoCampus

Centre universitari adscrit a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

Desenvolupament d'una eina per la creació de controladors per combats Souls-like

MEMÒRIA FINAL

Guillem Llovera Castillo

Tutor: Dr. Adso Fernández Baena

2022-2023

TecnoCampus Mataró-Maresme

Dedicatòria

Al Roger Llovera Castillo, el meu germà gran, per ser la meva major inspiració de la meva vida en l'àmbit professional i personal.

Agraïments

A la meva meravellosa parella i als meus amics més propers per acompanyar-me durant aquesta experiència.

A l'Adso Fernández Baena per ser un excel·lent tutor.

I a la meva família per donar-me suport sempre i en especial al meu germà gran per ajudar-me durant tota la carrera amb un entusiasme contagiador.

Abstract

This project aims to create a tool, intended for designers, to create and customize a third-person character controller with combat mechanics for Souls-likes. To demonstrate the results, an enemy has been created to fight.

Resum

Aquest treball consisteix a crear una eina, pensada per a dissenyadors, que permeti crear i personalitzar un controlador de personatge en tercera persona amb mecàniques de combat per a Souls-likes. Per fer una demostració dels resultats s'ha creat un enemic amb el qual combatre.

Resumen

Este trabajo consiste en crear una herramienta, pensada para diseñadores, que permita crear y personalizar un controlador de personaje en tercera persona con mecánicas de combate para Souls-likes. Para hacer una demostración de los resultados se ha creado un enemigo con el que combatir.

Índex

Índex de figures

Índex de taules

1. Introducció

El gènere de videojocs Souls-like ha adquirit un augment de popularitat molt gran en els últims anys, en maig de 2020 la saga Dark Souls havia aconseguit 27 milions de vendes i avui en dia Elden Ring ha aconseguit més de 20 milions de vendes (J. Clement, 2023). Aquest augment ha influenciat a les mecàniques de combat cos a cos de molts altres videojocs en tercera persona, a part d'augmentar també la demanda de videojocs del gènere.

Una característica important en els videojocs d'acció és el terme popularitzat per Steve Swink (2008), conegut com a game feel, aquest element fa que el jugador se senti més dintre del joc i que certes accions, sobretot les que es fan de manera repetitiva, siguin més satisfactòries de fer. El game feel també pot servir com a element de feedback per al jugador perquè entengui que està succeint en el joc. Segons Swink (2008) per poder considerar que un videojoc té game feel hi ha tres pilars a tenir en compte, el control a temps real, l'espai simulat i el polit.

Les eines de desenvolupament poden facilitar i agilitar la producció, modificació i manteniment de característiques (Margaret Rouse, 2020). L'objectiu principal d'aquest treball és desenvolupar una eina que supleixi de suficients elements per, des de zero, poder crear un controlador de personatge, amb les mecàniques bàsiques de combat dels Souls-like, sense necessitat de saber com escriure codi.

Per un correcte desenvolupament de l'eina s'ha estudiat què són els controladors de càmera, controladors de personatge, els combats en videojocs en tercera persona i que defineix un bon game feel.

Per veure exemples de controladors de personatges publicats en línia s'han analitzat tres controladors fets per diferents autors per veure els elements comuns i els que es troben a faltar de cada un.

Un cop fet l'estudi previ s'ha dividit el desenvolupament en fases i s'ha establert una metodologia i un cronograma de treball.

Abans de començar el desenvolupament de l'eina s'han analitzat diferents jocs Souls-like per veure quines mecàniques de combat són similars en tots els jocs i així poder veure quins són els aspectes fonamentals del combat en aquest gènere. Un cop aconseguit les mecàniques a programar i els seus paràmetres de disseny s'ha desenvolupat l'eina en el motor gràfic *Unity* (Unity Technologies, 2005). Finalment per validar les funcionalitats de l'eina i del controlador desenvolupat s'ha creat un personatge no controlat pel jugador amb el qual poder combatre.

2. Objectius

En aquest apartat es defineixen els objectius d'aquest treball, aquests objectius s'han dividit per poder marcar cada un com una fase del desenvolupament.

2.1 Objectiu principal

Desenvolupar una eina que permeti la creació i personalització d'un controlador en tercera persona centrat en el combat, de tipus Souls-like, només fent servir la finestra de l'eina.

2.2 Objectius secundaris

I. Analitzar la càmera, les mecàniques, el controlador de personatge i el game feel de videojocs del gènere Souls-Like.

II. Dissenyar el funcionament d'un controlador i tots els paràmetres necessaris un dissenyador voldria poder modificar a l'hora de crear-ne un de zero.

III. Programar en *Unity* el controlador de personatge dissenyat a partir de l'anàlisi prèvia.

IV. Crear una interfície d'usuari senzilla i fàcilment comprensible per la personalització del controlador de personatge.

V. Validar les funcionalitats del controlador i de l'eina creant un personatge no jugable amb el que poder combatre.

VI. Crear vídeos tutorials com a documentació per l'ús correcte de l'eina.

3. Marc Teòric

En aquest capítol s'explica els fonaments teòrics necessaris per poder entendre els conceptes més bàsics a l'hora d'exposar l'anàlisi de referents i el desenvolupament de l'eina. Aquests fonaments expliquen que és una càmera virtual i un controlador de personatge, les mecàniques de combat en els videojocs en tercera persona i la definició de Game Feel.

3.1 Càmera i controlador de personatge

Des de la creació del primer videojoc de computadora, *Spacewar!* (Russell, Graetz, Samson, Witaenem, 1962), que existeix una representació visual del món fictici de joc, l'element dins del joc que mostra aquesta representació visual se la coneix avui en dia com a càmera. Tot i que el 1980 ja es va crear el primer joc considerat 3D, *Battlezone* (Atari, 1980), no va ser fins a 1986 on va aparèixer el concepte de càmera en els videojocs (Krichane, 2021).

Segons Krichane (2021) el primer videojoc que va fer servir el terme càmera va ser *Starglider* (Argonaut Games, 1986), en el qual en el manual d'instruccions del joc es mencionava com a càmera un element del joc que permetia canviar la vista a un objecte diegètic. En *Starglider 2* (Argonaut Games, 1988) no es va fer sevir el concepte càmera en el manual d'instruccions, però la premsa ja feia servir el terme càmera per referir-se a totes les vistes del joc les quals són de fora de la vista cambra on se situa el personatge del jugador. Aquest concepte també s'ha popularitzat gràcies al món del cinema, per la qual cosa es pot considerar al jugador o a la càmera que mostra l'acció com si fos un director de cinema.

La càmera en un videojoc pot ajudar no només a mostrar el que està succeint en el món fictici, sinó també elements que afecten directament a la càmera, com veure gotes en cas que plogui o un sacseig de càmera quan cau un arbre a terra, aquests elements poden millorar la immersió del jugador en el joc.

3.1.1 Funcionament d'una càmera virtual

Per poder mostrar un objecte en una càmera virtual en 3D es necessita primer tenir la posició, rotació i escala de la càmera. Després, per poder mostrar un objecte, s'ha de calcular on està cada vèrtex de l'objecte en posició local i posteriorment aplicar a cada vèrtex del model de l'objecte la transformació de la matriu de món (posició, rotació i escala) (Code Labs 2013).

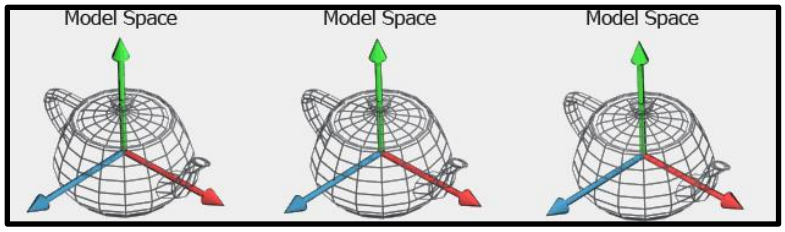

Figura 3.1. Visualització d'objectes en coordenades locals. Font: Coding Labs.

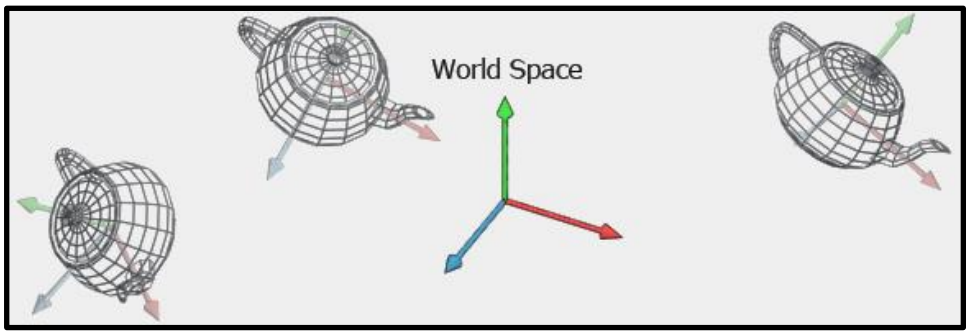

Figura 3.2. Visualització d'objectes en coordenades de món. Font: Coding Labs.

Després, junt amb la matriu inversa de la càmera s'obté la matriu de vista (Code Labs 2013).

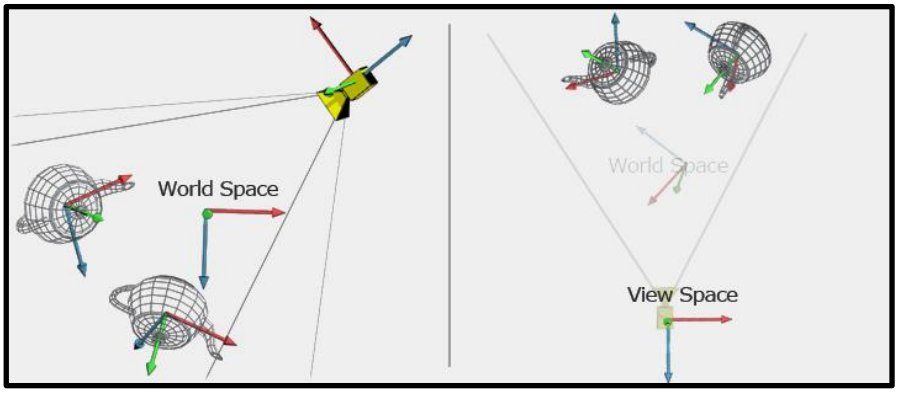

Figura 3.3. Visualització d'objectes i càmera en coordenades de món i visualització d'objectes i el punt 0,0,0 del món en coordenades de càmera. Font: Coding Labs.

Amb la matriu de vista s'aplica el punt focal, la relació d'aspecte de la pantalla, la profunditat mínima i màxima de renderitzat per aconseguir la matriu de projecció (Code Labs 2013). Aquestes profunditats es coneixen com a "clipping planes", les quals permeten excloure geometria de la vista de la càmera on el pla més proper és el near clipping plane i l'altre far clipping plane (3ds Max Documentation).

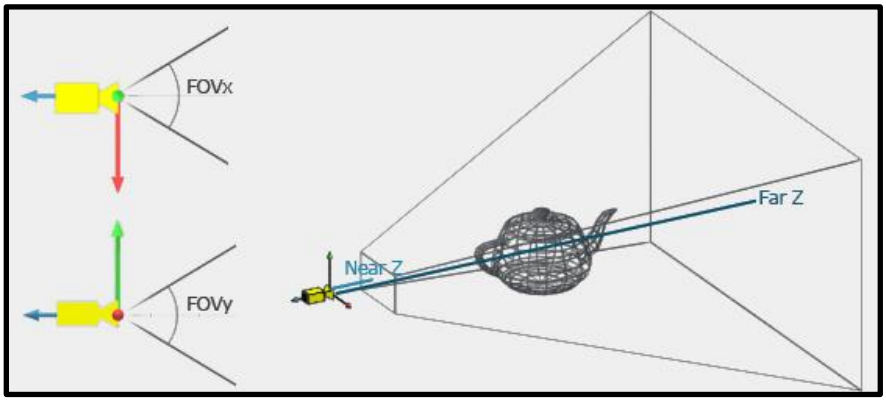

Figura 3.4. Visualització d'objectes en coordenades de càmera en la matriu de projecció en una càmera en perspectiva. Font: Coding Labs.

En una càmera ortogràfica, on no es diferencien profunditats, l'últim pas fet anteriorment es substitueix a només tenir en compte el near clip plane, el far clip plane, l'alçada i l'amplada de la càmera (Code Labs 2013).

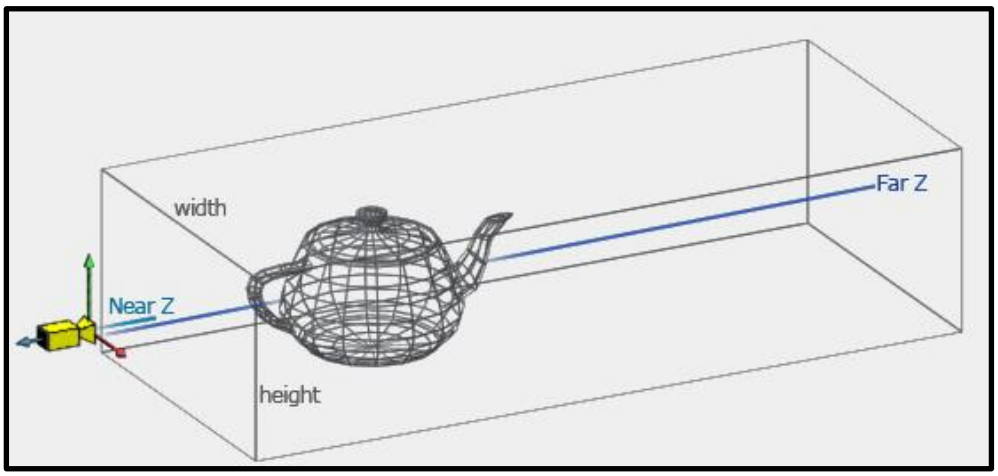

Figura 3.4. Visualització d'objectes en coordenades de càmera en la matriu de projecció en una càmera en perspectiva. Font: Coding Labs.

En un motor gràfic, com és per exemple *Unity*, per saber l'estat físic d'un objecte es fa servir un component conegut com a *Transform* que conté la informació de la

matriu transformació dels objectes, aquesta matriu conté la informació de la posició, rotació i escala (Unity Spripting API - Transform). La posició, rotació i escala es mostren en coordenades 3D (x,y,z), aquestes coordenades es coneixen com a *Vector3*.

Per rotar una càmera en un món tridimensional es fa servir el *Vector3* de rotació del transform. Per modificar aquests valors en codi s'ha estandarditzat en el món dels videojocs la nomenclatura d'aviació, on rotar en el pla perpendicular a les ales es coneix com a "yaw", el pla perpendicular al yaw i paral·lel al pla de les ales es coneix com a "pitch" i el pla restant que segueix del centre al morro de l'avió es coneix com a "roll". És a dir, si en *Unity* agafes el *Vector3* de rotació de la càmera per canviar la "x" en codi és amb el yaw, la "y" és el pitch i la "z" és el roll.

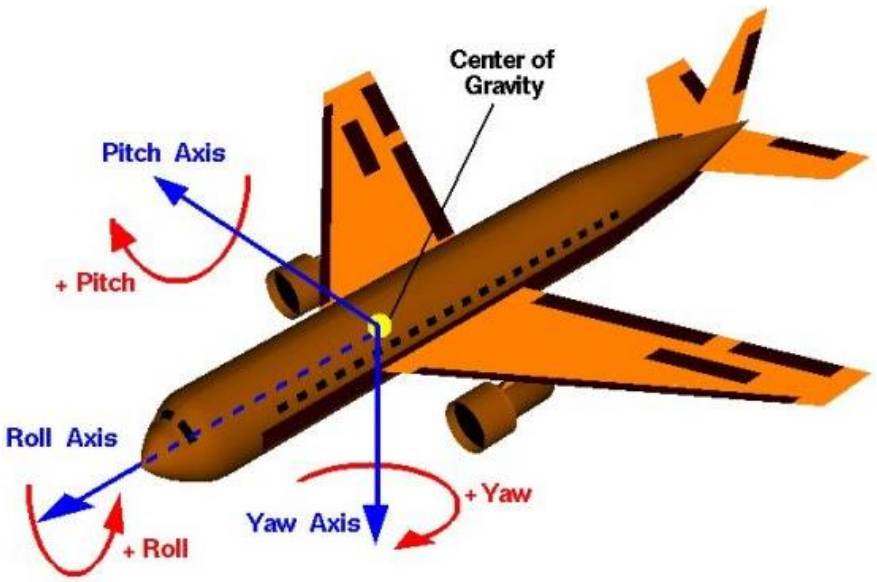

Figura 3.5. Nomenclatura de rotació dels eixos d'un avió. Font: Glenn Research Center - NASA.

3.1.2 Tipus de càmera: primera i tercera persona

Existeixen dos tipus de càmera en el món dels videojocs depenent de la vista de la càmera: la primera persona i la tercera persona. Una càmera en primera persona representa el que veu l'avatar que s'està controlant, o sigui que els ulls del personatge són la càmera i el que veu el personatge és el mateix que veiem nosaltres (Rouse, 1999).

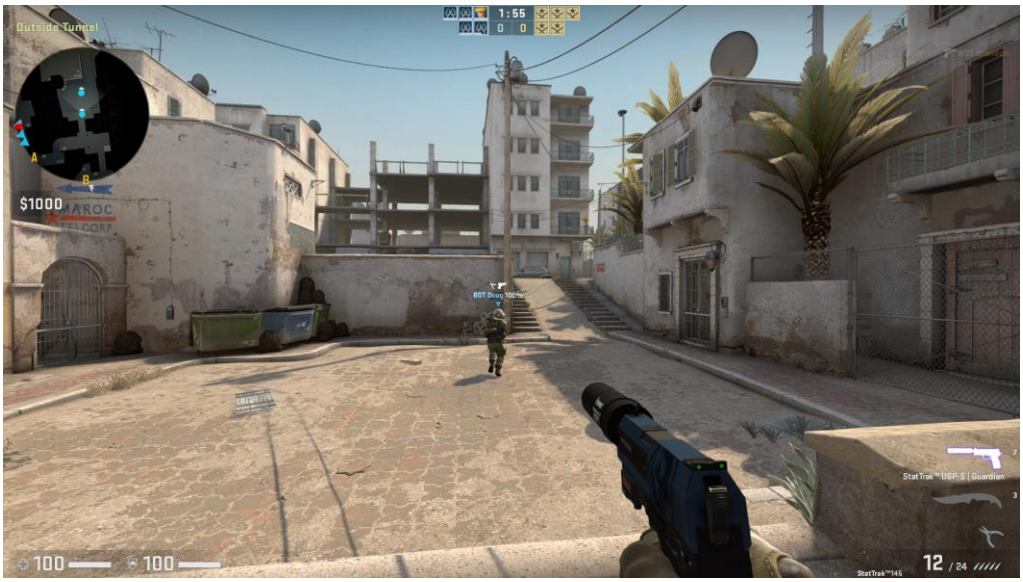

Figura 3.5. Visualització de la càmera en primera persona del *Counter-Strike: Global Offensive*. Font: Valve Corporation, 2012.

La tercera persona és tota aquella càmera on es pot veure l'espai simulat sense necessitat de fer-ho a través dels ulls d'un personatge dintre de la ficció. Normalment, la tercera persona es fa servir per videojocs on es vol veure el voltant de l'avatar ja sigui per tenir millor visibilitat del que està succeint al voltant o per poder mostrar com el nostre avatar interactua millor amb l'entorn (Rouse, 1999).

Figura 3.6. Visualització de la càmera en tercera persona del *Dead By Daylight*. Font: Behaviour Interactive, 2016.

Els videojocs 2D són, en la gran majoria, videojocs en tercera persona, per la limitació d'haver de posar tot en un mateix pla. Encara que és veritat que en un videojoc 2D es podria arribar a fer la primera persona, no se sol veure. Un cas d'un videojoc en primera persona 2D és el Doom (1993), on se simula un espai 3D representat en 2D, la profunditat de l'escenari canvia l'escala de cada textura en la càmera per donar sensació de profunditat (Sanglard, 2010).

Figura 3.7. Visualització de la càmera en el *DOOM*. Font: id Software (1993)

L'ús d'una càmera en tercera persona té avantatges i desavantatges (Gorisse, Christmann, Amato i Richir, 2007):

Avantatges:

- Millor visibilitat i percepció de l'entorn del voltant de l'avatar (Gorisse, *et al*., 2007).
- Facilita la interacció amb elements en moviment (Gorisse *et al*., 2007).
- És menys probable que un jugador pateixi de cinetosi, ja que les càmeres en tercera persona acostumen a tenir un moviment més suau i tot està prou lluny per a no arribar a experimentar mareig (Rouse, 1999).
- Es pot mostrar un avatar protagonista amb més personalitat, ja que el jugador el veu constantment mentre juga i pot generar molta més empatia amb el protagonista (Rouse, 1999).

- Es poden fer unes animacions més impressionants i de manera més cinemàtica que en un videojoc en primera persona (Rouse, 1999).

Desavantatges:

- La primera persona ofereix més facilitat per veure i interactuar amb objectes petits de l'entorn (Gorisse *et al*., 2007).

- S'han de cuidar molt les col·lisions de la càmera i polir molt el moviment de la mateixa perquè no faci moviments bruscs. En col·lidir amb un objecte sòlid és complicat programar una resolució bona per totes les situacions possibles (Rouse, 1999).

- Les càmeres en tercera persona acostumen a tenir problemes amb espais petits, ja que la càmera estarà constantment col·lidint amb les parets (Rouse, 1999).

3.1.3 Control de personatge en jocs en tercera persona

Segons Rogers (2014) depenent del gènere de videojoc hi ha diferents tipus de càmera:

- Una càmera fixa, on depenent de la posició del jugador la càmera enfoca des d'una posició predefinida pels desenvolupadors del joc i amb una rotació nul·la o fixada amb l'avatar. Un videojoc que fa servir aquesta càmera és Resident Evil (Capcom 1996).

Figura 3.7. Visualització d'una càmera fixa en Resident Evil. Font: Capcom, 1996

- Una càmera on el moviment de la càmera ve ancorada a l'avatar que controla el jugador i aquesta càmera òrbita de manera esfèrica a l'avatar, aquest tipus de càmera se sol fer servir molt en jocs d'acció per poder mostrar des de prop com es mou l'avatar i les animacions que fa. Segons Jason Rutter i Jo Bryce el videojoc que va popularitzar les càmeres d'aquest tipus va ser el videojoc Super Mario 64 (Nintendo 1996).

Figura 3.8. Visualització d'una càmera en òrbita al jugador en Super Mario 64 (Nintendo). Font: Nintendo (1996)

- Una càmera en vista isomètrica que pot seguir constantment a l'avatar controlat o també pot ser de mobilitat lliure controlada pel jugador. És una càmera més distant que sempre enfoca a terra i permet una millor visibilitat d'allò que passa dins de l'abast de la càmera, aquest tipus de càmera acostuma a funcionar molt bé amb controls "point and click".

Figura 3.9. Visualització d'una càmera de vista isomètrica en Diablo III. Font: Blizzard Entertainment, 2012.

Rogers (2014) també exposava que un cop es té el tipus de càmera s'ha d'escollir el tipus de control que tindrà el jugador sobre l'avatar, si existeix. Els controls poden ser:

- Relatius al personatge: L'avatar intentarà seguir la direcció donada per l'usuari tenint només en compte la posició de l'avatar, és a dir que un input de moure's endavant fa que l'avatar es mogui cap on ell està mirant, independentment d'on estigui enfocant la càmera.

- Relatius a la càmera: L'avatar intentarà seguir la direcció donada per l'usuari relatiu a la càmera. Si un usuari fa un input cap endavant, l'avatar es mou cap on enfoca la càmera i en cas de fer un moviment cap a la dreta o l'esquerra el personatge es mou en perpendicular a la càmera.

3.2 Mecàniques de combat en els videojocs

En el món dels videojocs el concepte mecànica serveix per definir una funcionalitat d'un videojoc, que va associada a un disseny i unes normes establertes per un dissenyador. Aquestes dicten les possibles accions dels jugadors (Tyler, 2023). En aquest apartat s'explica les normes del combat en els videojocs i també més específicament les normes en els combats dels Souls-like.

3.2.1 Característiques de combat

L'inici del combat en videojocs es va originar en el videojoc 2D de consola domèstica *Karate Champ* (Technōs Japan, 1984), on dos jugadors combatien l'un contra l'altre controlant un karateca que disposava d'una varietat d'atacs. De fet, el gènere de videojocs de combat es nombrava com "karate game" llavors del "fighting game" que estem acostumats a escoltar avui en dia (Ketonen, 2016).

Figura 3.10. Imatge del videojoc *Karate Champ* amb el jugador 1 a l'esquerra i el jugador 2 a la dreta. Font: *Technōs Japan*, 1984.

Els videojocs de combat tant els de 2D com els 3D comparteixen un element fonamental per una bona interacció entre dos combatents. Ketonen (2016) diu que per un combat equilibrat es requereix el fet que els atacs de cada combatent tingui 3 parts, la primera és l'inici de l'atac o anticipació, on l'atacant comença l'atac, deixant així un temps a l'altre per reaccionar, la segona part és la part on l'atac pot fer mal al contrincant en impacte amb l'usuari i la tercera part és la part on l'atac ha acabat i l'atacant es recupera de l'atac, normalment aquest últim pas és on el contrincant pot aprofitar per castigar un atac fallat.

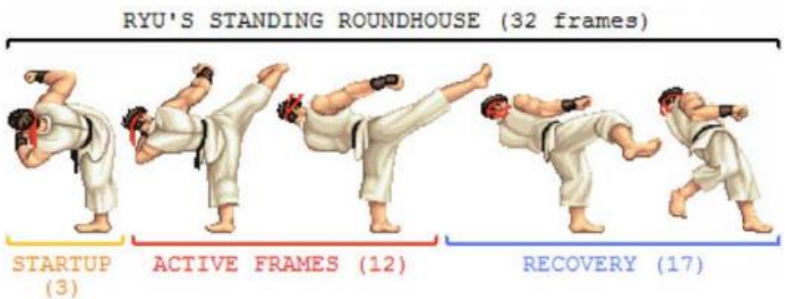

Figura 3.11. Representació de les tres fases d'un atac de Ryu de *Street Fighter 2* (Capcom, 1991). Font: Ketonen (2016).

3.2.2 Combat en els Souls-like

Els Souls-like són coneguts per la seva alta dificultat. En els combats dels Souls-like els enemics acostumen a poder derrotar al jugador en pocs atacs. L'element més important del combat Souls-like és el recurs conegut com a resistència, la qual disminueix cada cop que s'efectua una acció i requereix ser major de zero per poder continuar efectuant accions. Aquest recurs es va regenerant pel temps, podent així els dissenyadors tenir un millor control del ritme esperat dels combats. Aquest element també ajuda a connectar les diferents mecàniques de combat amb un element en comú (Guzsvinecz, 2022).

Figura 3.12. Barra de resistència de *Dark Souls III*, el verd és resistència restant, el groc la gastada anteriorment i en transparent la que falta. Font: FromSoftware, 2016.

3.3 Game feel

Game feel, també conegut com a "game juice", sempre ha sigut un terme amb definició confusa, ja que diferents persones del sector dels videojocs l'entenen de manera diferent (Swink, 2008).

3.3.1 Definició de game feel

El terme "Juicy game" es va veure i definir en un article que es diu "How to Prototype a Game in Under 7 Days" de Matt Kucic (2005), allà es va definir com a resposta del joc a accions de l'usuari i que el joc se senti viu perquè fa que el jugador se senti poderós i en control del món virtual.

Steve Swink en una publicació feta el 2007, "Game Feel: The Secret Ingredient" va donar nom al concepte de Game Feel i va dividir el procés d'aconseguir un bon game feel en sis mètriques: Input, resposta, context, polit, metàfora i regles.

En la publicació del llibre "Game Feel: A Game Designer's Guide to Virtual Sensation" Steve Swink (2008) va aconseguir una definició més clara sobre aquest tòpic i va popularitzar el terme game feel. Aquesta definició és la més feta servir en la indústria dels videojocs i la que s'ha fet servir en aquest treball. "Mai s'ha definit col·lectivament el game feel per sobre del que és necessari per parlar d'un joc específic" (Swink, 2008).

3.3.2 Tres pilars principals de game feel segons Steve Swink

Swink, en el seu llibre, parla de com totes les definicions de game feel que persones de la indústria li han donat es poden considerar com a vàlides, però molt variades, va posar en comú aquestes definicions i va extreure el que ell va nomenar com els tres pilars principals del game feel.

3.3.2.1 Control a temps real

Swink (2008) defineix el control a temps real com la interacció d'un usuari amb la computadora fent un bucle tancat d'input-resposta, quan l'usuari fa una acció la computadora ho rep, mostra el resultat de l'input i l'usuari reacciona al seu input i pot decidir fer-ne un altre i així fins a acabar la sessió de joc.

Chris Cawford (2003) va parlar en el seu llibre "Chris Crawford on Game Design" de com les converses entre dos agents actius escolten a l'altre, processen la informació i responen amb el que volen dir, un cop fet aquesta interacció l'altra persona repeteix el procés però llavors de receptor com a emissor.

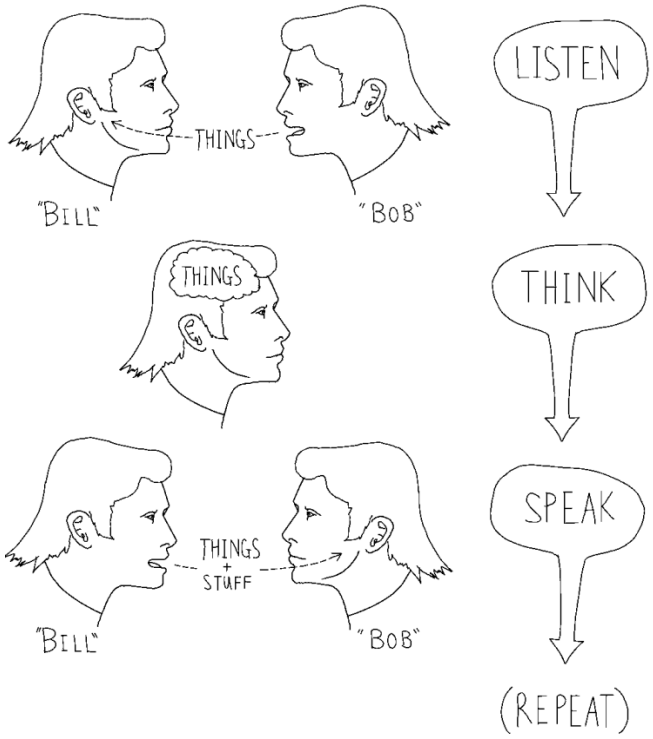

Figura 3.13. Representació visual de la interacció entre dos agents. Font: Swink, 2008.

En el model de Cawford (2003) un dels agents és substituït per una computadora, computadora que "escolta" els inputs del jugador a través d'un dispositiu d'entrada, "processa" l'efecte que han tingut els inputs del jugador i "parla" mostrant per pantalla i altres perifèrics de sortida el resultat d'aquesta interacció.

Swink (2008) parla de com no li agrada molt el model de Cawford, ja que en un videojoc amb control a temps real no se sent tant com una conversa sinó com podria ser conduir un cotxe, on més que un usuari fer una acció i esperar la reacció moltes accions ja les fa l'usuari sense haver de pensar, el pas de processar la informació es fa de manera inconscient. Swink (2008) parla d'aquest fet com que el resultat de l'input es percep al mateix moment que s'està efectuant, aquest resultat es pot definir com el control continu d'un avatar en moviment.

No tots els videojocs necessiten controls a temps real, però els jocs que els necessiten han de complir unes normes perquè els notem, llavors la pregunta que surt aquí és "Quan sabem que un joc té control a temps real?". Card, Moran i Newell (1986) van publicar un article conegut com "The Model Human Processor" en el qual

van estipular que la mitja mínima de temps que una persona tarda a processar un input pels seus ulls és de 240 mil·lisegons, parlaven que l'ésser humà passa per tres processos en aquests 240 mil·lisegons i que cada un tarda temps diferents.

- Procés perceptiu (entrada d'inputs pels sentits): ~100 ms [50–200ms]
- Procés cognitiu (decideix que fer amb la informació): ~70 ms [30–100ms]
- Procés motor (ordena als músculs una resposta): ~70 ms [25–170ms]

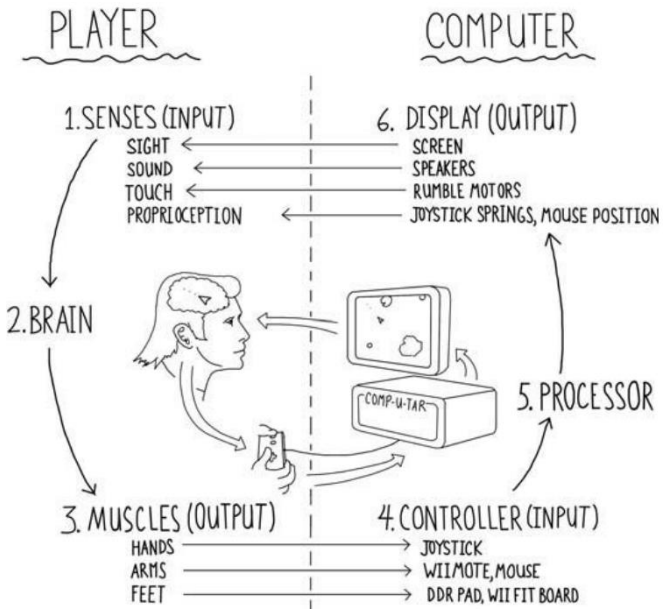

Figura 3.14. Representació visual de la interacció entre jugador i computadora. Font: Swink, 2008.

Un cop es té aquest concepte clar s'ha d'adaptar el joc perquè en aquests temps l'usuari tingui accés a un cercle de correcció, o sigui que en qualsevol moment pugui reaccionar al que està processant i poder efectuar un canvi, en el cas de conduir si un gir s'ha efectuat massa tancat el conductor rectificarà i s'obrirà més per no xocar (Swink, 2008).

També per part de la computadora hi ha tres necessitats que s'han de tenir per poder obtenir aquest control en temps real (Swink, 2008):

- Impressió de moviment: Els fotogrames per segon necessaris que necessita una persona per poder percebre un moviment ha de ser igual o major a 10

fotogrames per segon, com més fotogrames millor impressió de moviment (Swink, 2008).

- Resposta instantània: Un ordinador o consola no pot mostrar una resposta instantània, ja que hi ha un temps també de processament i mostra que tarda uns quants mil·lisegons, però és important que la resposta es rebi abans que acabi el cicle de processament d'informació (240 ms). Per poder percebre la resposta com instantània s'ha de tenir una resposta en la meitat del temps del procés perceptiu (50 ms), si dura el mateix que el procés perceptiu (100 ms) es pot percebre el retard, però és ignorable per l'usuari i a més de 200 ms la resposta se sent lenta i no fa sensació de control a temps real (Swink, 2008).

- Continuïtat de la resposta: L'ordinador o consola ha d'actualitzar les respostes constantment en un menor temps que el procés perceptiu de l'usuari, o sigui en menys de 100 mil·lisegons (Swink, 2008).

3.3.2.2 Espai simulat

Swink (2008) va dir que per tenir una sensació de control a temps real és necessari l'existència d'un espai simulat, en aquest espai simulat s'han de veure les interaccions de l'avatar controlat pel jugador amb l'entorn, les col·lisions i les interaccions físiques de l'espai simulat. En aquest punt és important el disseny del nivell que es mostra, si volem deixar clar aquest espai simulat l'escenari on es pot moure l'avatar del jugador ha de tenir objectes que reaccionin amb la interacció amb l'avatar esmentat, això ens fa sentir que és un món tàctil i reactiu.

3.3.2.3 Polit

El polit d'un videojoc és la capa final d'acabat d'un videojoc, els elements d'aquesta capa són addiccions artificials que no afecten el funcionament del joc, però que milloren la interacció de l'avatar amb l'entorn i fan el joc més complet i convincent. Aquests elements poden ser des de partícules de pols en l'entorn, fins a sons ambient o unes transicions entre animacions més suaus i naturals (Swink, 2008).

3.3.3 Classificació de game feel

Per poder considerar que un joc té game feel total, Swink (2008) va estipular tres preguntes a l'hora d'analitzar un videojoc:

1. Té control en temps real?
2. Té un espai simulat?
3. Té polit?

Amb aquestes preguntes es poden extreure set tipus diferents de game feel en videojocs que va mostrar amb aquest diagrama.

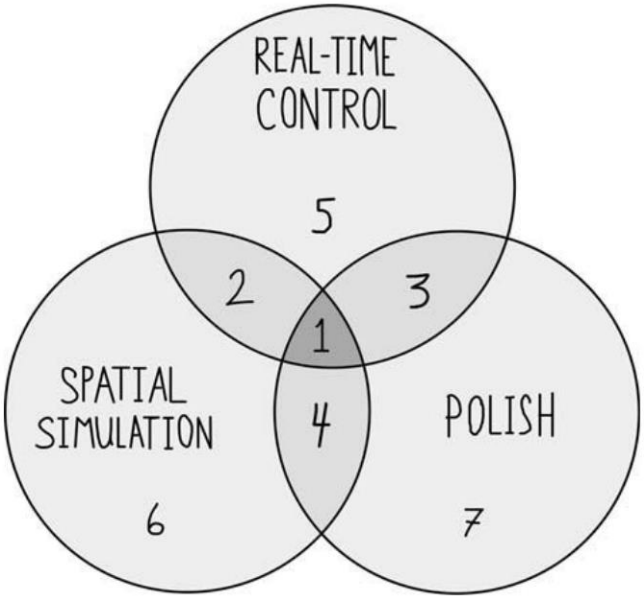

Figura 3.15. Diagràma dels diferents tipus de game feel a partir dels tres pilars del game feel. Font: Swink, 2008.

Els set tipus diferents de game feel en videojocs es poden definir d'aquesta manera:

1. Els videojocs que compleixen amb els tres pilars del game feel són els únics jocs que Swink (2008) contempla en el seu llibre, tot altre joc que no entri dins d'aquesta categoria no són considerats jocs amb game feel de veritat. La gran majoria de videojocs del mercat entren dintre d'aquesta categoria (Swink 2008).

2. Els videojocs que tenen un espai simulat i uns controls a temps real, però que manquen de components de polit, com podria ser so o animacions. És estrany veure videojocs en aquesta categoria, ja que la majoria que surten al mercat tenen alguna part de polit per mínim que sigui (Swink, 2008).

3. Els videojocs que tenen un control a temps real i un polit, però no es percep l'espai simulat. Aquest tipus de videojocs no acostumen a existir, pel fet que, si es fa un avatar amb controls a temps real, el videojoc requereix forçadament un espai on poder aplicar aquests controls (Swink, 2008).

4. Els videojocs que tenen espai simulat i polit són aquells en els que pot no haver-hi un cicle de correcció activa, pel qual encara que el joc pot ser responsiu no entra dins de la definició de control a temps real (Swink, 2008).

5. Els videojocs amb només control a temps real no solen existir per la mateixa raó que s'ha explicat en el punt 4 (Swink, 2008).

6. Els videojocs en el qual es pot percebre un espai simulat, però no hi ha cap mena de control en temps real o polit (Swink, 2008).

7. Els videojocs amb només polit poden existir, ja que l'espai simulat pot ser abstracte, sense necessitat de col·lisions o fins i tot un avatar a controlar. En aquest joc el polit és el punt més fort i el control a temps real pot ser parcial, és a dir, que hi hagi una resposta immediata encara que no tingui cicle de correcció (Swink, 2008).

3.3.4 Mètriques de game feel

En l'article de 2007 Swink va definir el game feel amb 6 mètriques que més tard va desenvolupar a fons en el llibre de 2008.

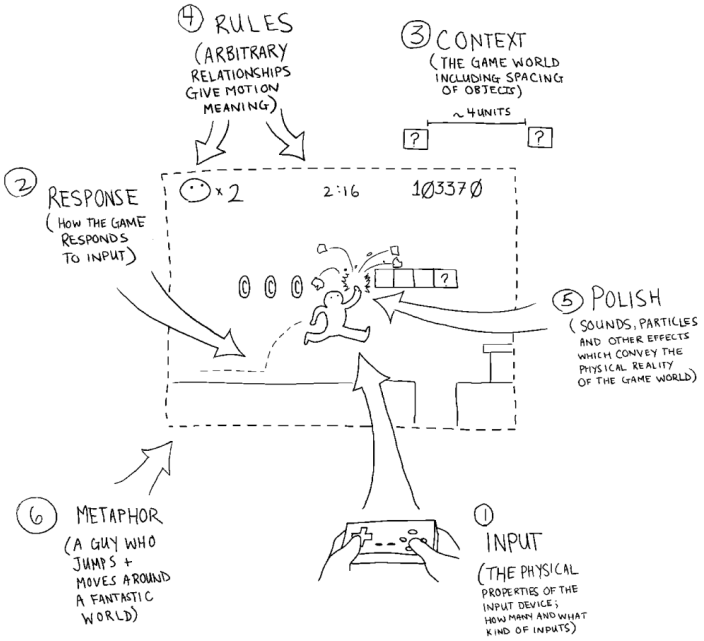

Figura 3.16. Representació del concepte de les sis mètriques del game feel. Font: Swink, 2008.

1. Entrada de l'usuari o "input" -- Són el dispositiu que fa servir l'usuari per comunicar-se amb el joc, per exemple un comandament d'Xbox que és molt còmode i satisfactori de fer servir, i també són els controls que s'han de fer servir.

2. Resposta -- Com el sistema de joc reacciona als inputs del jugador, ja sigui la fluïdesa del personatge a controlar o la sensibilitat que es necessita en un joystick per detectar moviment.

3. Context -- Com són d'importants les capacitats que té el jugador, o sigui el resultat de l'input i la resposta, amb l'entorn presentat.

4. Polit -- L'afegit de sons, efectes visuals i animacions que acompanyin les accions del jugador per donar una millor satisfacció visual o sonora.

5. Metàfora -- És la part del joc que depèn de les expectatives del jugador, si es té un element que s'espera que funcioni d'una manera ha de funcionar d'aquella manera.

6. Regles -- Posar múltiples objectius, tant petits com grans, i buscar tenir interaccions divertides en el joc.

4. Referents

Els referents que s'han tingut en compte a l'hora de crear aquest treball han sigut tres eines que es poden adquirir de manera gratuïta a la Asset Store de *Unity* (https://assetstore.unity.com/) i que s'han provat en el motor gràfic *Unity*. Aquestes eines estan dirigides a un públic que vol crear videojocs o prototips sense necessitat de saber programar.

4.1 Starter Assets - Third Person Character Controller

Controlador senzill per personatges en tercera persona creat per *Unity Technologies* publicat el 9 de juny el 2021 en:

https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526

Aquest asset ve incorporat amb els packages *InputSystem* i *Cinemachine*, necessaris pel seu funcionament.

L'asset proveeix d'un controlador de personatge molt bàsic, encara que molt polit, amb possibilitat de personalització de variables. Algunes de les variables del jugador són per modificar elements de polit que milloren el game feel.

En l'inspector de *Unity* cada apartat té una capçalera i alguns elements tenen sliders per delimitar els valors dintre d'uns límits recomanats o necessaris, però manca d'un inspector personalitzat.

Aquesta eina està bastant limitada, ja que a partir de cert punt si, es vol tenir més mecàniques que no siguin només moure's i saltar, s'ha d'escriure codi, per la qual cosa és un bon asset per qualsevol usuari que principalment vulgui un personatge que es pugui moure per un escenari, però requereix modificacions de codi en cas de voler fer-ho servir per a un projecte on el jugador tingui més mecàniques.

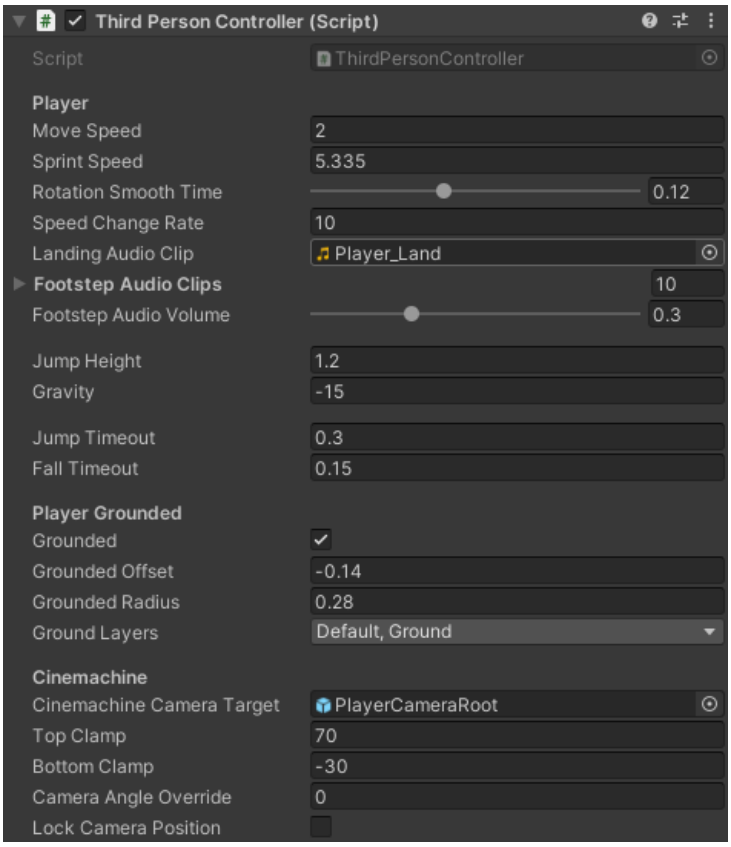

Figura 4.1. Inspector del controlador de l'asset "*Third Person Character Controller*". Font: *Unity Technologies*.

Aquest asset conté:

- Documentació per facilitar l'ús de l'eina.
- Un model humanoide amb un rig complet amb múltiples animacions.
- Un script amb el controlador de personatge.
- Un script de gestió d'inputs de l'usuari.
- Una escena d'exemple.

El codi que ve en aquest asset s'entén fàcilment i té comentaris per explicar el funcionament d'aquest, per la qual cosa és senzill afegir o modificar codi.

4.2 3rd Person Controller + Fly Mode

Controlador creat per Vinicius Marques. Asset enfocat per crear controladors per shooters en tercera persona publicat el 10 de febrer de 2015 en:

https://assetstore.unity.com/packages/templates/systems/3rd-person-controller-fly-mode-28647

Aquest asset ve amb funcionalitats similars a les esmentades en l'anterior apartat amb l'afegit de poder apuntar amb botó dret, mantenint la direcció del personatge cap on mira la càmera, i també un mode extra perquè el personatge controlat pel jugador pugui volar.

A diferència de l'asset anterior aquest no té un inspector personalitzat, totes les variables estan exposades a l'inspector, però sense cap modificació de com es mostren.

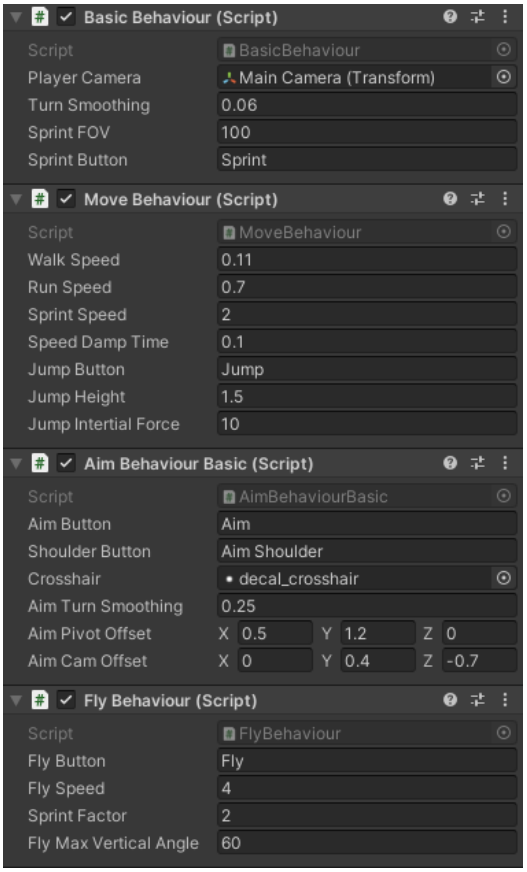

Figura 4.2. Inspector del controlador de l'asset "3rd Person Controller + Fly Mode". Font: Vinicius Marques.

L'asset conté:

- Documentació per facilitar l'ús de l'eina.
- Un model humanoide amb un rig complet amb múltiples animacions.
- Un script que fa com a controlador general de tots els estats i gestiona alguns inputs de l'usuari.
- 3 scripts per cada funcionalitat del personatge, en aquest cas moviment, apuntar i volar.
- Una escena d'exemple.

El principal problema d'aquest asset regeix en el codi. El nom del script que serveix per gestionar els estats té la mateixa nomenclatura que els mateixos estats en si. Un altre problema és que el script de gestió d'estats també gestiona part dels inputs del moviment del personatge, encara que el mateix personatge té un estat de moviment on tot el que va relacionat amb el moviment hauria d'anar allà. La gestió dels estats no funciona realment com una màquina d'estats, per la qual cosa el codi de gestió d'estats no és escalable.

En un shooter on el moviment del personatge sol anar separat de les mecàniques de disparar aquest controlador és molt bo, però en un joc on el combat requereix accions on cada acció és un estat per si mateix, el qual pot restringir o canviar el moviment, requeriria molts canvis per poder arribar a fer-se servir.

Un altre possible problema que podria haver-hi és el fet que el personatge creat amb aquest controlador funciona per Rigidbody llavors de per Character Controller. Fer servir un rigidbody pot donar problemes i té moltes més limitacions que el component Character Controller, l'avantatge de fer servir un Rigidbody és la de no haver d'escriure codi per crear les físiques del personatge, però normalment no es recomana l'ús de Rigidbody en personatges jugables.

4.3 Character Controller SUPER

Controlador creat per Aedan Graves publicat el 8 de febrer de 2019 en: https://assetstore.unity.com/packages/tools/game-toolkits/character-controller-super-135316

Aquest asset permet crear un controlador en primera i tercera persona de manera molt personalitzable. Igual que el primer asset esmenat té mecàniques bàsiques de moviment, com és caminar, córrer, saltar i caure. Hi ha una gran opció de personalització de les mecàniques integrades i es presenten de manera molt visual gràcies a l'inspector personalitzat.

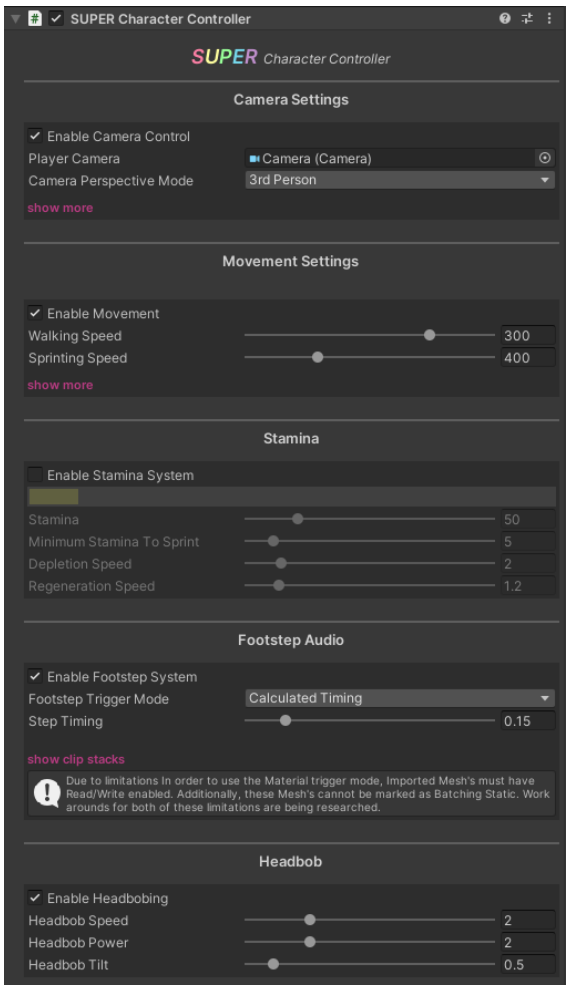

Figura 4.3. Part de l'inspector personalitzat del controlador de l'asset "Character Controller SUPER". Font: Aedan Graves.

L'asset conté:

- Documentació per facilitar l'ús de l'eina.
- Un model humanoide amb un rig complet amb múltiples animacions**.**
- Un script amb el controlador de personatge.
- Un inspector completament personalitzat.
- Una escena d'exemple.

Tot es fa en un script únic i no sembla que s'esperi iterar en el mateix script, o sigui que no s'esperen modificacions de codi. Existeixen estats i subestats pel moviment, però no fa servir una màquina d'estats.

Igual que el controlador descrit en l'apartat anterior en aquest també fa servir Rigidbody llavors de Character Controller.

4.4 Taula comparativa de referents

Característiques	Starter Assets - Third Person Character Controller	3rd Person Controller + Fly Mode	Character Controller SUPER
Documentació d'ús			
Escena demostrativa			
Scripts diferents per cada mecànica			
Ús d'estats en el controlador			
Inspector personalitzat			
Modificació d'elements de Game Feel			
Suport de mecàniques de combat			
Ús de finestra personalitzada			
Script de gestió d'inputs			

Taula 1:Taula comparativa de característiques de cada eina. Font: Elaboració pròpia.

Vermell: Característica inexistent.

Taronja: Característica parcialment implementada, però manca aspectes importants.

Verd: Característica completament implementada.

5. Disseny metodològic i cronograma

En aquest capítol s'explica la metodologia de treball i les diferents fases del desenvolupament de l'eina. S'ha creat un cronograma en el qual es mostra l'inici i quan s'ha assolit cada fase del treball. Aquest projecte s'ha realitzat seguint la metodologia Waterfall, metodologia definida per Winston W. Royce el 1970 en el seu escrit "Managing the development of large software systems".

Figura 5.1. Cicle del model en waterfall.

Font: Elaboració pròpia a partir de Winston W. Royce, 1970.

5.1 Fases de desenvolupament en waterfall

Abans dels requisits s'ha necessitat una anàlisi prèvia de diferents videojocs del gènere, tenint això en compte i la metodologia de treball en Waterfall s'han establert les següents fases:

5.1.1 Fase 1: Anàlisis de jocs de gènere Souls-like

Per poder fer una anàlisi de les diferents mecàniques i característiques dels controladors dels Souls-like s'han analitzat sis jocs Souls-like, aquests jocs són Dark Souls (FromSoftware, 2011), Dark Souls II (FromSoftware, 2014), Dark Souls III (FromSoftware, 2016), Sekiro: Shadows Die Twice (FromSoftware, 2019), nomenat com Sekiro d'ara en endevant, ELDEN RING (FromSoftware, 2022) i Star Wars Jedi: Fallen Order (Respawn Entertainment, 2019), nomenat com Star Wars d'ara en endevant.

En aquesta fase s'han analitzat diferents aspectes rellevants dels videojocs mencionats. El primer element analitzat ha sigut la càmera. Per l'anàlisi de combat s'han llistat en una taula les mecàniques rellevants i s'han ajuntat totes aquelles que són variacions d'una mateixa per aconseguir un llistat de les mecàniques bàsiques que defineixen el gènere. Després s'ha analitzat els sistemes que contenen els controladors de personatge i finalment els elements de game feel que existeixen en el combat.

5.1.2 Fase 2: Disseny del controlador

Un cap acabada la fase d'anàlisi s'han agafat les mecàniques bàsiques que defineix el combat d'un Souls-like i s'han exposat els requisits que es volen assolir amb l'eina a desenvolupar. Per això aquí s'ha definit el funcionament de la càmera, les mecàniques de combat, els sistemes necessaris per les mecàniques, els elements de Game feel i els paràmetres de disseny del controlador de personatge.

5.1.3 Fase 3: Creació del controlador

Pel desenvolupament de l'eina d'aquest treball el software de desenvolupament de videojocs escollit és el motor gràfic Unity (Unity Technologies, 2005).

En aquest apartat s'ha explicat el funcionament bàsic de Unity i les parts més rellevants del desenvolupament de l'eina. Per agilitzar el desenvolupament de l'eina s'han importat tres packages desenvolupats per Unity Technologies: ProBuilder, Cinemachine i Input System.

En aquesta explicació s'han obviat aquelles parts les quals no afecten el funcionament principal del projecte, s'han obviat explicacions com comprovar que alguns elements de codi no siguin nuls per evitar errors d'objecte no instanciat, els valors subjectius dels paràmetres dels elements de l'escena i altres.

Les variables i mètodes rellevants per una mateixa classe s'han fet privades. Aquells mètodes que requereixen altres classes s'han fet públiques i per les variables s'ha creat una propietat per poder exposar el valor de la variable sense permetre una modificació d'aquesta.

Per la nomenclatura de les variables privades s'ha fet servir Camel Case (Microsoft Documentation), per la qual cosa el nom de les variables comença amb una lletra minúscula i l'inici de cada paraula amb una lletra majúscula. Per les variables públiques, mètodes i noms de classe s'ha fet servir Pascal Case (Microsoft Documentation), és com el Camel Case exceptuant que la primera lletra també comença en majúscula.

Pels diagrames UML de les explicacions s'ha fet servir Visual Studio (Microsoft, 1997).

5.1.4 Fase 4: Disseny i implementació de la interfície gràfica

En aquesta fase s'ha explicat com és la creació de finestres i inspectors personalitzats en l'editor de *Unity*. Per començar el desenvolupament de la finestra de l'eina s'ha creat un mock-up, per aquest mock-up s'ha fet servir el software *Figma* (Figma, Inc., 2016), aquest software permet la creació d'interfícies d'usuari interactives.

Un cop fet el mock-up s'ha implementat la interfície dissenyada en una finestra personalitzada de *Unity*. Aquesta interfície és la que s'ha fet servir per assolir les fases posteriors del desenvolupament.

5.1.5 Fase 5: Demostrador

Per validar les capacitats de l'eina i el controlador s'ha dissenyat un escenari juntament amb un personatge el qual s'ha fet servir per solucionar errors i comprovar els valors òptims per les animacions del personatge. El model i les animacions del primer personatge creat s'han extret de Mixamo (Mixamo, 2009).

D'esprés d'experimentat amb la creació d'un personatge s'han creat tres personatges més de diferents característiques els quals han sigut modelats i animats per Montes (2023).

Un cop creats els personatges s'ha desenvolupat un comportament per personatges no controlats pel jugador i afegit a la interfície gràfica opcions i paràmetres per modificar el comportament d'aquest. D'aquesta manera es pot realitzar un combat i comprovar que tots els valors definits i les mecàniques funcionen de la manera esperada.

5.1.6 Fase 6: Cas d'ús

Per demostrar que l'eina pot funcionar en projectes ja desenvolupats o en desenvolupament s'ha agafat un projecte fet en *Unity*, el qual no és un Souls-like,

creat per la "Mini Jam 113: Final Boss" (ZahranW, KingW, 2022). En aquest projecte, nomenat Serpent Blade, s'ha modificat a la protagonista perquè funcioni amb el controlador de l'eina. L'objectiu final és aconseguir adaptar el videojoc perquè funcioni amb les mecàniques d'un Souls-like mantenint la idea de disseny original del projecte.

5.2 Cronograma

El cronograma en la següent taula representa de manera visual la duració de cada una de les fases del desenvolupament de l'eina.

	Gener	Febrer	Març	Abril	Maig	Juny
Marc Teòric						
Anàlisi de referents						
Disseny Metodològic						
Entrega: Avantprojecte						
Fase 1						
Fase 2						
Fase 3						
Entrega: Memòria intermèdia						
Fase 4						
Fase 5						
Fase 6						
Conclusiones						
Memoria Final						

Taula 2 Cronograma del desenvolupament del treball. Font: Elaboració pròpia.

6. Resultats

En aquest apartat s'exposa els resultats de cada fase del desenvolupament de l'eina. En aquests resultats es mencionarà el personatge controlat pel jugador com a protagonista per simplicitat de la lectura.

6.1 Anàlisis de jocs de gènere Souls-like

L'elecció dels videojocs analitzats ha sigut donat pel fet que la saga de *Dark Souls* va ser la creadora del gènere Souls-like i l'evolució de sistemes que s'han afegit, tret o modificat poden indicar les característiques més rellevants dels Souls-like. *Sekiro* i *Star Wars* han sigut escollits perquè són videojocs considerats Souls-like encara que enfocats amb un tipus de combat bastant diferent, per aquesta raó veure els elements en comú amb la saga *Dark Souls* pot ajudar a definir les mecàniques essencials del gènere. *ELDEN RING* s'ha escollit perquè els sistemes de combat d'aquest videojoc mesclen mecàniques de tots els Souls-like que havia fet FromSoftware i unifica gran part dels sistemes.

6.1.1 Anàlisis de càmera

La càmera virtual que controla el jugador en els videojocs analitzats es controla amb un input que s'interpreta com un vector de dues dimensions (*Vector2*), aquestes dimensions es componen de les coordenades X i Y. L'input pel moviment d'aquesta càmera pot ser donat pel joystick dret del comandament o en cas de no fer servir comandament en ordinador es fa servir el ratolí.

Aquesta càmera en tercera persona persegueix el moviment del protagonista amb certa velocitat i com més lluny la càmera és del protagonista més augmenta dita velocitat fins iguala la velocitat del protagonista, d'aquesta manera hi ha una major sensació de moviment i el protagonista no està fixat al mig de la pantalla constantment.

Quan el protagonista es mou, la càmera rota parcialment cap a la direcció a la qual va el personatge. Aquest fet aporta un avantatge davant d'una càmera amb rotació fixa i és que el jugador no ha de moure constantment la càmera per poder veure cap

a la direcció d'interès, que acostuma a ser cap a la direcció del moviment. En *Dark Souls* el seguiment de la càmera manté molt centrat al protagonista al centre de la pantalla i a partir de *Dark Souls II* la càmera podia separar-se molt més del protagonista.

Figura 6.1 i 6.2. Diferència de distància protagonista-centre màxima en *Dark Souls* i *Dark Souls II*. Font: FromSoftware, 2011-2014.

La distància de la càmera i el protagonista sol ser fixa menys en algunes situacions en específic. Quan el jugador es troba en un espai que és molt petit, la càmera té una distància més reduïda i, com és d'esperar en el cas contrari, en espais més grans la càmera té una distància major. Tan bon punt el jugador és en combat contra enemics de gran envergadura la càmera també té una distància major i en situacions on la velocitat del protagonista s'ha vist reduïda, la càmera té una distància i un FOV menor, per augmentar la sensació de reducció de velocitat de moviment.

Si hi ha un obstacle en mig del recorregut entre el protagonista i la càmera, s'escorça la distància de la càmera perquè d'aquesta manera la càmera no travessi cap paret o obstacle, d'aquesta manera el jugador no pot veure elements que els desenvolupadors no volen que vegin. Aquesta correcció es fa instantàniament, ja que, si es fes amb un suavitzat, es podria arribar a veure a través dels obstacles, un cop l'obstacle ja no hi és en *Star Wars* es fa un moviment suavitzat cap a la posició normal de la càmera, en els altres jocs analitzats es fa de manera instantània.

Es consideren obstacles aquells elements del joc que són estàtics, principalment geometria de l'escenari, els altres elements són ignorats. En el cas d'elements estàtics que poden generar molèsties a l'hora de col·lidir, l'element es mostra transparent llavors de col·lidir amb la càmera.

Figura 6.2. Visualització d'un element estàtic que es torna transparent al mostrar-se entre el protagonista i la càmera en Sekiro. Font: FromSoftware, 2019.

La càmera en els Souls-like també té un mode en el qual la càmera pot fixar a un objectiu prement un botó assignat, aquest objectiu acostuma a ser un enemic o monstre neutral i per poder fixar-lo ha d'estar visible en pantalla. Quan un objectiu està fixat la càmera manté el jugador i a l'enemic seleccionat a prop del centre de la pantalla, el jugador és en la part interior de la pantalla prop del centre i l'enemic en la part superior.

Figura 6.2. Mostra de posició del protagonista i enemic en el mode de fixat d'objectiu en Elden Ring. Font: FromSoftware, 2022.

6.1.2 Anàlisis de mecàniques

En aquest apartat s'han llistat les mecàniques de combat de cada joc. Una mecànica de combat és aquella que pot influenciar el resultat d'aquest. En verd s'han marcat els videojocs que tenen dita mecànica i en vermell els que no la tenen. Amb aquesta llista es busca veure les mecàniques més comunes dels videojocs analitzats per poder fer la llista de mecàniques que defineixen el combat d'un Souls-like.

Mecànica	Dark Souls 1	Dark Souls 2	Dark Souls 3	Sekiro	Star Wars	ELDEN RING
Caminar						
Correr						
Esprintar						
Rodar						
Esquivar						
Atac lleuger						
Atac pesat						
Atacar en l'aire						
Backstep						
Backstab						
Bloquejar						
Bloqueig perfecte						
Parry						
Riposte						
Deathblow						
Utilitzar un objecte						
Canviar d'arma						
Atac després de rodar						
Atac després de correr/backstep						
Fer una patada						
Agafar l'arma amb dues mans						
Saltar						
Dual wielding						
Art de l'arma						
Atacs especials						
Ajupir-se						

Taula 3: Taula comparativa de mecàniques de combat dels videojocs analitzats.

Font: Elaboració pròpia.

De la llista de la Taula 2 hi ha diversos elements que encara que en diferents jocs es tracten de manera diferent es poden agrupar una mecànica que en funcionament és igual. D'aquesta manera si s'ajunten les mecàniques que tenen un mateix funcionament d'aquesta llista es pot veure com les mecàniques dels combats resultants dels Souls-like és menor, quedant així les següents agrupacions:

- **Moviment:** S'ajunta caminar, córrer, esprintar i ajupir-se, ja que, exceptuant animacions i diferència de valors en els paràmetres, la lògica darrera d'aquestes mecàniques es manté igual entre totes elles.
- **Evadir:** Esquivar, rodar i backstep es poden considerar una mateixa mecànica d'evasió d'atacs. Igual que en el moviment, la lògica de les mecàniques es manté igual, només canvien alguns valors i animacions. L'única diferència del backstep és que sempre s'efectua cap endarrere.
- **Atacar:** Les mecàniques d'atac són l'atac lleuger, l'atac pesat, atacar en l'aire, atacar després de rodar, córrer o backstep, art de l'arma i atacs especials.
- **Bloquejar**: Bloquejar, bloqueig perfecte i parry es podrien considerar una mateixa acció la qual s'ha nomenat per simplicitat bloquejar, ja que el propòsit de les tres mecàniques és la de no evadir un atac però evitar el seu atac parcial o totalment.
- **Atac crític:** Els atacs crítics són aquells atacs els quals fan un dany major o directament eliminen l'objectiu un cop complertes certes condicions, com podria ser atacar per l'esquena o deixar un enemic a poca vida. Aquí entraren les mecàniques de backstab, deathblow i riposte. Aquestes mecàniques requereixen un enemic per efectuar-se, són atacs que bloquegen el moviment del protagonista i la de l'objectiu mentre es duen a terme.
- **Canvi d'arma:** Les mecàniques de dual wielding, canviar d'arma i agafar l'arma amb dues mans només canvien els valors dels paràmetres i les animacions dels atacs.
- **Utilitzar un objecte**: Cada objecte que es fa servir té una animació i uns efectes diferents, molts objectes restringeixen o alenteixen el moviment del personatge mentre es fa servir.
- **Saltar:** És una mecànica complementària del moviment que s'acostuma a fer servir més fora de combat. En la saga *Dark Souls* només es pot saltar si el

personatge prèviament ha estat corrents i el salt és poc alt, ja que es fa servir per saltar forats no molt grans. En *Elden Ring*, *Sekiro* i *Star Wars* el salt es pot fer des de qualsevol estat de moviment i és una mecànica que s'ha tingut molt més en compte a l'hora de dissenyar escenaris i enemics.

6.1.3 Selecció de mecàniques

De les mecàniques restants s'han eliminat aquelles les quals la manca d'elles no canvien els combats substancialment. Les que s'han mantingut són la definició de les mecàniques necessàries per poder fer un combat en un Souls-like.

La primera mecànica que s'ha tret de la llista final de mecàniques a desenvolupar ha sigut l'atac crític, ja que no és molt comú fer un atac crític en mig d'un combat en la majoria de Souls-like. En Sekiro i Star Wars l'atac crític serveix com una manera més impressionant visualment d'acabar amb un enemic. En la saga Dark Souls i en Elden Ring l'atac crític sol ser una manera de començar un combat, amb un atac per l'esquena, o en cas de fer un "parry", que és una mecànica difícil de masteritzar.

El canvi d'arma principalment afecta a les animacions i els valors d'atac d'un personatge, per la qual cosa no és una mecànica que pugui afectar un combat, ja que mentre es tingui una arma es pot combatre i normalment no s'acostuma a canviar d'arma en mig de combat. Per aquestes raons el canvi d'arma no és una mecànica que s'ha implementat en l'eina.

Poder usar o consumir un objecte és un aspecte addicional no necessari en la majoria dels combats, de fet els objectes se solen fer abans d'un combat o en cas d'haver d'usar un en mig d'un combat els jugadors acostumen a allunyar-se del seu enemic per poder fer-lo servir i després continuar el combat, per la qual cosa no s'ha tingut en compte a l'hora de fer el controlador de personatge.

Per últim, el salt en *Sekiro*, *Star Wars* i *Elden Ring* hi ha combats on el salt té certa importància, però per norma general en els combats més freqüents no és una mecànica que tingui un ús recorrent, per la qual cosa també s'ha obviat del controlador de l'eina.

Les mecàniques resultants hi són en tots els Souls-like analitzats i defineixen el controlador de combat de l'eina. Aquestes mecàniques es poden considerar mecàniques bàsiques imprescindibles en el gènere, d'aquesta manera les mecàniques restants han quedat així:

Mecànica	Dark Souls 1	Dark Souls 2	Dark Souls 3	Elden Ring	Sekiro	Star Wars
Moviment						
Evadir						
Atacar						
Bloquejar						

Taula 4: Taula comparativa de mecàniques bàsiques de combat dels videojocs analitzats. Font: Elaboració pròpia.

Un cop feta aquesta llista tenim quatre mecàniques les quals s'han definit com funcionen en els jocs analitzats per a poder extraure l'objectiu a complir de l'eina:

- **Moviment:** El protagonista es pot moure cap a la direcció designada pel jugador. Quan un personatge es mou cap a una direcció ho fa de cara, o sigui que la rotació del protagonista sempre és la mateixa que la seva direcció horitzontal del moviment. Quan el jugador fixa un objectiu amb la càmera el protagonista canvia la seva rotació a mirar sempre l'objectiu i el moviment canvia a un moviment en múltiples direccions sense canviar la rotació, ja que es vol que el personatge del jugador continuï mirant a l'objectiu constantment. La velocitat del moviment va lligada al tipus d'input que li arribi, en cas d'un teclat les tecles tenen un estat de premudes o no premudes, però en comandament caminar ve amb diferents velocitats depenent de si s'ha fet el recorregut sencer o només part d'ell.

  Mantenint premut el botó d'evadir fa que el personatge entri en una carrera més ràpida que consumirà resistència i quan s'acabi automàticament el personatge deixarà de córrer.
- **Evadir:** Les esquives depenent del joc l'animació del personatge pot ser la del salt molt curt i en altres roda, però en funcionament són exactament igual.

Mentre aquesta acció s'està efectuant no es pot canviar de direcció el moviment, el moviment és més ràpid durant l'esquiva, l'acció consumeix resistència i durant l'esquiva el jugador té un breu temps d'intangibilitat amb els atacs dels enemics. En els jocs que fan servir una animació de salt curt llavors de la de rodar acostumen a tenir un menor temps d'intangibilitat a canvi de ser una esquiva més ràpida i que acaba abans, per la qual cosa permet efectuar una altra acció abans. En *Star Wars* es fa ús del salt curt i roda, quan es prem el botó d'evadir el jugador fa un salt curt de poca durada i si mentre s'efectua aquesta esquiva s'intenta evadir de nou el protagonista roda, fent una acció que cobreix més terreny i aporta un major temps d'intangibilitat.

- **Atacar:** Els atacs acostumen a tenir un moviment cap endavant per donar més força a l'atac i també per facilitar la connexió de l'atac amb l'objectiu a atacar. Si s'està fixant a un objectiu, el personatge ataca en direcció a l'objectiu, independentment de la direcció de moviment del jugador. Un cop s'efectua un atac es pot tornar a atacar i es farà un segon atac amb animació i propietats diferents, un si es continua atacant en arribar al final de la llista de combinacions d'atacs es torna al primer, en la saga *Dark Souls* aquestes combinacions d'atacs sempre són de dos atacs, en *Elden Ring, Sekiro i Star Wars* les combinacions d'atacs poden arribar fins a 5 atacs abans de tornar al primer.
- **Bloquejar:** En *Dark Souls* i *Elden Ring* bloquejar és un estat addicional en el qual el protagonista es cobreix amb l'escut o arma disponible que pugui bloquejar i es pot mantenir mentre el protagonista és quiet, camina, esprinta i en alguns casos fins i tot evadir o atacar. En *Sekiro* i *Star Wars* el bloqueig actua com a estat complet, ja que manté al protagonista quiet durant una estona i si es manté el botó de bloqueig i el jugador intenta moure's el protagonista ho fa de manera més lenta i no pot bloquejar mentre esprinta. Mentre un personatge bloqueja, si rep un atac per la direcció en la qual està bloquejant, el personatge redueix parcialment o totalment el dany rebut.

6.1.3 Anàlisis de controladors

En aquest apartat s'han llistat els elements fonamentals i addicionals més comuns dels controladors dels jocs analitzats:

- **Sistema de vida:** Tots els sistemes de vida dels jocs analitzats funcionen de manera similar. Quan el protagonista rep mal, sigui per un enemic o per algun element de l'escenari, es perd la vida. En la interfície d'usuari aquesta vida està representada per una barra, quan la vida del protagonista disminueix en la barra es veu com desapareix de cop, però darrere de la primera barra hi ha una segona d'un altre color que disminueix gradualment, d'aquesta manera el jugador pot veure fàcilment quanta vida ha perdut. Quan la vida d'un personatge arriba a zero entra en estat de mort.
- **Sistema de resistència:** La trilogia *Dark Souls* i *Elden Ring* tenen un sistema de resistència el qual és molt rellevant en els combats, a diferència de *Sekiro* i *Star Wars* que manquen d'aquest. Aquest sistema consisteix d'un valor de resistència la qual es regenera gradualment a través del temps i que hi ha accions que poden accelerar o alentir la generació de resistència i altres accions que directament drenen la resistència i que requereixen un mínim d'ella per poder executar-se. En la interfície d'usuari també es mostra amb una barra similar a la de vida i per sota de la mateixa.
- **Sistema d'estabilitat:** L'estabilitat és un sistema que depèn de rebre un atac d'un enemic o obstacle. En rebre mal un personatge també rep dany d'estabilitat, quan arriba al màxim el personatge queda atordit durant un breu període de temps. Quan un personatge bloqueja amb un bloqueig perfecte en *Sekiro* o *Star Wars* el dany d'estabilitat augmenta menys i encara que arribi al màxim no es trenca, en els altres jocs analitzats mentre el protagonista bloqueja cada atac rebut drena resistència depenent de la força de l'atac i de l'estabilitat total del personatge, si després d'un atac té zero de resistència, llavors queda atordit.
- **Sistema de gestió d'input:** Per evitar haver de ser molt precís amb els inputs aquests jocs tenen un sistema que ajuda al jugador al fet que els seus inputs tinguin resposta encara que es facin abans de temps, fent així els controls

més responsius. Mentre es du a terme una acció si es prem el botó per fer una altra acció, com podria ser un atac després de rodar, el joc guarda el botó premut per fer l'acció immediatament quan sigui possible. Les accions emmagatzemades tenen diferents tipus de prioritat, si una acció amb major prioritat és premuda mentre s'està efectuant una acció aquesta acció guanyarà prioritat davant d'altres amb menor prioritat. Per exemple si mentre el jugador roda, intenta efectuar un altre roda i abans d'acabar la primera roda intenta atacar, si l'atac té major prioritat que la roda, en acabar la primera roda el jugador atacarà, ignorant les accions amb menor prioritat.

6.1.4 Anàlisis de game feel

Per poder considerar que un videojoc té game feel total, ha de complir les característiques dels pilars estipulats per Swink (2008). En tots els jocs analitzats els elements de game feel són el mateix, per la qual cosa es pot considerar que el gènere Souls-like compleix els pilars del game feel d'aquesta manera.

6.1.4.1 Control a temps real

Els Souls-like tenen un moviment major als 10 fotogrames per segon, per la qual cosa generen impressió de moviment. Els inputs es processen i mostren una resposta en menys de 240 ms i en el cas d'algunes de les accions, com pot ser el moviment del protagonista, es pot rectificar la direcció del moviment mentre s'estigui caminant amb menys de 100 ms entre cada actualització de l'input.

Algunes accions que restringeixen el moviment del personatge també permeten modificar la direcció del personatge durant uns quants mil·lisegons a l'inici de l'acció, permetent així una correcció de l'input. Per aquestes raons els jocs analitzats tenen control a temps real.

6.1.4.2 Espai simulat

En els Souls-like, exceptuant quan s'hi és al menú principal, el jugador pot veure el personatge que controla i un espai en el qual el pot moure i interactuar amb l'entorn que se li presenta, per aquesta raó els Souls-like tenen espai simulat.

6.1.4.3 Polit

Els Souls-like tenen molts elements de polit, s'han analitzat els elements de polit més remarcables que es poden veure durant un combat.

Totes les animacions i tot el moviment és progressiu en aquests tipus de jocs, no hi ha salts entre animacions.

En impactar atacs amb superfícies o contrincants apareixen partícules indicant un impacte i el resultat d'aquest, un impacte on surten espurnes vol dir que l'atac ha sigut bloquejat o ha rebotat, un impacte on surt sang vol indicar que ha fet dany al contrincant. També en cas de jugar amb comandament els atacs fets i rebuts també venen acompanyats una vibració al comandament.

Quan hi ha un impacte gran, sigui ocasionat per un atac o per altres causes, depenent de la distància entre l'impacte i el jugador la càmera tremola per augmentar la impressió d'impacte.

6.2 Disseny del controlador

En aquest apartat s'han definit com es vol que funcioni la càmera, els sistemes del controlador del personatge, les mecàniques i el Game feel.

6.2.1 Càmera

Per la càmera s'han fet dos estats possibles, la de moviment lliure on el jugador pot modificar la rotació de la càmera amb l'input designat, mentre la càmera orbita al personatge, i l'altre estat és l'estat de fixar objectiu, on la rotació de la càmera manté centrat al protagonista i l'enemic sense deixar de seguir al protagonista.

6.2.2 Sistemes del personatge

El controlador del personatge requereix múltiples sistemes per funcionar:

- **Gestió d'inputs:** Efectua una acció depenent del botó premut i en cas de no poder efectuar l'acció assignada s'aguanta l'input durant un temps determinat.
- **Gestió de vida:** Defineix la vida màxima, actual i la regeneració d'aquesta.

- **Gestió de resistència:** Defineix la resistència màxima, actual i la regeneració d'aquesta. En cas de gastar-ne més de la que es té el valor actual pot ser menor a zero i que no es comenci a veure resistència fins que el valor torni a nombres positius. Quan la resistència ha arribat a zero o menys el personatge estarà cansat i les accions que requereixin que el personatge no estigui cansat no es podran fer servir, per deixar d'estar cansat s'ha de regenerar resistència fins a un percentatge designat.
- **Gestió d'estabilitat:** Defineix l'estabilitat màxima, actual, la regeneració de la mateixa i el temps que es tarda a començar a regenerar l'estabilitat. En cas d'arribar a 0 el personatge serà atordit i l'estabilitat es regenerarà al màxim de cop.
- **Gestió de físiques:** Comprova si el personatge està tocant el terra i aplica al personatge elements de físiques, com és la gravetat.
- **Gestió d'estats del personatge:** Aquest gestor efectua les accions d'un estat definit. Es permeten canvis d'estat, però només pot tenir un estat alhora i els canvis d'estat necessiten complir uns requisits per poder efectuar-se.
- **Gestió de subestats del personatge:** Depenent de l'estat del personatge es poden permetre tenir subestats, en el cas de les mecàniques analitzades anteriorment només bloquejar entraria dintre d'aquest subestat, ja que es poden fer altres accions, com caminar o córrer, mentre es bloqueja.
- **Gestió d'equipament:** Els personatges poden tenir una arma o escut a cada mà o una arma a dues mans, depenent de l'arma els atacs efectuats tindran uns valors, unes col·lisions i unes animacions diferents.
- **Gestió de combinacions d'atacs:** Cada equipament pot tenir una cadena d'atacs assignada. Cada cop que s'intenti fer un atac d'aquesta cadena s'efectuarà el següent, en cas que passi un temps designat, que s'arribi a l'últim atac o que es comenci una altra combinació d'atacs el pròxim atac serà el primer de la cadena.
- **Gestió d'estadístiques:** Cada personatge té estadístiques rellevant per un o més estats, com podria ser la velocitat, vida, resistència o la força.

6.2.3 Estats del personatge

Cada acció del personatge que restringeix altres accions es considera un estat dintre del sistema de gestió d'estats del personatge. Aquests estats comparteixen certs paràmetres:

- El temps abans de poder canviar d'estat.
- Els estats als quals pot efectuar una transició.
- Els estats als quals es pot efectuar una transició sense comprovar cap requisit, aquests serveixen com a interruptors d'estats, com podria ser morir mentre es fa un atac.
- Els subestats que es poden efectuar durant aquest estat.
- La duració de l'estat, un cop s'acaba es torna a l'estat principal, per defecte el de moviment.
- La resistència que es drenarà en començar l'estat.
- El multiplicador de regeneració de resistència mentre s'està en l'estat.
- La resistència mínima necessària per poder canviar d'estat.
- L'estat pot ignorar o no que el personatge estigui cansat per canviar poder canviar a aquest estat.
- El nom de l'animació i de la transició perquè l'animator pugui efectuar l'animació corresponent a l'estat.

Per cada mecànica definida en l'anàlisi s'han creat un estat, dels quals hi pot haver múltiples instàncies, i també s'ha creat un estat d'atordiment, ja que el sistema d'estabilitat el requereix.

6.2.3.1 Caminar, córrer, esprintar i moviment en l’aire:

- El moviment mínim necessari per passar d'estar quiet a moure's, sempre entre 0 i 1.
- El multiplicador de la velocitat del personatge, per exemple en esprintar el personatge es podria moure dues vegades més ràpid que corrents.
- El temps en el qual es tarda a accelerar a màxima velocitat i també per desaccelerar.

- En cas que la càmera estigui fixant un objectiu es canviarà les variables de les animacions i la rotació del personatge per sempre està mirant en direcció a l'objectiu. També una opció per mirar a l'objectiu només quan el personatge controlat està quiet.
- El multiplicador de la velocitat del personatge mentre es fixa un objectiu.

6.2.5.2 Evadir:

- El temps abans que el personatge ja no pugui rotar i la direcció del moviment es quedi bloquejada.
- La velocitat a la qual es mou el protagonista durant l'acció.
- Es pot activar que el personatge miri a l'objectiu mentre s'efectua l'esquiva.
- Es pot activar que, en cas d'estar fixant objectiu, la direcció de la roda intenti fer una circumferència on l'objectiu és el centre i la distància entre el jugador i l'objectiu és el radi.
- Es pot activar que es mantingui la velocitat després d'evadir, en cas contrari a l'acabar l'estat d'esquiva la velocitat del personatge és zero.
- El temps abans que el personatge sigui intangible amb els atacs dels enemics.
- La duració de la intangibilitat.

El temps a l'inici d'una esquiva en el qual es pot fer un canvi de direcció és per no treure el control al jugador al mateix moment en el qual inicia una esquiva, d'aquesta manera es manté el control a temps real important pel game feel del joc. No s'acostuma a mantenir aquesta finestra de correcció d'input durant tot el moviment.

6.2.5.3 Atacs:

- El temps abans que el personatge ja no pugui rotar i la direcció del moviment es quedi bloquejada.
- La velocitat màxima a la qual el personatge es pot arribar a moure durant l'estat.
- La velocitat de rotació durant el temps en el qual es pot rotar.
- Es pot activar que quan s'ataqui fixant un objectiu l'atac sempre sigui en direcció a l'enemic o en la direcció de l'input del jugador.

- El temps abans que la combinació d'atacs es reinici.
- El temps per activar que l'arma faci dany en col·lisió i el temps que dura actiu.
- El multiplicador de dany de l'arma i també un multiplicador de dany d'estabilitat.
- El valor d'estabilitat addicional del personatge en atacar.
- Els multiplicadors de velocitat del moviment durant el transcurs de l'atac.

6.2.5.4 Atordiment:

L'estat d'atordiment fa servir unes quantes variables d'estat, però no requereix totes, ja que al ser un estat el qual no s'arriba a demanda, sinó que t'han d'atacar i sempre s'ha de poder entrar a aquest estat les variables que tenen a veure amb la resistència i els subestats permesos durant aquest estat no es fan servir. Els altres paràmetres són:

- La velocitat màxima de retrocés a la qual es pot arribar.
- Els multiplicadors de velocitat del moviment durant el transcurs de l'atac.

6.2.5.5 Bloquejar:

Bloquejar subestat, per la qual cosa no fa servir els paràmetres comuns dels estats, l'únic paràmetre que fa servir és el nom de l'animació a realitzar. Els altres paràmetres del bloqueig són:

- El multiplicador de velocitat, el qual s'espera que sigui menor o igual a 1.
- El temps necessari d'estar bloquejant per parar atacs.
- El temps que dura, des de l'inici de l'estat, el bloqueig perfecte. Aquest bloqueig denega tot el dany i també el dany a l'estabilitat.

6.3 Creació del controlador

En aquest apartat s'explica el software de desenvolupament escollit, la preparació del projecte i la implementació del disseny dels sistemes i les mecàniques definides en l'apartat anterior.

6.3.1 Implementació en Unity

Per implementar el controlador i l'eina dissenyada s'ha escollit fer servir el motor gràfic *Unity* (*Unity Technologies*, 2022a) per la flexibilitat que té a l'hora de crear i modificar elements del mateix editor. També *Unity* (*Unity Technologies*, 2022a) aporta una manera fàcil d'exportació i importació d'assets amb l'ús dels Packages.

6.3.1.1 Unity

Unity és un motor gràfic de videojocs multiplataforma crear per *Unity Technologies* el 2005. És un motor que funciona amb C++ (Bjarne Stroustrup, 1985) encara que l'API de scripting del mateix funciona amb C# (Microsoft Corporation, 2000). *Unity* s'utilitza principalment pel desenvolupament de videojocs, ja sigui en 3D com en 2D, encara que també s'ha fet servir per desenvolupar apliacions i animacions gràcies al seu suport a animacions esqueletals. Des de *Unity* 5 (2015), el sistema de físiques simulades que es fa servir és Nvidia PhysX 3.3 (NVIDIA Corporation, 2015) i pel scripting fa servir les llibreries de codi obert .NET (Microsoft, 2002).

Unity és un motor amb una documentació extensa publicada a internet i de codi obert, això vol dir que és un motor altament modificable pels mateixos usuaris del software.

6.3.1.2 Scripting d'editor

Per poder afegir funcionalitats personalitzades als elements de l'editor de *Unity* es fa servir una API creada per *Unity Technologies*. Per a les funcionalitats bàsiques de l'editor del motor hi ha múltiples scripts amb el namespace *UnityEditor* i per un usuari de *Unity* poder fer servir o modificar dites funcionalitats del motor ha d'escriure "using UnityEditor", això farà que cada script pugui llegir el contingut dels scripts que tenen el namespace amb el mateix nom.

Els scripts d'editor s'acostumen a posar dintre de carpetes amb el nom "Editor", no només perquè ajuda a mantenir un ordre dins d'un projecte, sinó també perquè *Unity* pot llençar un error en cas que un script d'editor estigui fora de la carpeta esmentada, ja que *Unity* ha de poder diferenciar que aquells scripts no formen part del codi necessari per a la build del joc final dels que sí.

En els scripts també es pot escriure codi de preprocessador, aquest codi farà que es llegeixi o no un codi encapsulat dintre d'unes condicions, per exemple es pot fer que un codi s'efectuï depenent de si estem en editor o no, o també una altra manera molt comuna de fer servir codi de preprocessador és per evitar elements que mal funcionen en diferents plataformes.

6.3.1.3 Asset Packages

Els Packages de *Unity* (*Unity Technologies*, 2022a) serveixen per exportar arxius de *Unity* en format .unitypackage per poder importar dits arxius en un altre projecte, mantenint l'estructuració de carpetes, sense perdre meta-dades (Unity Documentation).

Per poder exportar un Package s'ha de prémer botó dret en un o múltiples arxius i seleccionar "Export Package". Un cop fet s'obre una finestra en la qual es mostren tots els arxius seleccionats i en el cas de tenir l'opció d'inclusió de dependències també es mostren totes les dependències necessàries perquè els arxius seleccionats funcionin correctament.

Figura 6.3. Finestra d'exportació d'un Package amb dependències.
Font: *Unity Technologies*.

Per la importació d'un Package només s'ha de seleccionar l'arxiu des de les opcions d'importació de Packages o també es pot obrir l'arxiu amb un projecte de *Unity* obert.

Un cop ha començat la importació del Package en *Unity* s'obre una finestra d'importació, aquesta finestra té el mateix funcionament que la d'exportar exceptuant l'opció d'inclusió de dependències i el fet que el menú d'importació té icones per mostrar arxius nous i d'altres que es poden substituir amb la importació.

6.3.2 Preparació de projecte

El projecte creat pel desenvolupament de l'eina s'ha creat una plantilla buida 3D en la versió de *Unity* 2021.3.23f1 (*Unity Technologies*, 2023a). Per agilitzar el desenvolupament de l'eina s'han inclòs tres Packages desenvolupats per *Unity Technologies*, que són Input System (*Unity Technologies*, 2023b), ProBuilder (*Unity Technologies*, 2022a) i Cinemachine (*Unity Technologies*, 2022b).

6.3.2.1 Input System

Serveix com a substitució al sistema de gestió d'inputs conegut com a *Input Manager* (Unity Manual - Input Manager) que ve de base quan es crea un projecte des de zero. Aquest nou sistema de gestió d'inputs permet major flexibilitat, ja que permet registrar qualsevol input de qualsevol dispositiu per executar accions.

Input System ve amb la possibilitat de crear un arxiu amb informació dels diferents mapes d'acció, un mapa d'acció és aquell en el qual es designen accions i els inputs que activen aquelles accions. Cada acció es pot fer servir per cridar un mètode, també hi ha possibilitat de rebre informació de l'input rebut, com podria ser la X i la Y d'un *Vector2* (Unity Manual - Input System).

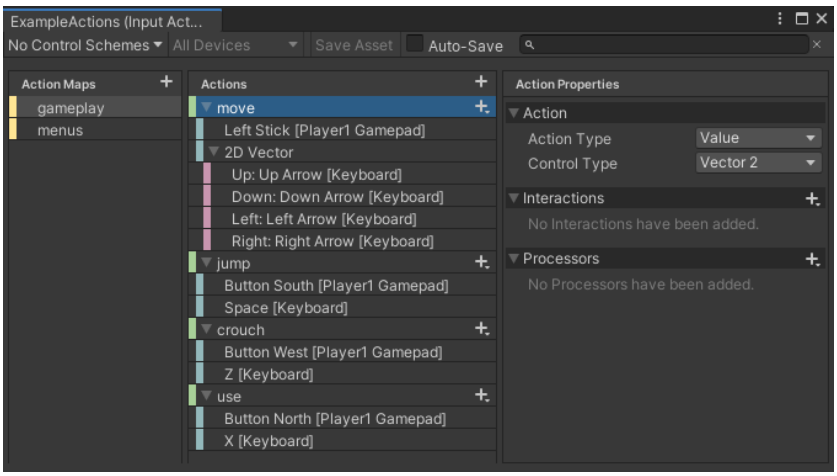

Figura 6.4. Finestra de configuració de mapa de controls. Font: *Unity Technologies*.

6.3.2.2 ProBuilder

És una eina per, des de zero, crear, editar i texturitzar geometria. *ProBuilder* facilita la creació de disseny de nivells amb les funcionalitats proporcionades.

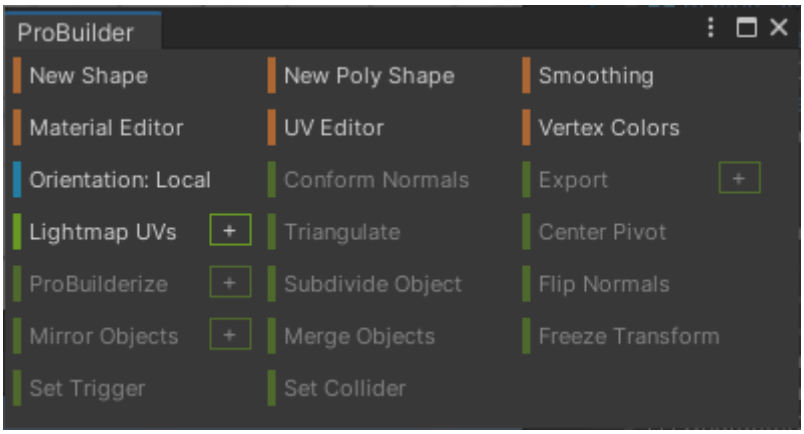

Figura 6.5. Finestra d'opcions de la geometria de *ProBuilder*.
Font: *Unity Technologies*.

6.3.2.3 Cinemachine

És una eina que ve amb mòduls dissenyats per facilitat el funcionament de la càmera virtual de *Unity* i reduir els possibles problemes que pot aportar crear el funcionament d'una càmera a mà.

Aquest Package ve amb multitud de càmeres amb diferents funcionaments preparades per usar en un projecte només requerint una referència de la posició d'un objecte al qual seguir i la posició de l'objecte al qual mirar. Aquestes càmeres venen amb una gran possibilitat de personalització de paràmetres i tenen mòduls per funcionar amb altres assets, com per exemple l'*Input System* (*Unity Technologies*, 2023b).

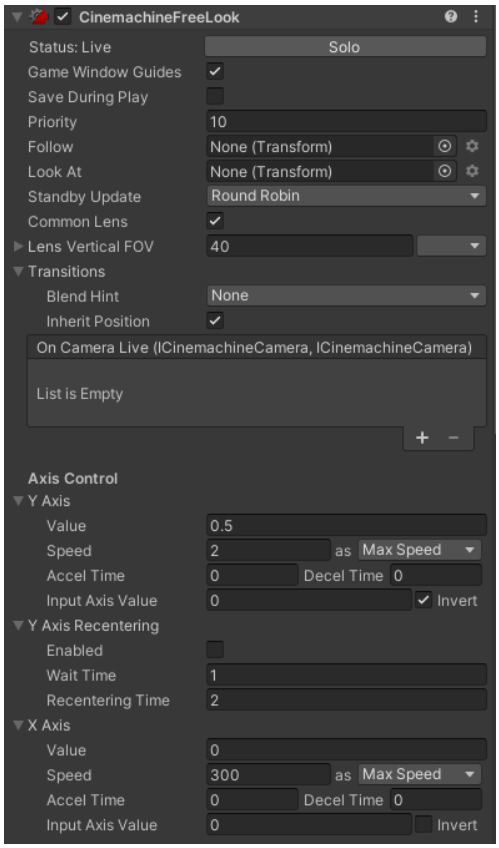

Figura 6.6. Inspector d'una càmera "Free Look" de Cinemachine.

Font: *Unity Technologies.*

6.3.3 Desenvolupament de sistemes

Pel correcte desenvolupament de les mecàniques s'han creat diversos sistemes de gestió els quals són necessaris per a les mecàniques.

6.3.3.1 Dades

Totes les dades que fan servir els sistemes i que un usuari pot modificar s'han posat en *ScriptableObjects*. Les dades del mateix personatge, com estadístiques o les dades dels estats, resideixen en un *ScriptableObject* amb el nom de *CharacterData*.

Els estats requereixen variables comunes per poder efectuar canvis d'estat, per això els *ScriptableObjects* de dades d'estats hereten d'una classe *ScriptableObject* nomenada *StateData*. Tots els estats i subestats depenen d'una variable *AnimationID(string)* per, en l'*OnStart* de cada estat i subestat, poder efectuar les transicions de l'*AnimatorController*.

En el *StateData* s'han posat les variables necessàries per fer les comprovacions pels canvis d'estat, aquestes variables són:

- *CanTransitionStates*(*ECharacterState*): Els estats als quals es pot efectuar una transició.
- *CanAlwaysTransitionStates*(*ECharacterState*): Els estats als quals es pot efectuar una transició sense necessitat de fer cap comprovació.
- *AllowedSubstates*(*ECharacterSubstate*): Els subestats que es permeten mentre s'està en aquest estat.
- *TimeBeforeCanTransition*(*float*): El temps que ha d'haver passat per poder canviar a un estat de *CanTransitionStates.*
- *Duration*(*float*): Duració de l'estat, quan l'estat porta més temps que la duració es retorna a l'estat *WALKING*.
- *StaminaCost*(*float*): Valor de quanta resistència es consumeix en fer un canvi d'estat.
- *StaminaRecoveryMultiplier*(*float*): Multiplicador de la recuperació de resistència durant l'estat.
- *StaminaNeeded*(*float*): Resistència necessària per fer un canvi d'estat.
- *IgnoreBeingTired*(*bool*): Ignorar l'estat de cansat quan s'intenta fer un canvi d'estat.

Les variables no constants de moviment a mantenir entre diferents estats es guarden en un script nomenat *MovementRecord*. Aquestes variables són:

- *InputDirection*(*Vector2*): Valor el modificat per *PlayerInputsManager* cada cop que es fa un input de moviment, serveix per als estats que requereixen l'input del jugador per influenciar el moviment.
- *HorizontalSpeedMultiplier*(*float*): Valor que multiplica la velocitat d'un estat, d'aquesta manera un personatge pot ser més o menys ràpid sense modificar el seu valor base de velocitat de moviment.
- *HorizontalSpeed*(*float*): Velocitat actual del personatge en el pla horitzontal XZ.
- *MovementDirection*(*Vector2*): Direcció del moviment en el pla XZ.

- *Movement*(*Vector3*): Moviment que s'efectua cada cop que es fa el *Move* del *CharacterController.*
- *DodgeDirection*(*Vector3*): Direcció del moviment en el moment de començar una esquiva.
- *VerticalSpeed*(float): Velocitat en l'eix Y del personatge.
- *ImpulseVelocity*(*Vector3*): Impuls que s'aplica al *Movement* en cas que algun element empenyi al personatge.
- *IsBeingPushed*(*bool*): És cert en el cas que el personatge estigui rebent un impuls en el pla XZ major a 0.01f.

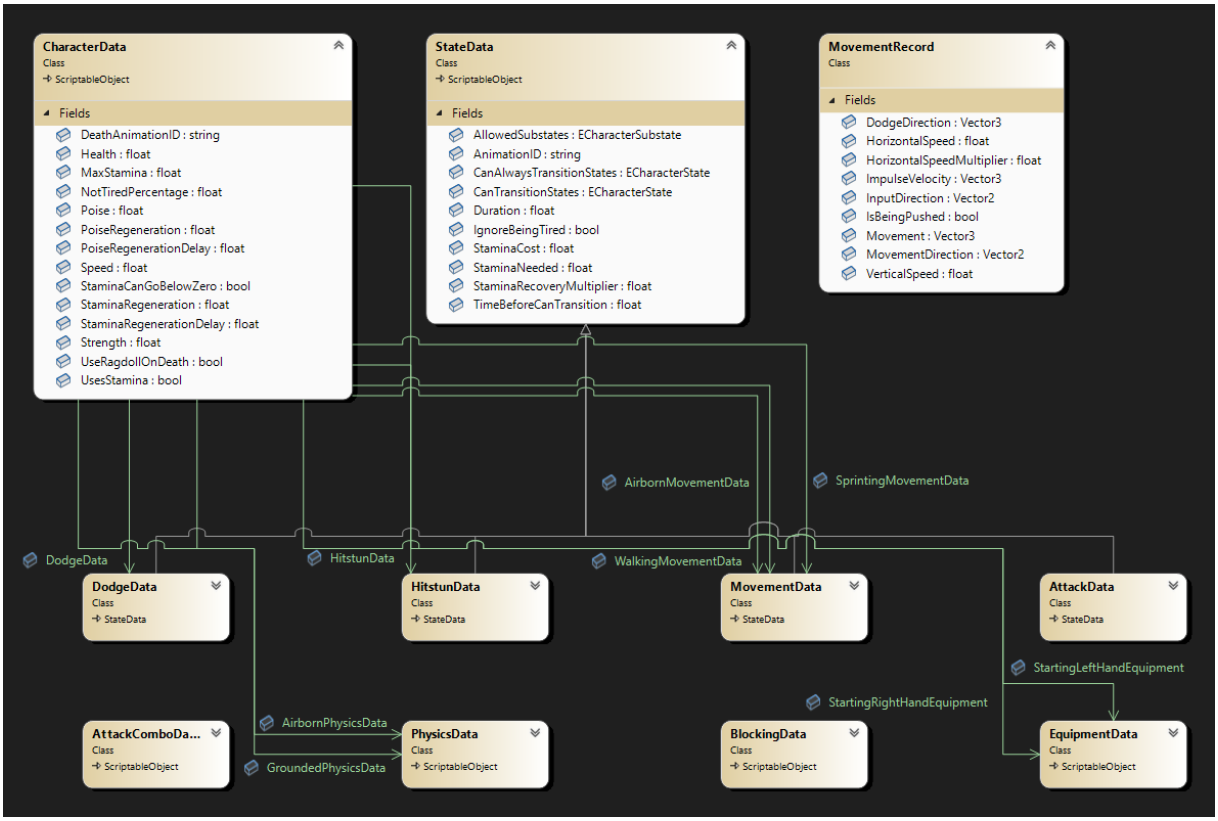

Figura 6.7. Diagrama UML simplificat de les dades.

Font: Elaboració pròpia.

6.3.3.2 Gestió d'estats

Els comportaments d'un personatge es componen d'un estat, un subestat i un estat de físiques en cada moment. Per gestionar aquests estats s'ha creat una classe *MonoBehaviour* amb el nom de *CharacterStatesController* el qual s'ha posat com a component al personatge de l'escena.

Primer s'han definit els possibles estats i subestats en *enums*(C# reference) amb l'atribut *Flags,* el dels estats s'ha nomenat *ECharacterState* i l'altre *ECharacterSubstates*. *ECharacterState* conté els membres *WALKING, SPRINTING, AIRBORN, DODGING, ATTACKING, DEAD i HITSTUN* i *ECharacterSubstates* conté els membres NOTHING i BLOCKING.

El gestor d'estats també controla les físiques del personatge, per la qual cosa s'ha afegit un *enum* del tipus *EPhysicsState* amb els membres *GROUNDED* i *AIRBORN.*

Pels estats s'ha creat una *interface* (C# Reference) *IStateBehaviour* la qual conté les funcions *OnEnter*(), *OnUpdate*(*float deltaTime*), *OnExit*() i *GetStateData*(). *OnEnter* es crida cada cop que s'entra a l'estat i *OnExit* el mateix quan se surt de l'estat. En cada *Update* del *CharacterStatesController* es crida a l'*OnUpdate* de l'estat actual i se li passa per paràmetre el *Time.deltaTime* per no haver de repetir la crida cada cop que es necessita el temps entre *Updates*. *GetStateData* retorna el *StateData.* Pels subestats s'ha fet una *interface* igual, però sense el *GetStateData* que s'ha nomenat *ISubstateBehaviour.*

Per cada estat s'ha creat una classe que hereta de *IStateBehaviour*, per cada subestat s'ha creat una classe *ISubstateBehaviour* i per les físiques s'ha creat una classe *PhysicsState.* En el *CharacterStatesController* s'ha creat una variable de cada tipus dels mencionats anteriorment per inicialitzar-los en el *Start*. En cada *Update* es criden els *OnUpdate* de cada estat, subestat i estat de físiques actiu.

Per poder fer un canvi d'estat s'han fet tres mètodes *ChangeState, ChangeSubstate* i *ChangePhysicsState.* Cada mètode canvia l'estat actiu del seu tipus i efectua l'*OnExit* de l'estat anterior i l'*OnEnter* del nou estat.

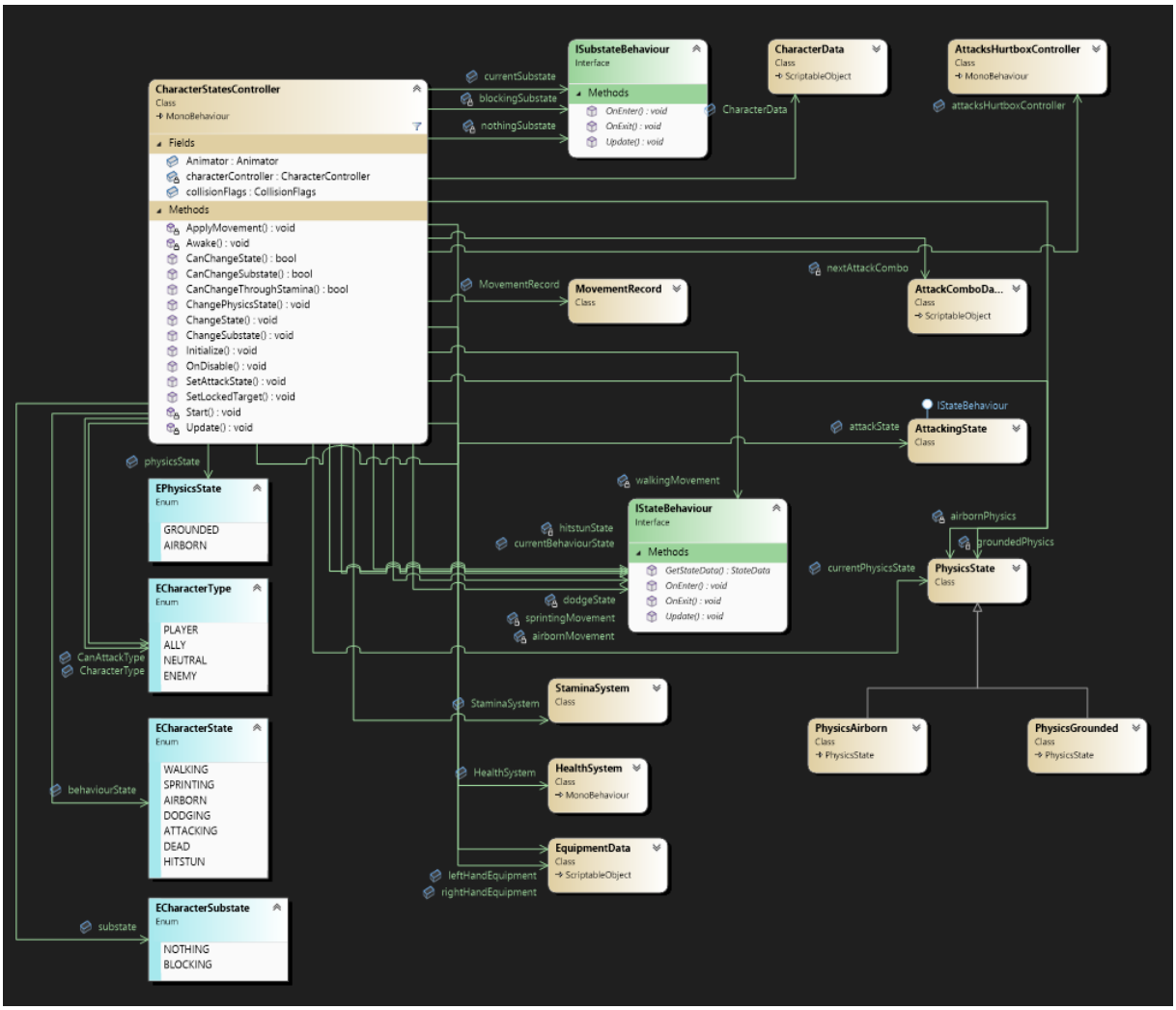

Figura 6.8. Diagrama UML simplificat de la gestió d'estats.

Font: Elaboració pròpia.

6.3.3.3 Gestió d'inputs

Per poder gestionar els inputs del jugador s'ha creat un mapa d'accions amb l'*Input System* en el qual s'han assignat totes les accions que es poden fer mitjançant inputs i els inputs que aquestes accions requereixen per poder fer la crida al mètode de l'acció.

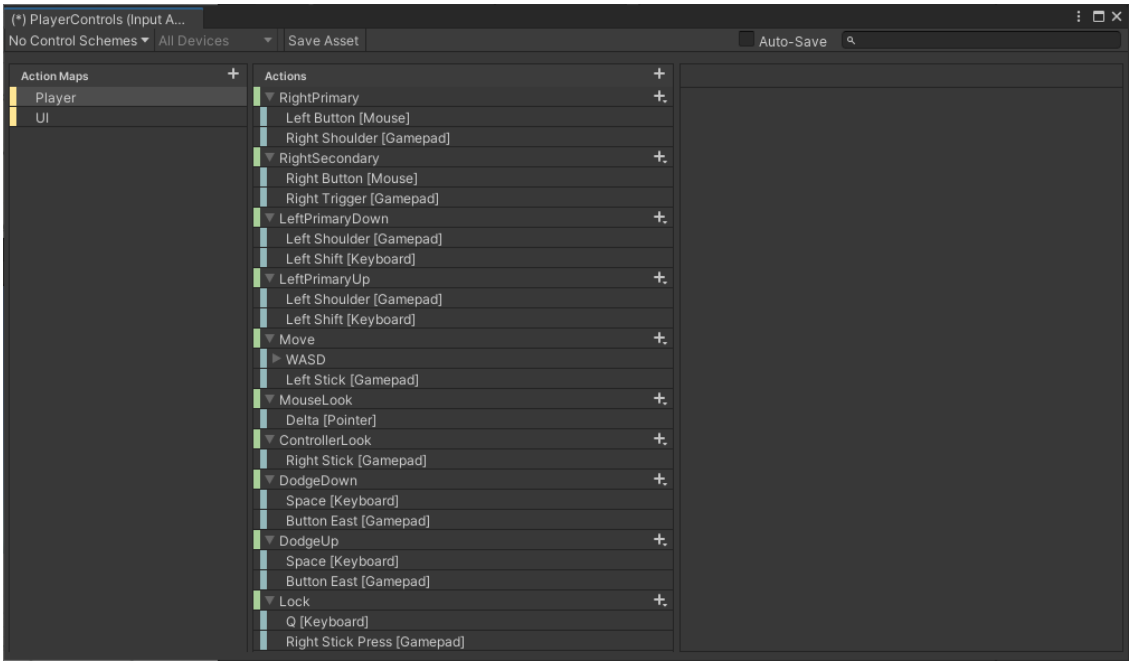

Figura 6.9. Captura del mapa d'accions del controlador del personatge.

Font: Elaboració pròpia.

Per poder llegir aquests inputs s'ha afegit un component *PlayerInput.* Aquest component és del mateix *Input System* i serveix per suplir a altres components de l'objecte que tingui *PlayerInput* de les crides als mètodes de les accions del mapa d'accions.

Pels mètodes dels inputs s'ha creat un script *MonoBehaviour* nomenat *PlayerInputsManager* el qual està com a component dintre del personatge creat. Per poder fer el buffer d'inputs, perquè s'aguantin fins que es pugui efectuar l'acció desitjada, s'ha creat una classe de tipus *Input.* En aquesta classe hi ha dos constructors, els quals defineixen la prioritat de l'input en *int* i una *Action*, un dels constructors rep un paràmetre *ECharacterState* i l'altre un *ECharacterSubstate*. Aquesta diferenciació de constructors és per poder fer la comprovació de canvi d'estat o subestat depenent de la necessitat.

Quan es crida un mètode a través d'un input aquell input es guarda en una variable *inputToExecute(Input),* en cas que la prioritat d'un nou *Input* sigui igual o major a la de l'input *ToExecute* llavors el nou *Input* el substitueix.

En el *Awake* es fa un *GetComponent* del component *CharacterStatesController* per tenir una referència directa als seus mètodes. En el *Start* s'han definit els inputs i l'*Action* a efectuar un cop es permet el canvi d'estat al de l'input, l'*Action* en cada input fa un *ChangeState* o *ChangeSubstate* del *CharacterStatesController*. En el *Update* si *inputToExecute* no és nul, es comprova si es pot fer el canvi d'estat necessari per poder cridar l'*Action* de l'input *ToExecute*, en cas de no poder si tarda més de *timeToResetInput(float)* en poder per la crida de l'*Action* llavors *inputToExecute* es torna nul.

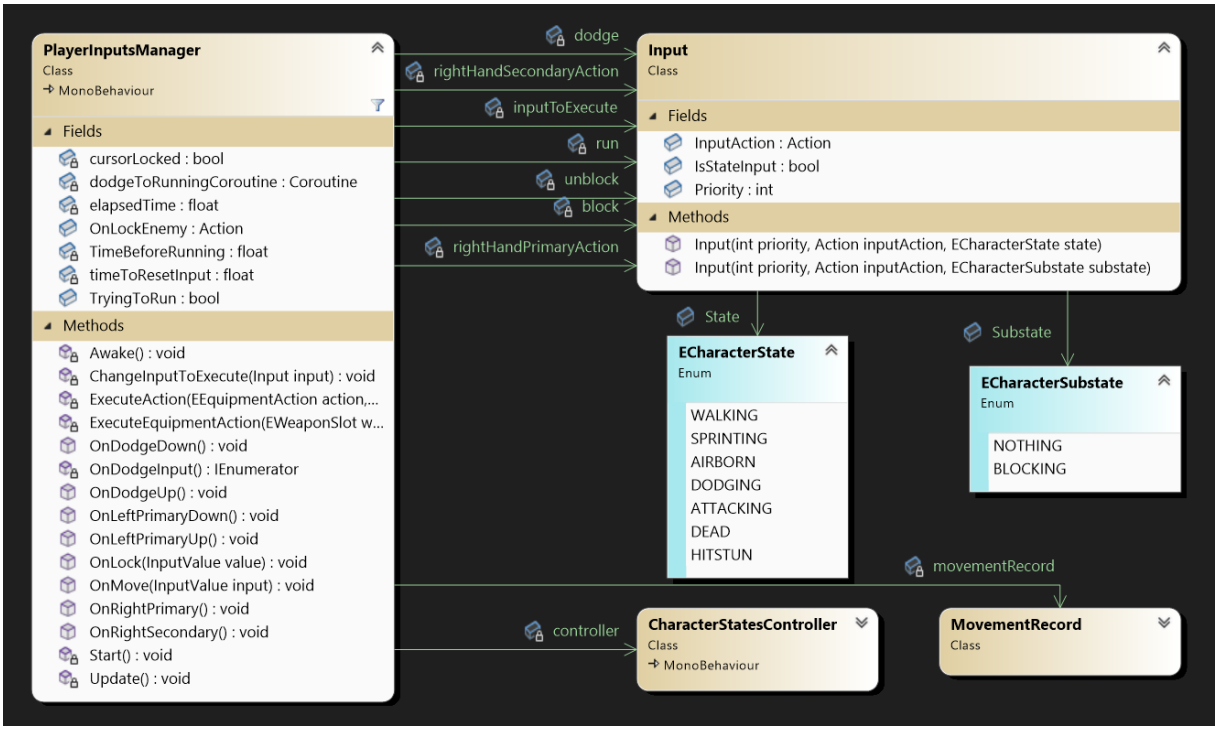

Figura 6.10. Diagrama UML simplificat de la gestió d'inputs.

Font: Elaboració pròpia.

6.3.3.4 Gestió de resistència

Pel sistema de gestió de resistència s'ha creat una classe *StaminaSystem,* la qual s'instància i s'efectua el constructor en el *Awake* del *CharacterStatesController.*

Per la creació d'aquest sistema s'han afegit variables en el *CharacterData,* aquestes variables són:

- *UsesStamina*(*bool*): En cas de ser fals no es fa servir el sistema de resistència.

- *MaxStamina*(*float*): La resistència màxima del personatge.
- *StaminaRegeneration*(*float*): La resistència per segon del personatge.
- *StaminaRegenerationDelay*(*float*): El temps que tarda el personatge a regenerar resistència després d'haver-ne gastat.
- *NotTiredPercentage*(*float*): Percentatge de resistència respecte *MaxStamina* en el qual el personatge deixa d'estar cansat
- *StaminaCanGoBelowZero*(*bool*): Permet o prohibeix que la resistència pugui disminuir més que zero.

StaminaSystem consta de les següents variables:

- *characterData*(*CharacterData*): Aconseguida a través del constructor.
- *maxStamina*(*float*): Aconseguida a través del C*haracterData*.
- *currentStamina*(*float*): És el valor de resistència actual, comença amb el valor de *maxStamina,* però no el pot superar.
- *currentRecovery*(*float*): Aconseguida a través del C*haracterData*.
- *timeToRegenerateAgain*(*float*): En aquesta variable s'assigna el valor *Time.time* més el valor *StaminaRegenerationDelay* del *CharacterData.*
- *tired*(*bool*): Quant *currentStamina* és igual o menor a zero aquesta variable és certa.
- *noLongerTired*(*float*): Percentatge de *currentStamina* respecte *maxStamina* en el qual *tired* torna a ser false*.*

En la funció *Update* es compta el temps que falta per poder regenerar resistència (*timeToRegenerateAgain*) en cas d'haver gastat recentment i un cop s'ha superat aquest temps la *currentStamina* augmenta *currentRecovery* per segon. En cas d'estar cansat si *currentStamina* >= *maxStamina* * *noLongerTired* / 100 llavors l'estat de *tired* passa a ser false.

El sistema de resistència també consta d'una funció *SetCurrentRecovery()* i *SetMaxStamina()* que reben un *float* per paràmetres i serveixen per modificar la *currentRecovery* i la *maxStamina* respectivament*.* Per fer el drenatge de resistència per cada canvi d'estat s'ha fet una funció *DrainStamina* que rep un *float*.

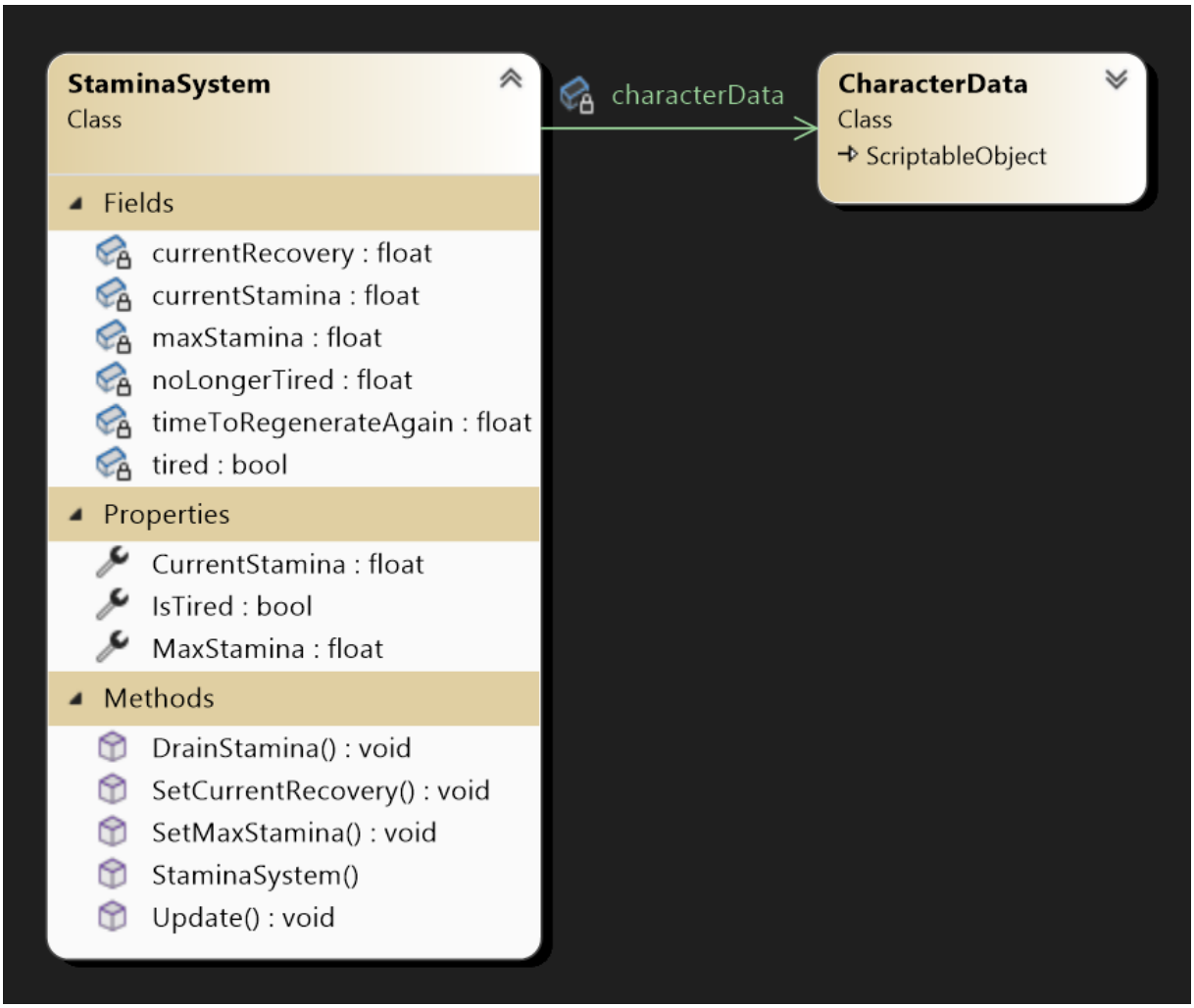

Figura 6.11. Diagrama UML simplificat del sistema de gestió de resistència.

Font: Elaboració pròpia.

6.3.3.5 Gestió de vida i estabilitat

Com l'estabilitat i la vida del personatge són dos elements que es modifiquen quan es rep un cop s'han ajuntat ambdós sistemes en un mateix script de classe *MonoBehaviour* nomenat *HealthSystem.*

Per la creació d'aquest sistema s'han afegit més variables en el *CharacterData,* aquestes variables són:

- *Health*(*float*): Vida màxima del personatge.
- *Poise*(*float*): Estabilitat màxima base del personatge.
- *PoiseRegeneration*(*float*): Regeneració de *Poise* per segon.

- *PoiseRegenerationDelay*(*float*): Temps que tarda la *PoiseRegeneration* activar-se després de rebre un impacte.

El *HealthSystem* consta de les variables:

- *controller*(*CharacterStatesController*): Referència al controlador d'estats del personatge per fer canvis d'estat.
- *characterData*(*CharacterData*): Referència al *ScriptableObject* que conté totes les estadístiques del personatge per poder agafar les afegides per aquest sistema.
- *maxHealth*(*float*): Valor aconseguit a través del *CharacterData*.
- *currentHealth*(*float*): Vida actual del personatge, comença sent igual que *maxHealth*.
- *maxPoise*(*float*): Valor aconseguit a través del *CharacterData*.
- *currentPoise*(*float*): Estabilitat actual, comença amb el valor de *maxPoise,* però no el pot superar.
- *timeTillPoiseRegeneration*(*float*): En aquesta variable s'assigna el valor *Time.time* més el valor *PoiseRegenerationDelay* del *characterData.*

En l'*Update* d'aquesta classe s'espera a que es pugui regenerar l'estabilitat per regenerar-la. La funció *IncreaseMaxPoise* rep un valor *float* el qual serveix per augmentar o disminuir l'estabilitat màxima en cas de necessitar-ho.

La funció de *TakeDamage* rep informació del personatge, arma i atac que ha rebut per disminuir la *currentHealth*, el *currentPoise* i, en cas d'estar bloquejant, la resistència del personatge. Si el *currentPoise* arriba o baixa de zero es regenera instantàniament i en el *CharacterStatesController* es canvia l'estat a l'estat de *HITSTUN*. En cas que la vida d'un personatge sigui igual o inferior a zero el personatge passa a l'estat de *DEAD*.

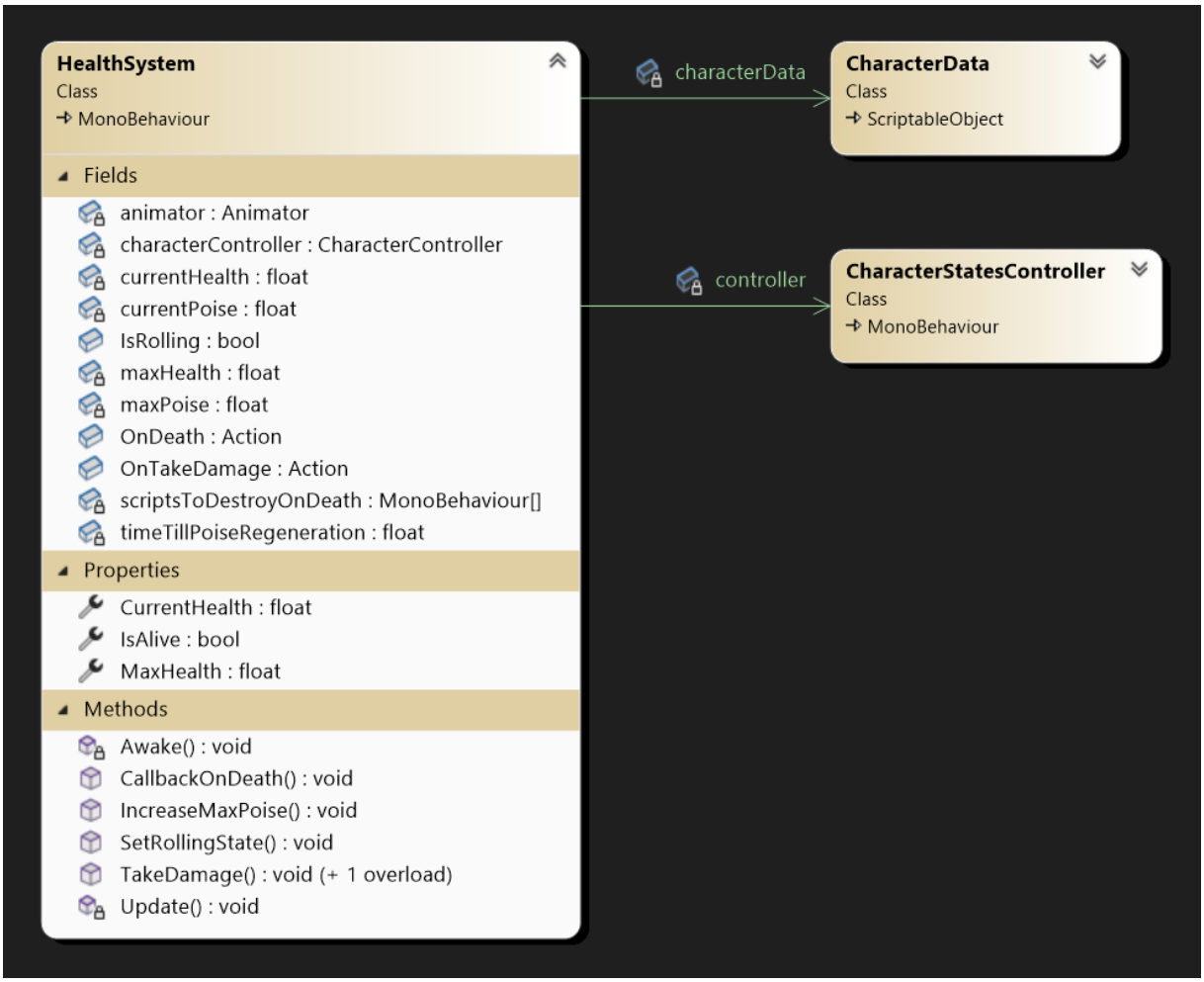

Figura 6.12. Diagrama UML simplificat del sistema de gestió de vida i estabilitat.
Font: Elaboració pròpia.

6.3.3.6 Gestió d'equipament

L'equipament pot ser una arma o un escut i cada equipament ha de poder efectuar fins a un màxim de dues accions en cada mà, per saber quina acció es pot fer en cada cas s'ha creat un *enum* amb el nom *EEquipmentAction* amb els membres *NONE*, *PRIMARY_ATTACK*, *SECONDARY_ATTACK* i *BLOCK*.

Per les dades de cada equipament dels personatges s'ha creat un *ScriptableObject* nomenat *EquipmentData*. Les variables de l'*EquipmentData* són:

- *PrimaryRightHand*(*EEquipmentAction*): Acció que fa l'equipament en fer l'input de l'acció primària de l'equipament quan es té equipada a la mà dreta.

- *SecondaryRightHand*(*EEquipmentAction*): Acció que fa l'equipament en fer l'input de l'acció secundària de l'equipament quan es té equipada a la mà dreta.
- *PrimaryLeftHand*(*EEquipmentAction*): Acció que fa l'equipament en fer l'input de l'acció primària de l'equipament quan es té equipada a la mà esquerra.
- *SecondaryLeftHand*(*EEquipmentAction*): Acció que fa l'equipament en fer l'input de l'acció secundària de l'equipament quan es té equipada a la mà esquerra.
- *BaseAttack*(*float*): Valor base de dany als punts de vida.
- *AttackStrengthScaling*(*float*): Valor multiplicat per l'estadística de força del personatge, la qual s'afegeix al dany d'un atac.
- *PoiseDamage*(*float*): Valor base de dany d'estabilitat.
- *PrimaryAttackCombo*(*AttackComboData*): Referència a la combinació d'atacs que pot fer l'arma en cas de ser una arma que pugui atacar en fer una acció primària.
- *SecondaryAttackCombo*(*AttackComboData*): Referència a la combinació d'atacs que pot fer l'arma en cas de ser una arma que pugui atacar en fer una acció secundària.
- *BlockDamageReduction*(*float*): Reducció de mal en rebre un atac mentre el personatge bloqueja.
- *BlockPoise*(*float*): Estabilitat afegida en bloquejar, també augmenta l'estabilitat màxima mentre s'està bloquejant.
- *BlockingData*(*BlockingData*): Les dades del subestat associat al bloqueig d'aquest equipament.

Per poder fer que un equipament en atacar pugui afectar al *HealthSystem* d'un enemic s'ha creat un script de tipus *MonoBehaviour* nomenat *HurtboxCollider* amb un atribut de *RequireComponent* perquè un cop afegit com a component hi hagi un component *BoxCollider*. El script s'ha afegit com a component a tot element que es fa servir com a arma. Aquest script té les funcions *ActivateCollider* i *DeactivateCollider* per activar i desactivar el component *BoxCollider*. En la funció *OnTriggerEnter* (Unity Scripting API - OnTriggerEnter) si es col·lideix amb un altre

personatge, es crida la funció de *TakeDamage* del *HealthSystem* del personatge impactat.

Per saber si el *HurtboxCollider* està en un equipament a la mà dreta o a l'esquerra s'ha fet un *enum* amb el nom *EWeaponSlot* amb els membres RIGHT i LEFT. En el *HurtboxCollider* s'ha creat una variable pública WeaponSlot(*EWeaponSlot*) la qual s'ha de modificar en l'inspector en cas de voler que la col·lisió sigui de la mà esquerra.

Per fer que un personatge no es pugui atacar a si mateix ni a cap personatge el qual no es vol que es pugui atacar s'ha creat un *enum* amb l'atribut *Flags* que s'ha nomenat *ECharacterType* el qual conté com a membre *PLAYER*, *ALLY*, *NEUTRAL* i *ENEMY*. Aquest *enum* funciona com a manera de definir equips. Cada personatge en el seu *CharacterStatesController* té una variable amb el nom *CharacterType* en la qual es defineix l'equip en el qual està el personatge i també una altra variable nomenada *CanAttackType* la qual són tots aquells equips als quals els atacs del personatge poden impactar.

Els atacs poden augmentar l'estabilitat mentre es duen a terme. Per activar i desactivar aquest augment i també les col·lisions de l'arma s'ha creat un script de tipus *MonoBehaviour* nomenat *AttacksHurtboxController* el qual s'ha afegit al *GameObject* que conté l'animator del personatge, d'aquesta manera es pot fer ús dels mètodes d'activació i desactivació d'estabilitat o col·lisions de les armes des de les mateixes animacions fent ús dels AnimationEvents (Unity Manual - Using Animation Events) per cridar aquestes funcions en el moment desitjat de l'animació.

També s'ha creat un mètode *AttackEnded* el qual si no s'ha desactivat la col·lisió o l'augment d'estabilitat els desactiva, aquesta funció serveix en cas de no arribar al punt de l'animació en el qual es criden els mètodes de desactivar col·lisió i augment d'estabilitat.

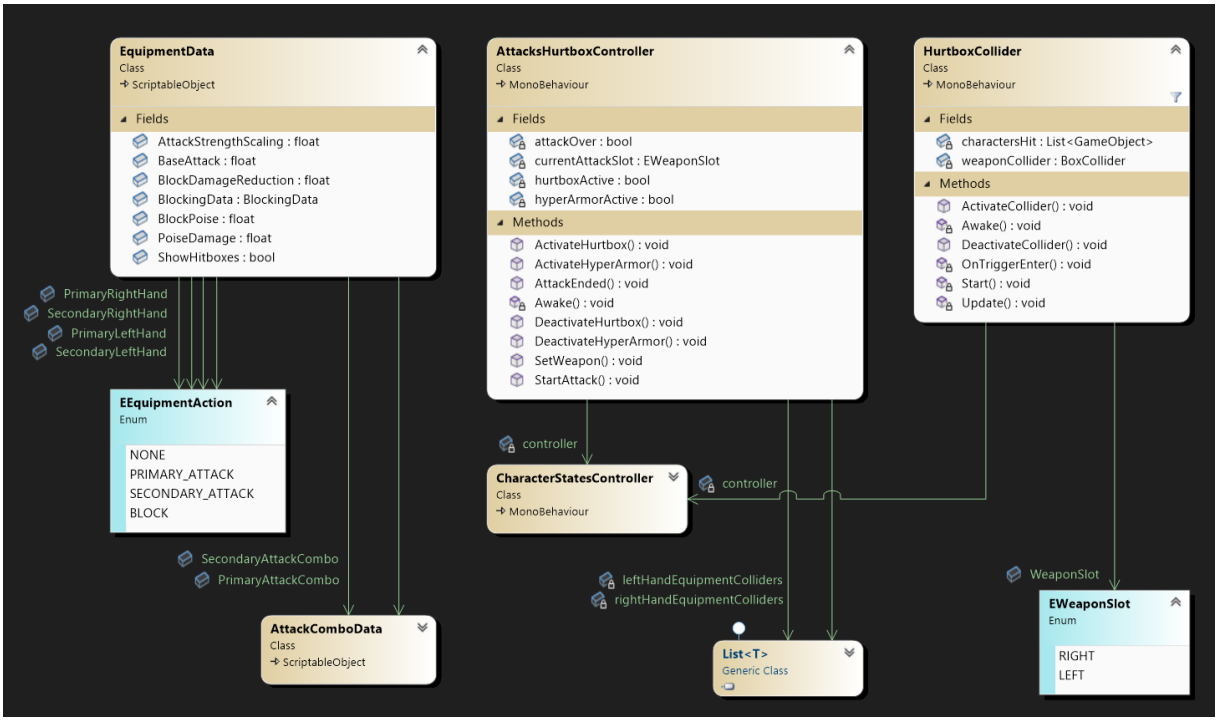

Figura 6.13. Diagrama UML simplificat del sistema de gestió d'equipament.

Font: Elaboració pròpia.

6.3.4 Desenvolupament de mecàniques

6.3.4.1 Càmera

Per la creació de la càmera s'ha creat un objecte de *Cinemachine* amb el component *CinemachineStateDrivenCamera,* el qual permet poder canviar entre una càmera interpretant cada càmera com un estat d'una màquina d'estats. Aquest component permet utilitzar un *Animator* per controlar el canvi i les interpolacions entre càmeres.

Per la càmera que el jugador pot moure amb un input s'ha creat una càmera *CinemachineFreeLook*. Per la càmera que fixa a un objectiu s'ha creat una càmera de tipus *CinemachineVirtualCamera.* En aquestes càmeres s'ha afegit una extensió de tipus *CinemachineCollider* perquè la càmera pugui col·lidir amb l'entorn i s'ajusti automàticament la distància de la càmera per no tenir cap obstacle endavant.

Per canviar l'estat del *CinemachineStateDrivenCamera* s'ha creat un script de tipus *MonoBehaviour* amb el nom de *CameraController* el qual s'ha afegit com a component al personatge que controla el jugador.

Pels estats de la càmera s'ha creat un *enum* nomenat *ECameraState* amb els membres *FREE* i *LOCKED*. Aquest script agafa l'*Animator* de la *CinemachineStateDrivenCamera,* i les dues càmeres que conté per poder modificar els paràmetres. Com a variables addicionals s'ha creat dos *float* per poder modificar la sensibilitat del moviment manual de la càmera.

Per poder fixar un objectiu s'ha creat un script de tipus *MonoBehaviour* nomenat *LockableTarget* el qual s'ha afegit com a component a tots els *GameObjects* que poden ser fixats per la càmera. Quan el *GameObject* del *LockableTarget* mor fa que totes les càmeres que el tenen fixat el deixin de fixar.

En l'*OnEnable* del *CameraController* s'assigna a la càmera el *Transform* al qual segueix i el *Transform* al qual la càmera enfoca, que en el cas de la càmera lliure és el mateix. Per fixar o desfixar objectiu es crida el mètode *AssignLockOnTarget* que rep un *LockableTarget.* Aquest mètode en cas de rebre un nul o un objectiu d'un tipus que el personatge no pot atacar la càmera passa a l'estat de *FREE*. En cas que sigui un objectiu vàlid s'agafa el *GameObject* del *LockableTarget* per fer que l'objectiu al qual ha de mirar la *CinemachineVirtualCamera* sigui l'enemic i es canvia l'estat a *LOCKED*. Aquest mètode es crida quan el *PlayerInputsManager* fa la crida que s'ha premut el botó de fixar objectiu.

Pel moviment de la càmera *CinemachineFreeLook* s'ha agafat l'input del ratolí i el del comandament per canviar la rotació en X i en Y.

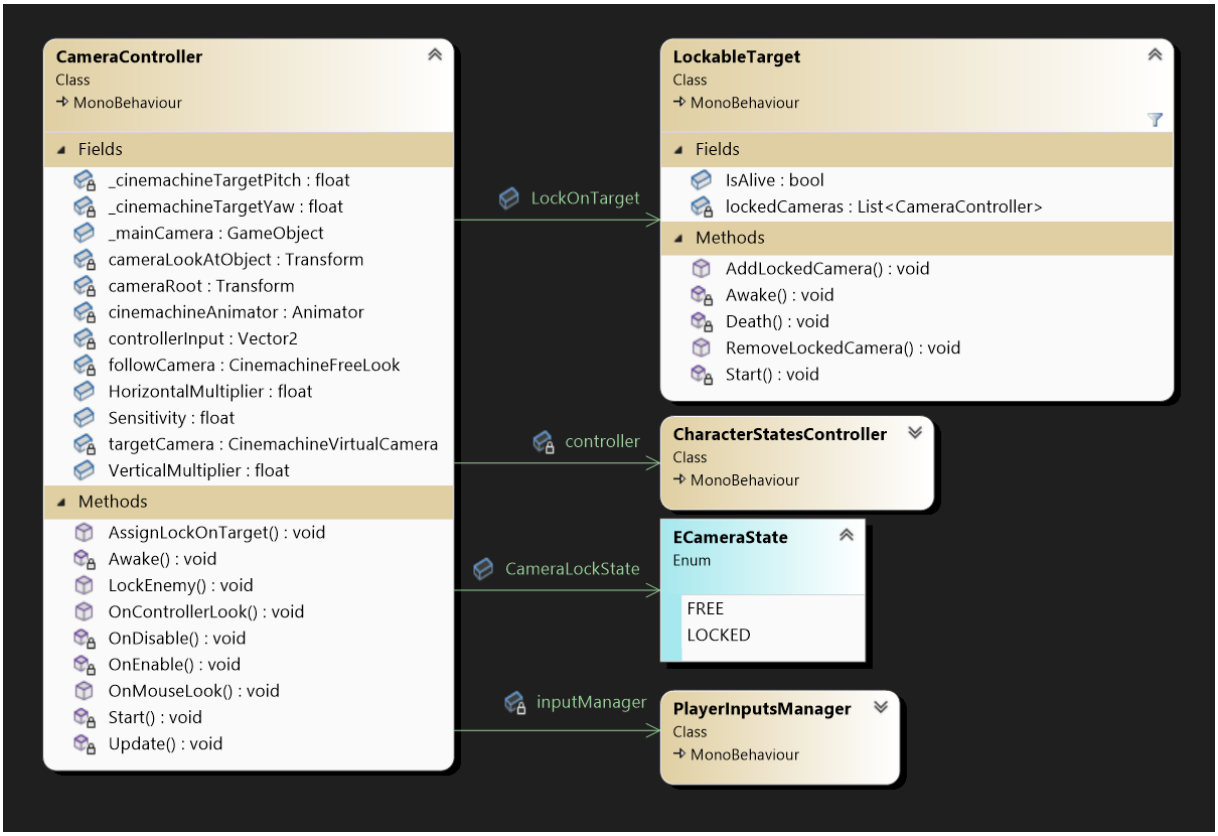

Figura 6.14. Diagrama UML simplificat del controlador de la càmera.

Font: Elaboració pròpia.

6.3.4.2 Estat de moviment

Per fer el moviment del *CharacterStatesController* s'ha creat una classe abstracta amb l'interfície de tipus *IStateBehaviour* nomenada *MovementState.* A partir d'aquesta classe s'han creat dues altres classes que hereten de *MovementState,* una pel moviment en el terra, la qual s'ha nomenat *GroundMovement* i l'altre pel moviment en l'aire el qual s'ha nomenat *AirbornMovement.*

Per les dades del moviment d'aquests estats que s'ha volgut que es modifiquin amb l'eina s'ha creat un *ScriptableObject* que hereta de *StateData* nomenat *MovementData* el qual conté:

- *MinMovement*(*float*): Moviment mínim d'un input necessari per iniciar un moviment.
- *SpeedMultiplier*(*float*): Multiplicador el qual s'aplica a la velocitat base del personatge per fer el moviment.

- *Acceleration*(*float*): Acceleració del moviment en cas que la velocitat a la qual es vol anar sigui menor a l'objectiu.
- *Deceleration*(*float*): Desacceleració del moviment en cas que la velocitat a la qual es vol anar sigui major a l'objectiu.
- *FaceLockedTarget*(*bool*): En cas d'estar activat el personatge sempre mirarà a l'objectiu en cas de tenir un objectiu fixat.
- *FaceLockedTargetWhenIdle*(*bool*): En cas d'estar activat el personatge sempre mirarà a l'objectiu en cas de tenir un objectiu fixat i no s'estigui en moviment.
- *LockedSpeedMultiplier*(*float*): Multiplicador de velocitat de moviment en cas d'estar fixant un objectiu.

En cada *OnUpdate* del *GroundMovement* es calcula la velocitat objectiu i es fa un *Lerp* per augmentar o disminuir la velocitat actual fins a arribar a la velocitat objectiu. Després del moviment es fa un *SmoothDampAngle* per canviar, de manera interpolada, la rotació del personatge a la del moviment o un objectiu fixat, en el cas que n'hi hagi un, i que els paràmetres del *MovementData* ho permetin.

La classe d'*AirbornMovement* funciona igual que la del *GroundMovement* exceptuant que llavors de fer-se servir l'input del jugador s'agafa la direcció del moviment en l'*OnEnter* i aquesta és la direcció en la qual rota el personatge.

Tant en *GroundMovement* com en l'*AirbornMovement* per definir el moviment s'agafa la rotació normalitzada del personatge en *Vector3*, es multiplica per la velocitat calculada i el *Time.deltaTime*.

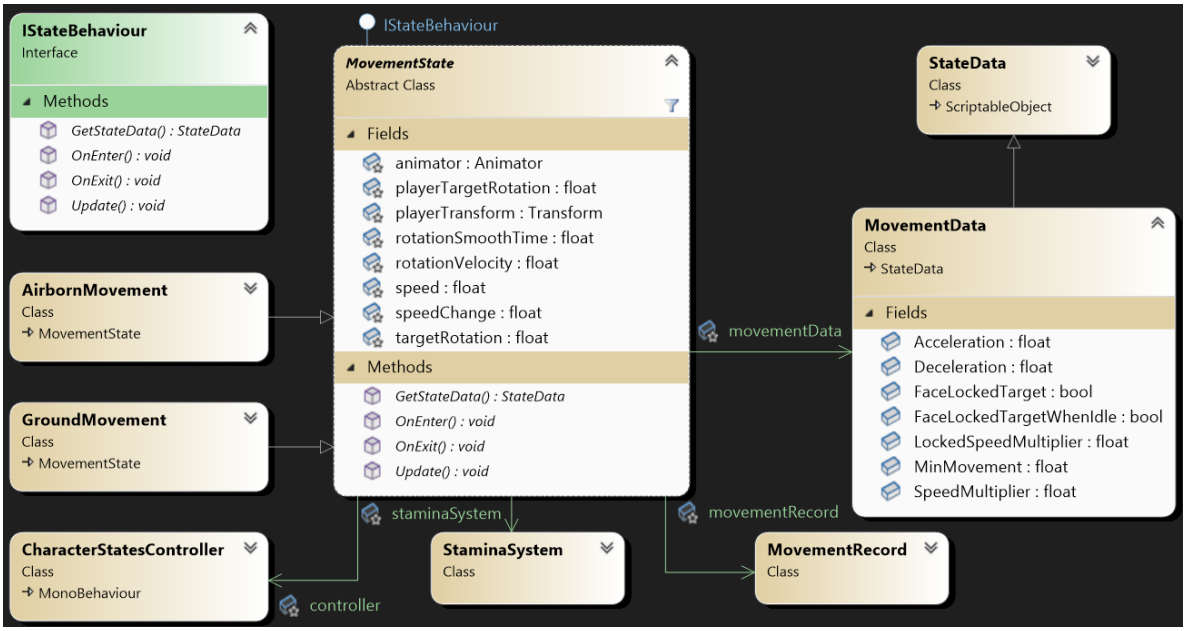

Figura 6.15. Diagrama UML simplificat de l'estat de moviment.

Font: Elaboració pròpia.

6.3.4.3 Estat d'evadir

Per l'estat d'esquiva s'ha creat una classe amb la interfície *IStateBehaviour* nomenada *DodgeState.* Per les dades modificables d'aquest estat s'ha creat un *ScriptableObject* que hereta de *StateData* nomenat *DodgeData* el qual conté:

- *MaxSpeed*(*float*): Velocitat màxima a la qual es pot arribar durant l'esquiva.
- *MovementCurve*(*AnimationCurve*): Corba en la qual en l'eix X és de zero fins a la duració de l'estat i la Y de zero a u. D'aquesta corba s'extreu el moviment fins a *MaxSpeed* relatiu al temps dins de l'estat.
- *TimeBeforeCantRotate*(*float*): Fins a arribar a aquest temps la rotació del personatge és igual a la de l'input de moviment, un cop s'acaba la rotació és fixa.
- *FaceLockedTarget*(*bool*): Funciona igual que en l'estat de moviment.
- *RotateAroundTarget*(*bool*): En cas d'estar activat i tenir un objectiu fixat el personatge a l'evadir no ho fa en línia recta a l'input de moviment sinó que a l'input es suma el moviment que faria en una circumferència, on l'objectiu fixat és el centre i la distància amb el personatge és el radi.

- *TimeBeforeIntangible*(*float*): Un cop el temps transcorregut en l'estat supera aquest valor s'activa la intangibilitat en vers atacs.
- *IntangibleDuration*(*float*): El temps que dura la intangibilitat.

Aquest estat funciona igual a l'estat de moviment amb dues excepcions. La primera és que no hi ha cap suavitzat en la rotació del personatge i un cop el temps de l'estat ha superat *TimeBeforeCantRotate* llavors el personatge ja no pot tornar a rotar. La segona és el fet que el moviment funciona amb una corba de moviment, per la qual cosa no hi ha una variable d'acceleració ni desacceleració. En acabar la duració de l'esquiva es fa un canvi d'estat a l'estat de moviment.

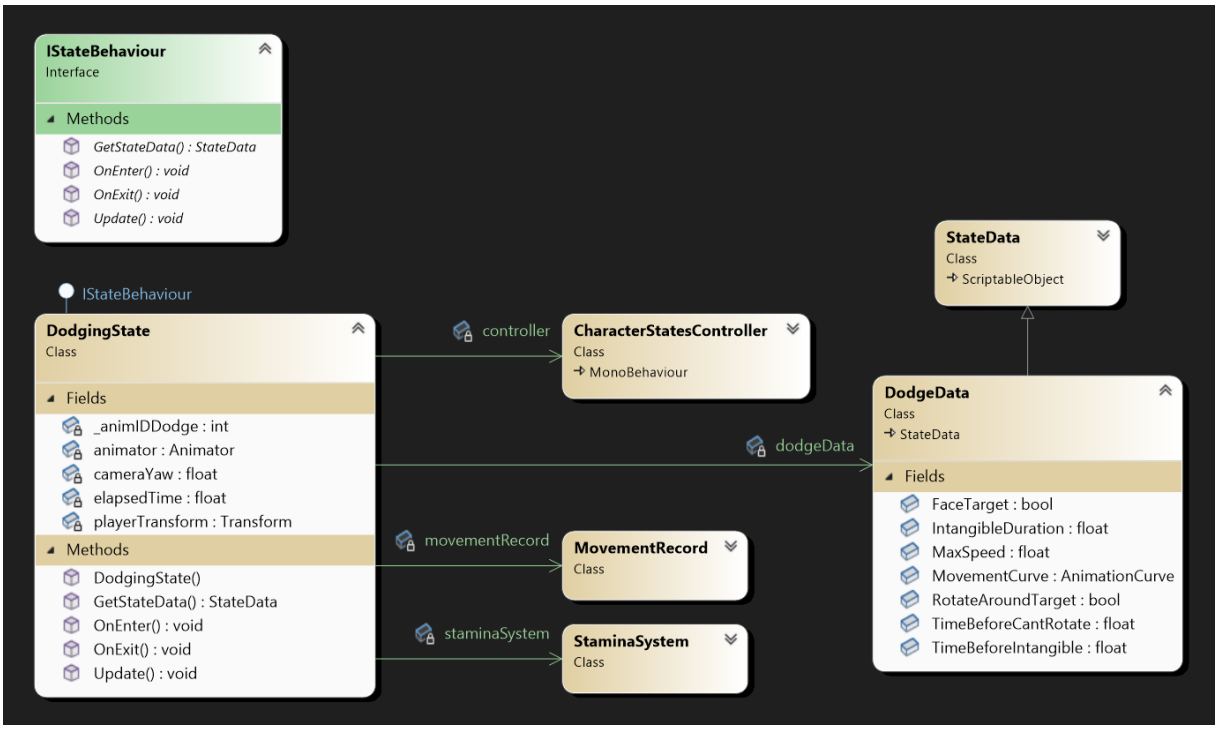

Figura 6.16. Diagrama UML simplificat de l'estat d'evadir.

Font: Elaboració pròpia.

6.3.4.4 Estat d'atacar

Pels estats d'atac s'ha creat una classe amb la interfície *IStateBehaviour* nomenada *AttackingState*. Per les dades modificables d'aquest estat s'ha creat un *ScriptableObject* que hereta de *StateData* nomenat *AttackData* el qual conté:

- *MaxSpeed*(*float*): Velocitat màxima a la qual pot arribar l'atac.

- *MovementCurve*(*AnimationCurve*): Moviment de l'atac durant el transcurs de l'estat.
- *TimeTillCantRotate*(*float*): El temps en el qual durant l'atac el personatge pot rotar abans de no poder fer-ho més.
- *RotationSpeed*(*float*): Velocitat a la qual pot rotar el personatge.
- *AttackTagetDirection*(*bool*): En el cas de ser cert i tenir un objectiu fixat l'atac sempre és cap a la direcció de l'objectiu.
- *TimeToComboAttack*(*float*): El temps abans que es torni al primer atac de la combinació d'atacs.
- *TwoHandedAttack*(*bool*): En cas de ser cert a l'activar col·lisions s'activen les de les armes de ambdues mans.
- *DamageMultiplier*(*float*): Multiplicador de dany en impactar a un enemic en el cas que es vulgui fer que cada atac amb la mateixa arma faci un mal diferent.
- *PoiseDamageMultiplier*(*float*): Multiplicador de dany d'estabilitat en impactar a un enemic.
- *HyperArmor*(*float*): Valor afegit a l'estabilitat del personatge mentre s'està atacant.

Com s'ha explicat en el sistema d'equipament cada arma pot fer fins a dues combinacions d'atacs, per fer aquestes combinacions s'ha creat un *ScriptableObject* nomenat *AttackComboData* el qual conté una llista del tipus *AttackData*.

En l'*OnEnter* de cada estat d'atac es comprova si l'atac és de la mateixa combinació d'atacs que l'atac anterior i també si ha passat menys temps que *TimeToComboAttack,* en cas afirmatiu es procedeix al següent atac de la llista d'atacs, en cas contrari es comença pel primer atac. Si s'arriba al final de la llista de combinacions d'atacs llavors es torna a l'inici.

L'*OnUpdate* funciona d'una manera similar a la de l'esquiva, a diferència que en aquest en el temps que es pot rotar ho fa amb un *SmoothDampAngle* com en l'estat de moviment. També en el cas de tenir activat *AttackTargetDirection* i un objectiu fixat la direcció de l'input del moviment és ignorada i la rotació objectiu és la de l'objectiu.

En l'*OnExit* es crida la funció *AttackEnded* de l'*AttacksHurtboxController* i en cas d'haver fet l'*OnExit* abans de temps es reinicia la combinació de la llista d'atacs.

En aquest estat també hi ha un temps, com en l'estat anterior, en el qual es pot fer una correcció del moviment abans de no poder rotar més.

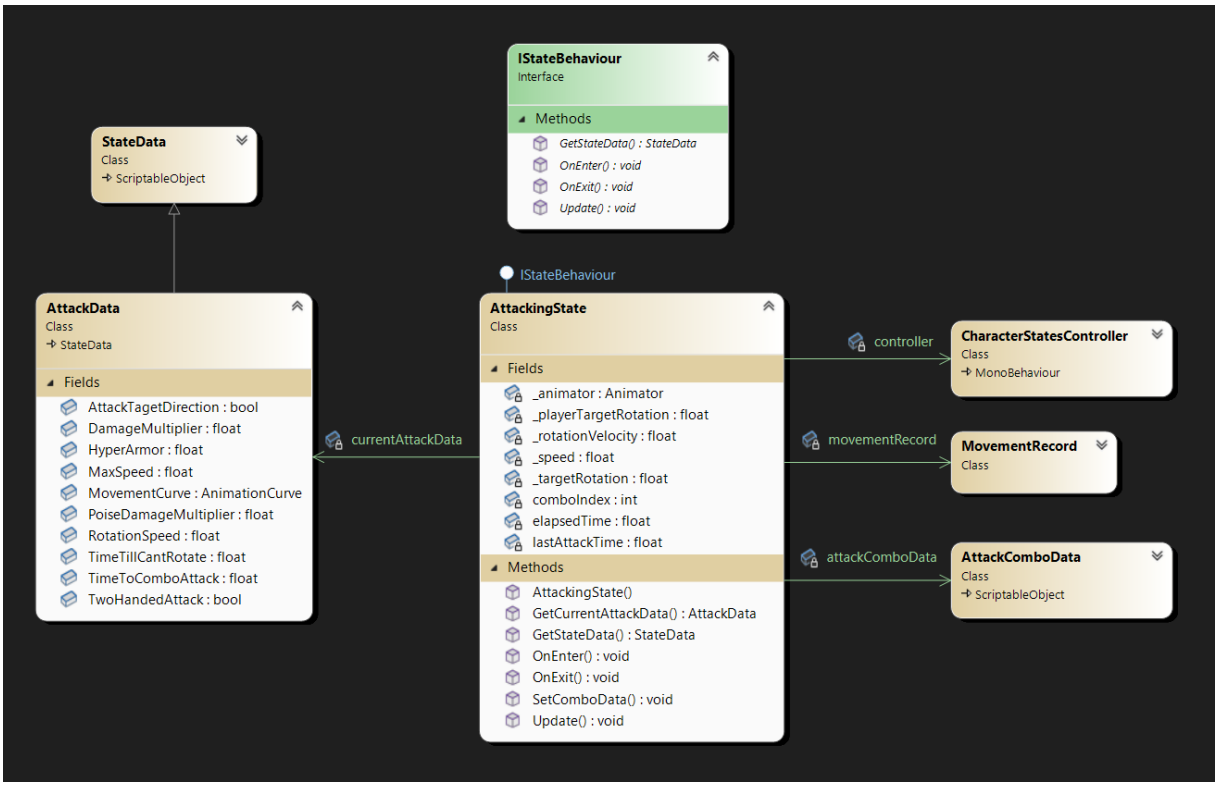

Figura 6.17. Diagrama UML simplificat de l'estat d'atacar.
Font: Elaboració pròpia.

6.3.4.5 Estat d'atordiment

Per l'estat d'atordiment s'ha creat una classe amb la interfície *IStateBehaviour* nomenada *HitstunState*. Per les dades modificables d'aquest estat s'ha creat un *ScriptableObject* que hereta de *StateData* nomenat *HitstunData* el qual no té cap variable extra, ja que les úniques variables que fa servir l'estat d'atordiment són el *AnimationID* i la duració de l'estat.

Per evitar que un personatge estigui constantment en estat d'atordiment s'ha fet que si s'entra en l'estat d'atordiment abans d'acabar l'anterior llavors la duració de l'atordiment es divideix entre el nombre d'atordiments consecutius més u. Cada cop

que s'acaba un estat d'atordiment el nombre d'atordiments consecutius es posa a zero.

Figura 6.18. Diagrama UML simplificat de l'estat d'atordiment.
Font: Elaboració pròpia.

6.3.4.6 Subestat de bloqueig

Pels subestats s'han creat dues classes amb la interfície *ISubstateBehaviour,* una ha sigut *NothingSubstate* i *BlockingSubstate*. *NothingSubstate* és una classe amb implementació del *ISubstateBehaviour* amb els mètodes buits la qual es fa servir com a subestat actual sempre que no hi hagi cap subestat actiu. Per les dades modificables del subestat de bloqueig s'ha creat un *ScriptableObject* nomenat *BlockingData* el qual conté:

- AnimationID(string): Nom de la transició de l'*AnimatorController* per efectuar l'animació corresponent.
- MovementSpeedMultiplier(float): Multiplicador de velocitat de moviment.
- TimeToStartBlocking(float): Temps necessari en l'estat per començar a bloquejar atacs.

En l'*OnEnter* i *OnExit* es modifica el multiplicador de moviment del *MovementRecord* i en l'*OnUpdate* es compta el temps que ha passat durant de l'estat i quan és major que *TimeToStartBlocking* augmenta l'estabilitat del *HealthSystem* i en rebre un atac es redueix el dany rebut depenent de l'estadística de bloqueig de l'equipament.

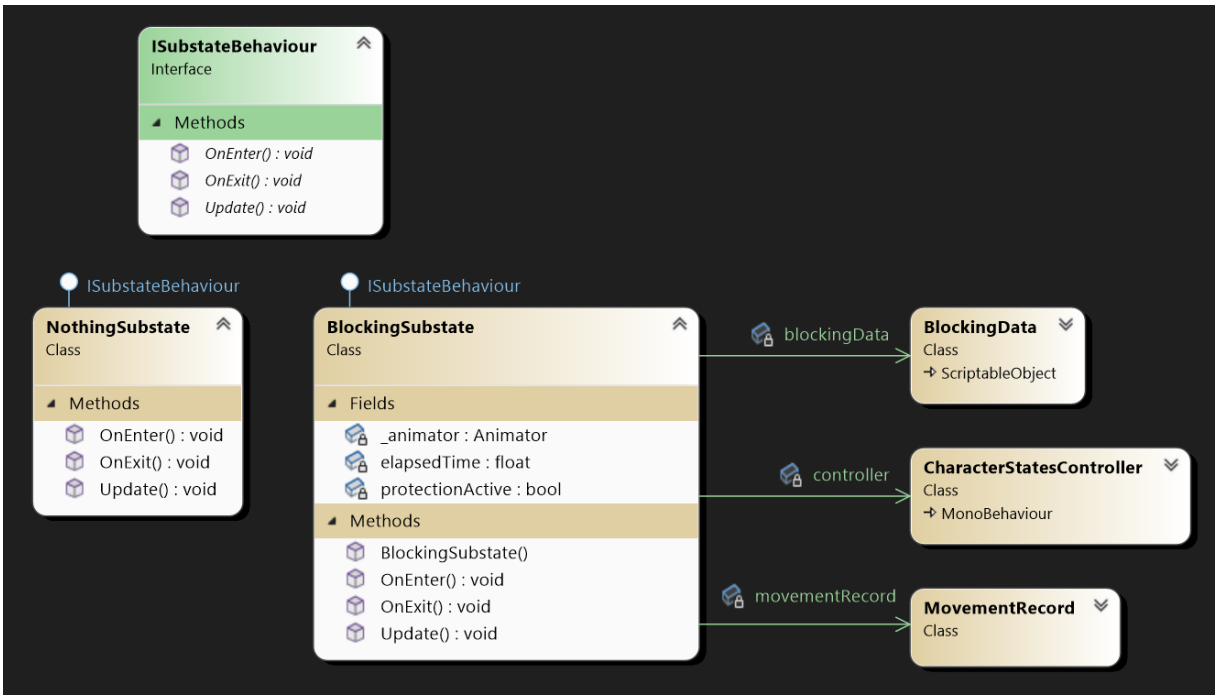

Figura 6.19. Diagrama UML simplificat del subestat de bloquejar i nul.

Font: Elaboració pròpia.

6.3.4.7 Estat de físiques

Per fer les físiques del personatge s'ha creat una classe nomenada *PhysicsState.* A partir d'aquesta classe s'han creat dues altres classes que hereten de *PhysicsState,* una per les físiques en el terra, la qual s'ha nomenat *PhysicsGrounded* i l'altre per físiques en l'aire el qual s'ha nomenat *PhysicsAirborn.*

En l'*OnUpdate* del *PhysicsState* s'aplica al moviment del *MovementRecord* la gravetat i les forces en cas d'haver rebut una empenta.

En l'*OnUpdate* del *PhysicsAirborn* i del *PhysicsGrounded* es comproven les *CollisionFlags* del *CharacterController* per saber si el personatge està tocant un objecte per la part superior i la part inferior.

En l'estat de *PhysicsAirborn* quan es toca un objecte per la part superior i tenir una velocitat vertical positiva la velocitat es torna zero. En cas de tocar un objecte per la part inferior es canvia l'estat de físiques del *CharacterStatesController* a *GROUNDED*.

En l'estat de *PhysicsGrounded* quan es deixa de tocar el terra s'espera 0,15 segons i, si durant aquest temps no s'ha tornat a tocar terra, es canvia l'estat de físiques a *AIRBORN*.

6.4 Creació de la interfície gràfica

Dintre de l'editor de *Unity* s'ha creat una finestra per poder mostrar de manera més organitzada tots els paràmetres a modificar per l'usuari. Tots aquells elements que es vol que un usuari pugui modificar per personalitzar cada personatge separat en aquesta finestra. L'objectiu d'aquesta finestra és poder crear i modificar els elements existents d'un personatge de manera més àgil que fent servir l'explorador i l'inspector de *Unity*.

6.4.1 Personalització d'inspector en Unity

L'inspector de *Unity* és un element de la UI del software que permet afegir components als GameObjects i en aquests components, que realment són scripts, modificar paràmetres o referències dels GameObjects de l'escena. Aquests components poden ser donats pel mateix motor, per un package introduït al projecte o creat pel mateix usuari. En codi, si s'està fent servir l'Using d'*UnityEngine*, es poden modificar una mica com s'exposen les variables en l'inspector.

Figura 6.8. Mostra de variables exposades al inspector de *Unity*.

Font: Elaboració propia.

6.4.1.1 CustomEditor

De base l'inspector de *Unity* té poques funcionalitats i opcions de personalització de contingut. En cas de voler afegir botons o modificar la manera en la qual la informació es mostra s'ha de crear una classe de tipus Editor amb un atribut de *CustomEditor* del tipus de la classe a la qual volem editar aquest editor personalitzat i fer un using del *UnityEditor* per poder accedir a tot el codi relacionat amb l'editor.

```
using UnityEngine;
using UnityEditor;

[CustomEditor(typeof(PlayerHealthSystem))]
Script de Unity | 0 referencias
public class PlayerHealthSystemEditor : Editor
```

Figura 6.9. Codi necessari per modificar l'inspector de *Unity*. Font: Elaboració propia.

6.4.1.2 Elements de GUI

Un cop s'ha heretat de la classe *Editor* es pot sobreescriure la funció d'escriptura d'editor *OnInspectorGUI*(), en cas de no voler escriure tot des de zero s'ha de cridar la funció base per després escriure els nous elements que es volen col·locar, com per exemple un text i un botó per diferenciar l'inspector normal del personalitzat.

```
public override void OnInspectorGUI()
{
    base.OnInspectorGUI();

    PlayerHealthSystem manager = target as PlayerHealthSystem;

    if (manager == null)
        return;

    GUILayout.Label("\nDebug Mode");

    if (GUILayout.Button("Increase 10 HP"))
    {
        manager.GetComponent<PlayerHealthSystem>().IncreaseHealth(10f, 0f);
    }
```

Figura 6.10. Codi necessari per afegir un botó a l'inspector de *Unity*.
Font: Elaboració propia.

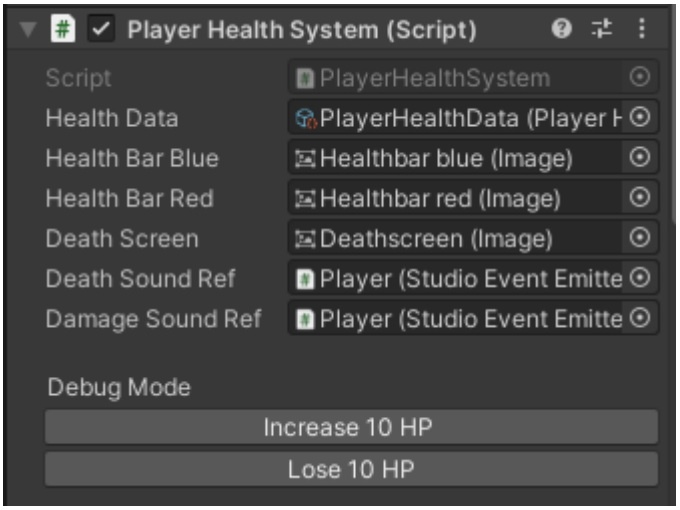

Figura 6.11. Mostra de com queden els botons en l'inspector de *Unity*.

Font: Elaboració propia.

Per escriure l'inspector des de zero el primer paràmetre al qual s'ha d'accedir és al paràmetre *serializedObject,* aquest paràmetre representa l'objecte que s'està sent inspeccionat en el mateix inspector (Unity Scripting API - serializedObject). Per aconseguir una dada d'un *serializedObject* es fa servir el mètode *FindProperty* que busca una variable la qual coincideixi el nom amb el *string* introduït com a paràmetre (Scripting API - FindProperty). La millor manera de buscar una variable de la qual ja se sap el nom és posant nameof() i dins el nom de la classe un punt i el nom de la variable que es vol aconseguir.

FindProperty retorna una *SerializedProperty* de la qual es pot extreure el valor, també es pot directament introduir en un *PropertyField* el qual agafa el tipus de la propietat automàticament i exposa en la UI el valor i el tipus corresponent d'element de la UI per modificar la variable.

6.4.2 Finestres d'editor en Unity

Tota la UI de *Unity* ve amb diverses finestres que compleixen una funció en específic, i en aquelles finestres es faran servir per desenvolupar el projecte. De la mateixa manera que *Unity* proporciona funcions les quals es poden utilitzar per personalitzar l'inspector, també hi ha funcions que permeten crear finestres i que es puguin usar de manera aïllada. Per una eina complexa és molt millor tenir una finestra específica per l'eina i no treballar tot amb l'inspector dels objectes de *Unity*.

6.4.2.1 EditorWindow

De la mateixa manera que es pot modificar l'apartat de l'inspector de l'editor de *Unity* també es pot fer amb una finestra d'editor si s'hereta de la classe *EditorWindow* amb l'using de *UnityEditor*. Les funcions a sobreescriure del *EditorWindow* són les mateixes que en el *CustomEditor*, a diferència de modificar la part de l'editor de l'inspector és que no tenim una base que ja ens mostri variables, aquí tot s'ha d'escriure des de zero.

6.4.2.2 Elements de GUI

De la mateixa manera que la finestra d'editor personalitzada funciona de manera similar que en l'inspector de *Unity*, els elements de la GUI també funcionen igual.

6.4.3 Disseny de la finestra

Els elements que s'han escollit per mostrar en la finestra són aquells els quals són dades que poden diferir entre personatges i que, a l'hora de la implementació, s'han creat com a ScriptableObjects. Aquestes dades són:

- Dades del personatge
- Dades d'un atac
- Dades d'un bloqueig
- Dades d'una combinació d'atacs
- Dades d'una esquiva
- Dades d'un equipament
- Dades d'un moviment
- Dades de les físiques del personatge
- Dades de l'estat d'atordiment

Per poder fer servir l'eina de manera fluida s'ha escollit que a l'esquerra de la finestra hi hagi una pestanya especifica per cada dada que es pot modificar, d'aquesta manera quan s'està en la pestanya seleccionada només es mostra les dades d'aquell apartat. Les variables dels ScriptableObjects es mostren agrupades per funcionalitats, ja que hi ha variables que serveixen per aspectes diferents d'ltres.

El mock-up s'ha creat per tenir una base i comprovar l'organització proposada. Durant el desenvolupament de l'eina s'han vist modificades, eliminades o afegides algunes variables per millorar la comoditat a l'hora d'usar l'eina.

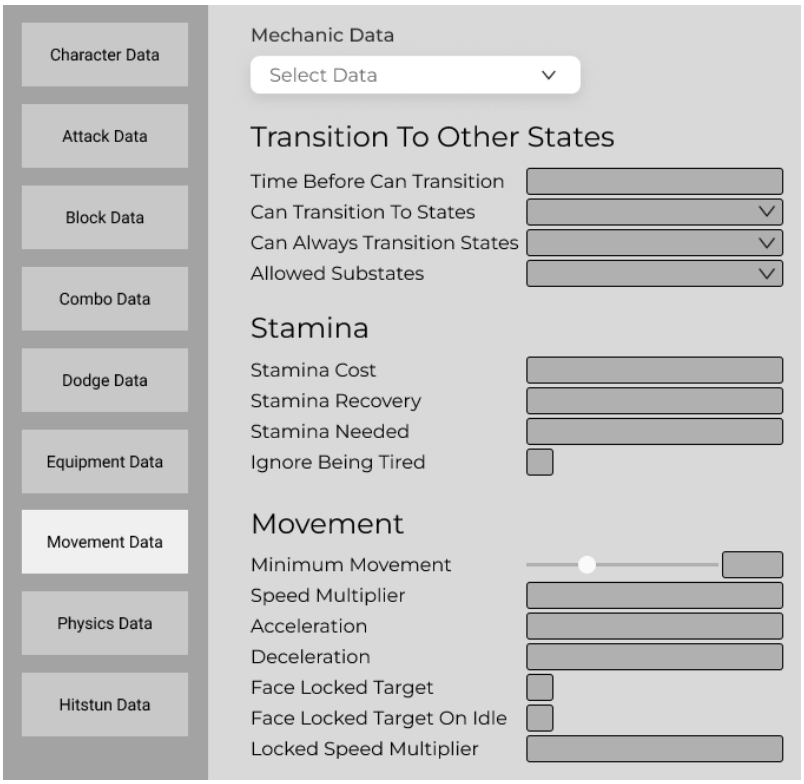

Figura 6.12. Captura del mock-up fet en Figma de la pestanya de les dades d'atordiment. Font: Elaboració propia.

6.4.4 Desenvolupament de la finestra

6.4.4.1 Creació de la finestra

Per poder accedir a la finestra s'ha creat una funció pública estàtica i definit un atribut MenuItem. Aquest atribut permet crear una pestanya a la part superior esquerra dins de l'editor de *Unity* amb el nom que se li hagi posat com a paràmetre, en el nom cada "/" serveix per fer una secció dins de la pestanya. En seleccionar la pestanya s'efectua el codi escrit en la funció, en aquest cas es crea la finestra que s'ha fet servir per mostrar les dades a modificar de l'eina.

```
[MenuItem("Souls/Character Settings")]
public static void ShowWindow()
{
    var instance = GetWindow<SoulsLikeWindow>();
    instance.ShowAuxWindow();
}
```

Figura 6.13. Codi per mostrar la finestra de l'eina. Font: Elaboració pròpia.

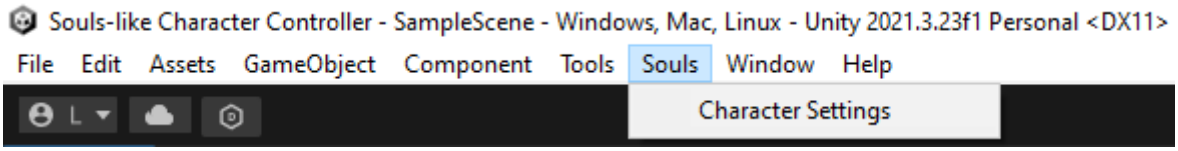

Figura 6.14. Visualització del resultat del codi mostrat en la figura anterior. Font: Elaboració pròpia.

Per aconseguir dividir en apartats de la mateixa manera que s'ha fet en el mock-up del Figma s'ha creat una funció que genera botons en vertical a l'esquerra de la finestra, els botons s'han designat en un *enum* per saber quin contingut de dades mostrar i en una llista els noms que es mostren en els botons, cada element de *l'enum* ha de tenir un *string* en la llista de noms en la mateixa posició.

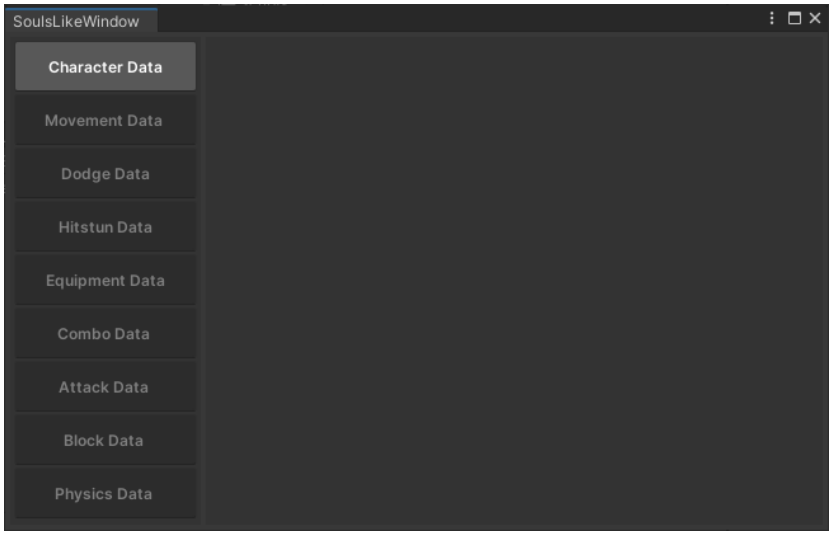

Figura 6.15. Visualització del resultat de la funció de mostra de botons. Font: Elaboració pròpia.

Per mostrar la informació de cada conjunt de dades s'ha creat una funció de pintat de pàgina per cada un. Per saber quina pàgina pintar s'ha fet un *switch* en cada

pintat del GUI en el qual, depenent de la pestanya del *enum* seleccionada, cada *case* del *switch* efectua la funció de pintat de la pàgina.

En cada pàgina s'ha creat un selector de *ScriptableObjects* del tipus de la pàgina, en el cas del *CharacterData* només permet seleccionar *ScriptableObjects* del tipus *CharacterData*. I també en el cas que no existeixi cap *ScriptableObjects* creat del tipus de la pàgina s'ha afegit en la part superior un espai per a poder crear un nou arxiu del tipus de la pàgina amb el nom que vulgui l'usuari. El botó per crear un nou arxiu només es pot prémer quan hi ha un nom, en cas contrari es manté desactivat. Quan es genera un *ScriptableObject* nou s'introdueix en una carpeta nomenada Data que està dins de la carpeta Assets del projecte.

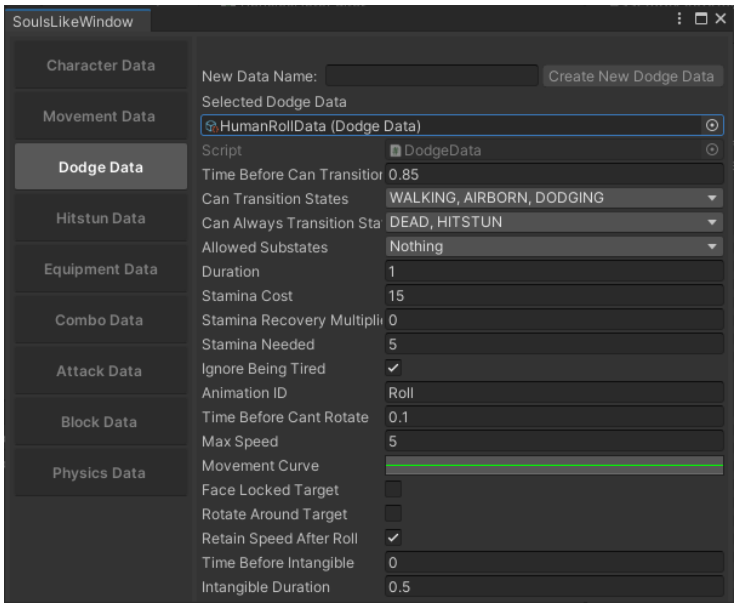

Figura 6.16. Visualització del resultat de selector i creador de ScriptableObjects i de la mostra de les variables del mateix. Font: Elaboració pròpia.

6.4.4.2 Creació dels inspectors personalitzats

Per fer la divisió entre variables s'han creat inspectors personalitzats per cada *ScriptableObject*. En cada inspector personalitzar s'ha agrupat les variables dintre d'una caixa d'inspector i a cada caixa d'inspector se li ha donat un títol per deixar clar a l'usuari a quin sistema afecten aquelles variables.

Un dels avantatges que té fer un inspector personalitzat és el de poder delimitar o donar valors base a variables que tenen un valor nul. En el cas de les corbes de

moviment s'ha canviat el color de la corba, també s'ha fet que si la corba no té cap keyframe, llavors s'afegeixen automàticament dos keyframes, un a la coordenada 0,0 i l'altre a la coordenada 1,1 i per últim s'ha delimitat que l'alçada de la corba sigui sempre u i la llargària és igual a la duració del moviment.

Un cop s'han afegit aquests inspectors personalitzats la finestra s'ha completat i està llesta per l'ús de l'usuari.

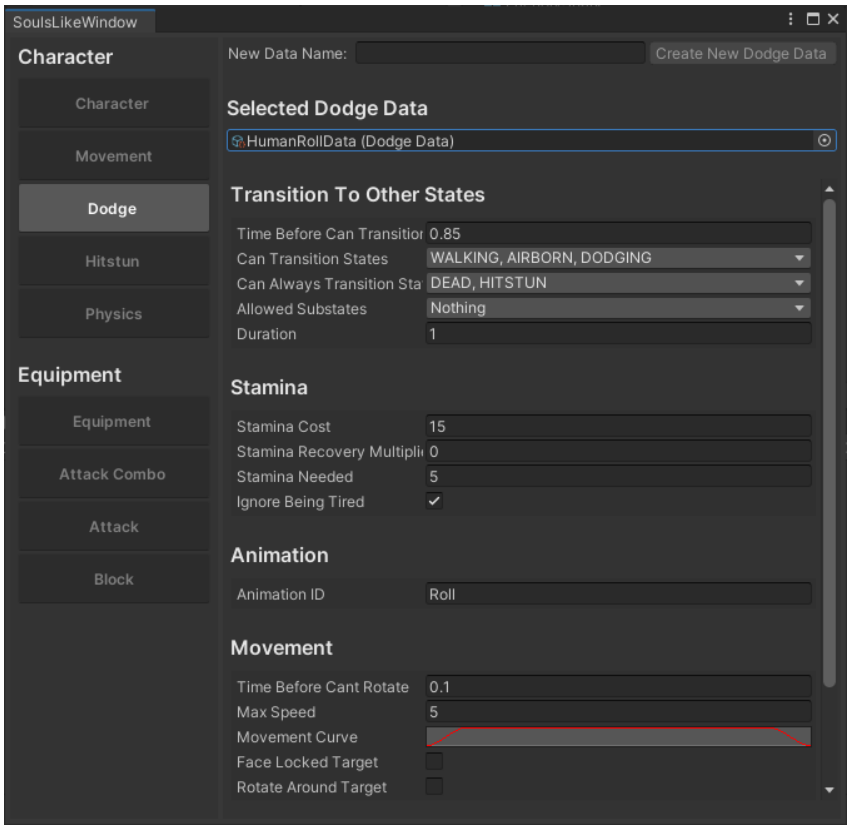

Figura 6.17. Visualització del resultat de l'inspector personalitzat de les dades de l'esquiva. Font: Elaboració pròpia.

6.5 Demostrador

Per validar el correcte funcionament dels sistemes i la finestra personalitzada de l'eina s'han creat diversos personatges cada un amb característiques diferents dels altres.

Pel primer personatge s'ha fet servir un model i múltiples animacions extretes de la pàgina web *Mixamo* (Mixamo, 2009) pel personatge del controlador, el qual s'ha posat en escena amb un *AnimatorController* (Unity Manual - Animator Controller) per

poder canviar d'animacions i un *CharacterController* (Unity Scripting API - CharacterController) per poder fer el moviment del *GameObject* en escena.

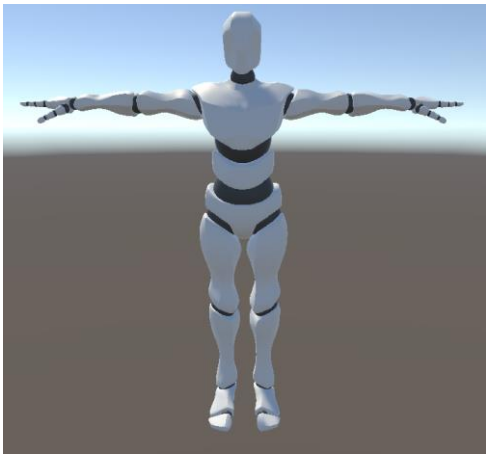

Figura 6.18. Captura en *Unity* del model del personatge importat de *Mixamo*.
Font: Elaboració Pròpia.

Amb *ProBuilder* s'ha fet un escenari per l'espai simulat en el qual poder moure el personatge. Aquest escenari consta de diverses rampes de diferents alçades i tres obstacles de diverses mides per comprovar les funcionalitats bàsiques del moviment del personatge i la càmera.

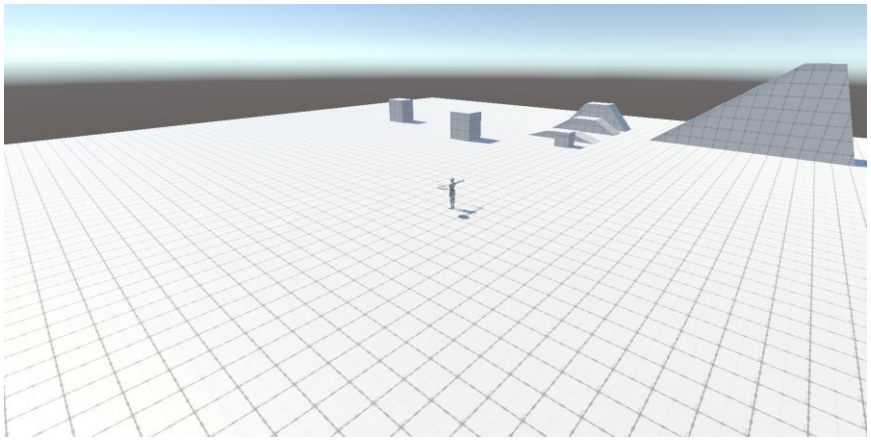

Figura 6.19. Captura en *Unity* de l'escenari creat amb l'eina *ProBuilder*.
Font: Elaboració Pròpia.

6.5.1 Implementació de les mecàniques

Cada mecànica, un cop programada, s'ha provat en el personatge de l'escena. S'ha creat un *AnimatorController* perquè cada estat pugui controlar, mitjançant els

paràmetres de l'*Animator*, l'animació que es mostra en cada moment. El personatge s'ha creat amb característiques equilibrades, prioritzant un moviment fluid entre estats.

Per la mort del personatge s'ha fet que el personatge pugui ser un *Ragdoll* (Unity Manual - Create a ragdoll). Per l'activació d'aquest *Ragdoll* s'ha creat un script *MonoBehaviour* nomenat *RagdollActivator* el qual crida un mètode en morir que desactiva l'*AnimatorController* del personatge i activa les físiques del *Ragdoll.* En cas de voler fer una animació i no un *Ragdoll* s'ha posat l'opció en l'editor del *CharacterData* per permetre-ho.

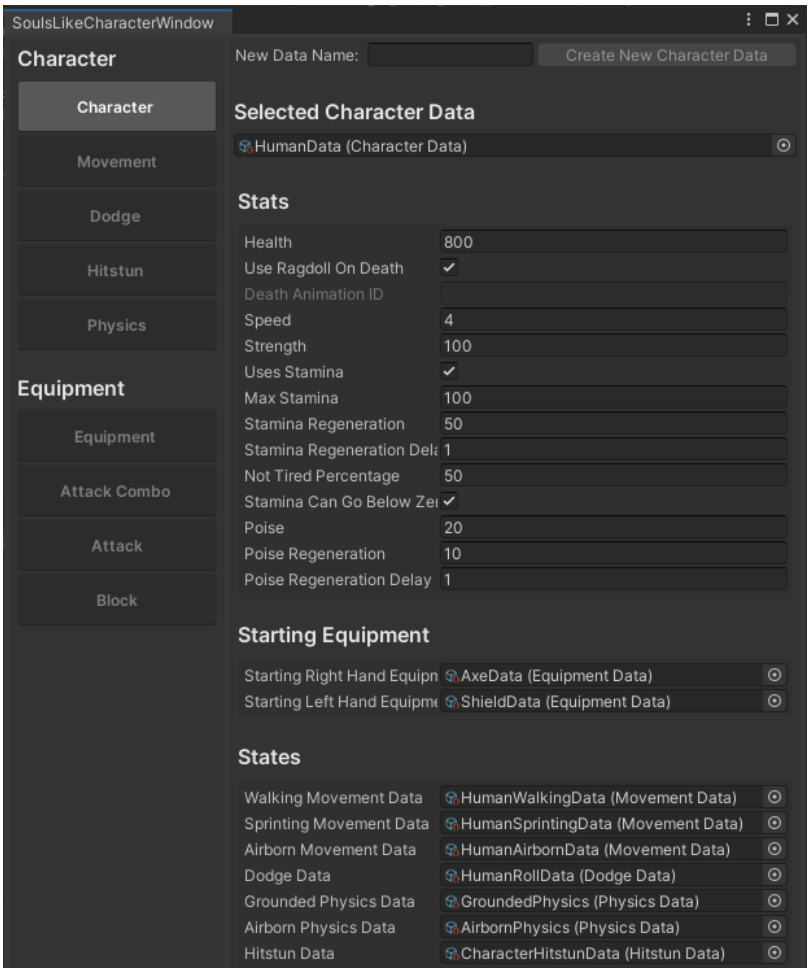

Figura 6.20. Configuració de les dades del personatge. Font: Elaboració pròpia.

6.5.1.1 Moviment

Pel *MovementState* s'han creat tres tipus de moviment, caminar, esprintar i caure. Per poder canviar, de manera còmoda en la finestra de l'eina dins de la pestanya de

la configuració, els diferents moviments d'un personatge en la finestra de l'eina s'ha creat un desplegable on es mostren els moviments del personatge.

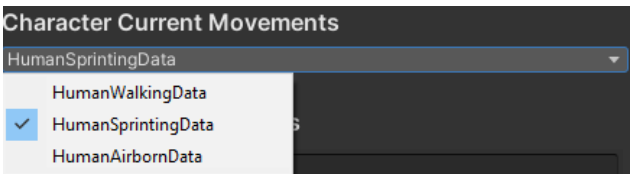

Figura 6.21. Desplegable dels diferents moviments d'un personatge.

Font: Elaboració Pròpia.

Pel moviment en el terra s'ha creat un *Blend Tree* (Unity Manual - Blend Trees) d'una dimensió en el *AnimatorController* el qual conté les animacions d'estar quiet, la de caminar i la de córrer. El paràmetre del qual depèn el *Blend Tree* és la velocitat actual dividida per la velocitat del personatge.

Quan un personatge té un objectiu fixat el *Blend Tree* efectuar una transició a un altre el qual és de dues dimensions i té un moviment per cada punt cardinal.

En l'estat de caminar s'inclou estar quiet, per la qual cosa no existeix un estat de "Idle". Si l'input de moviment del jugador és analògic, el moviment és gradual. Aquest estat no fa ús de l'*AnimationID,* ja que el moviment funciona pel *Blend Tree* de l'*Animator*. La configuració escollida per l'estat de caminar del personatge ha sigut:

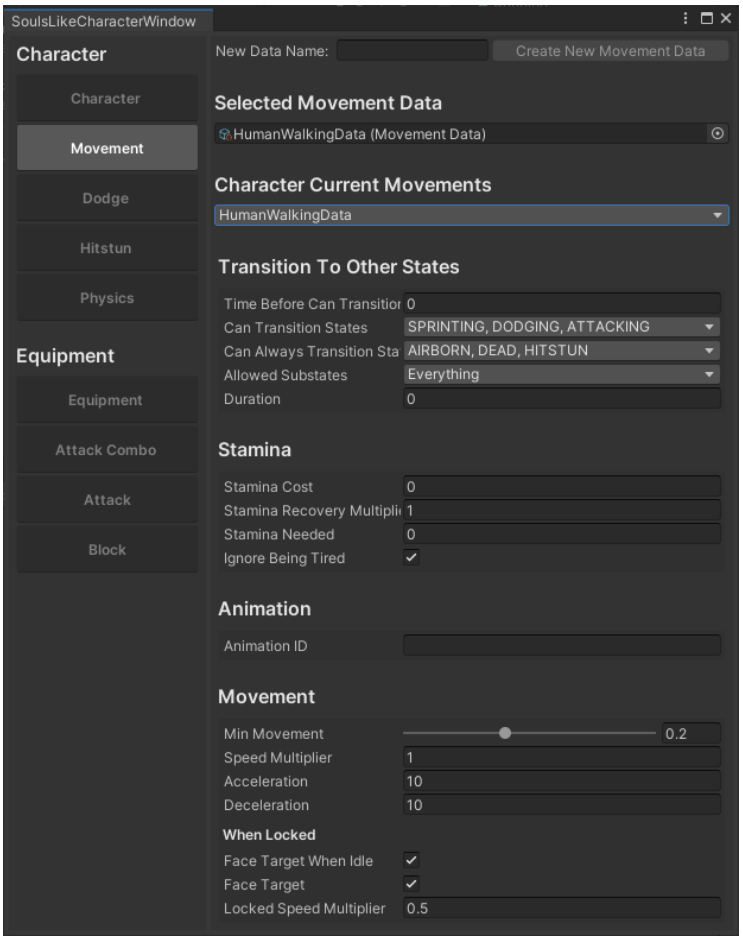

Figura 6.22. Configuració de les dades de l'estat de caminar. Font: Elaboració pròpia.

En mantenir premut el botó d'evadir s'efectua la transició a esprintar. En l'estat d'esprintar el personatge no es mou de manera gradual, sinó a la velocitat designada. La configuració escollida per aquest estat del personatge ha sigut:

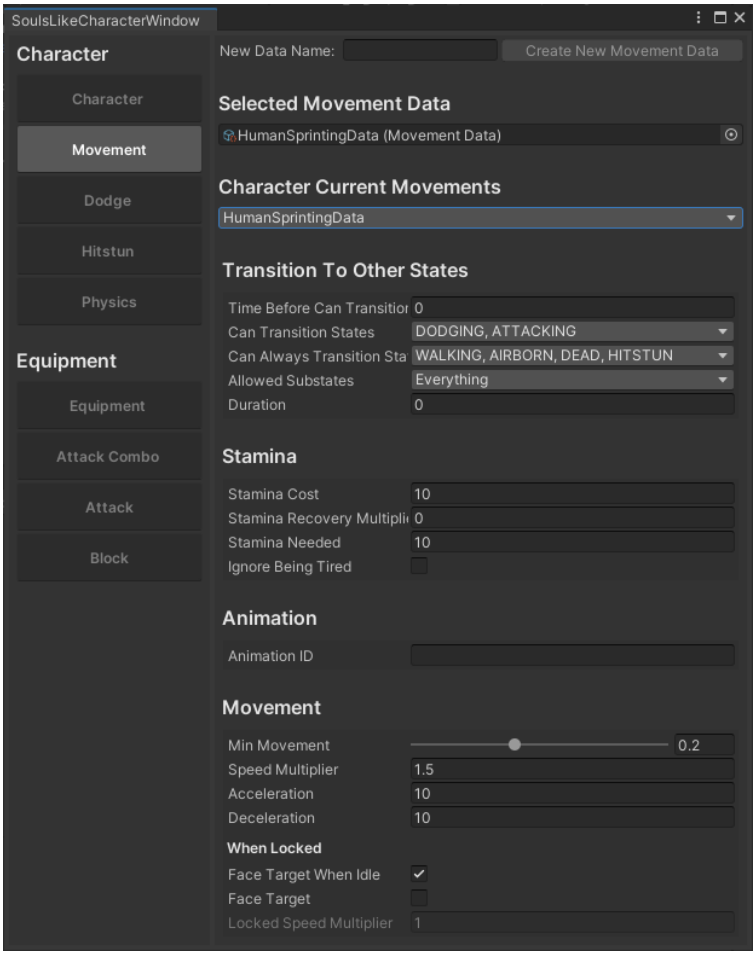

Figura 6.23. Configuració de les dades de l'estat d'esprintar. Font: Elaboració pròpia.

Sempre que el personatge estigui en l'aire, i l'estat actual ho permeti, el personatge efectuar una transició a l'estat de caure, en el qual el personatge no pot rotar. En aquest estat s'ha escollit que no es pugui dur a terme cap acció de manera voluntària, bloquejant així al personatge fins a tocar terra.

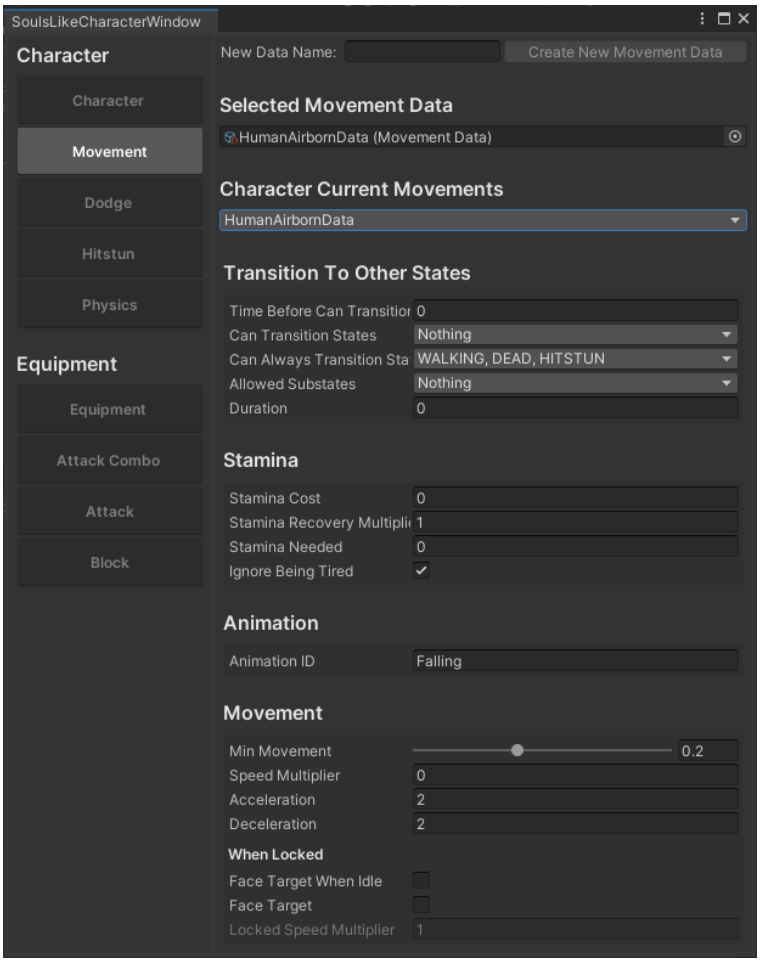

Figura 6.24. Configuració de les dades de l'estat de caure. Font: Elaboració pròpia.

6.5.1.2 Esquiva

Per l'esquiva d'aquest personatge s'han creat dues diferents. L'esquiva del personatge es defineix en la configuració del mateix personatge, no es poden tenir les dues alhora. Durant el transcurs de les esquives no es permet subestats, per la qual cosa no es pot bloquejar mentre s'esquiva.

Per la primera esquiva s'ha escollit una animació de rodar, la qual s'ha posat com a estat a l'animator. La direcció de l'esquiva és mateixa de l'input a l'inici. La configuració d'aquesta esquiva és la següent:

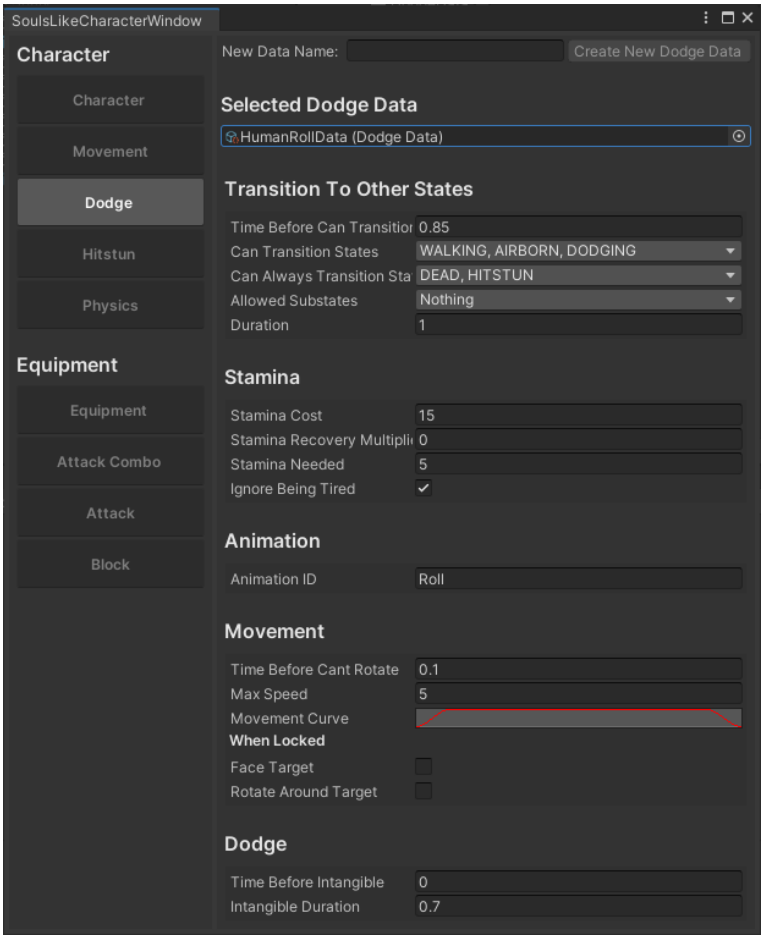

Figura 6.25. Configuració de les dades del primer estat d'evadir.

Font: Elaboració pròpia.

Per la segona esquiva s'ha escollit quatre animacions de salt curt, les quals s'han posat en un *Blend Tree* de dues dimensions en els quatre punts cardinals. Quan es fa l'esquiva amb un objectiu fixat, la rotació del personatge mira a l'objectiu i s'agafa la direcció de l'input relativa a l'objectiu per aplicar-la al *Blend Tree.* El moviment d'aquesta esquiva busca rotar al voltant de l'objectiu, per la qual cosa el moviment com més lateral més corbat. La configuració d'aquesta esquiva és la següent:

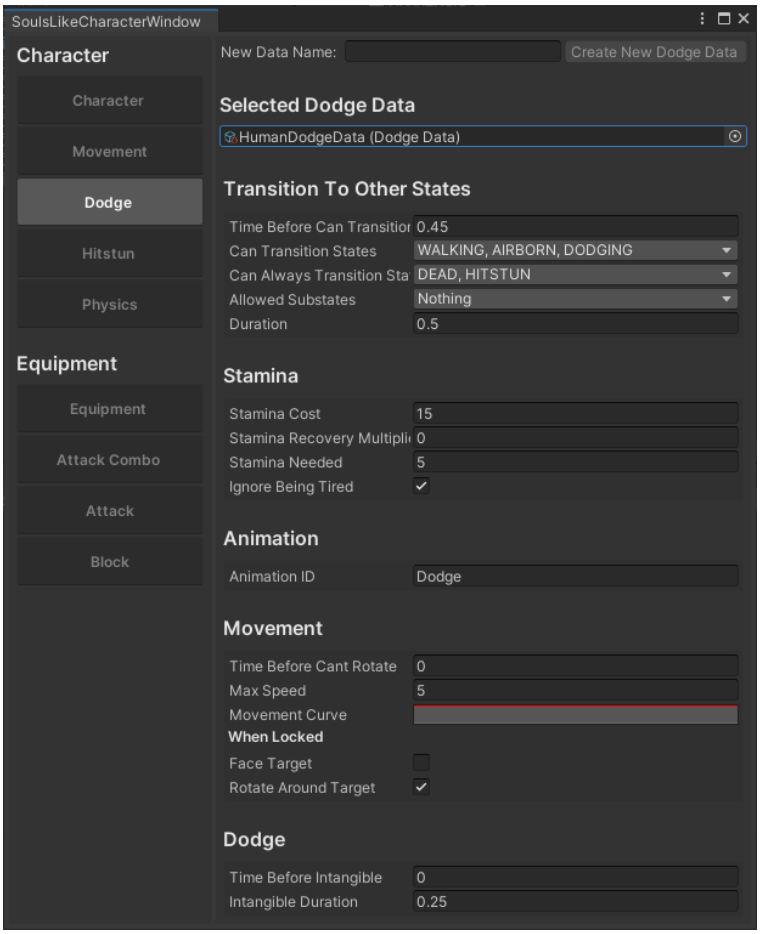

Figura 6.26. Configuració de les dades del segon estat d'evadir.

Font: Elaboració pròpia.

6.5.1.3 Atordiment

L'estat d'atordiment funciona com a interrupció d'altres estats i dintre d'ell no es pot fer res, per la qual cosa només consta de la duració i l'animació a efectuar. En aquest estat s'ha fet que es pugui efectuar una transició a un atac o esquiva abans d'acabar la duració de l'atordiment per no boquejar tant al jugador en aquest estat, però sí que es vol que per aconseguir-ho hagi d'efectuar una acció diferent de la de moviment. La configuració d'aquest estat és:

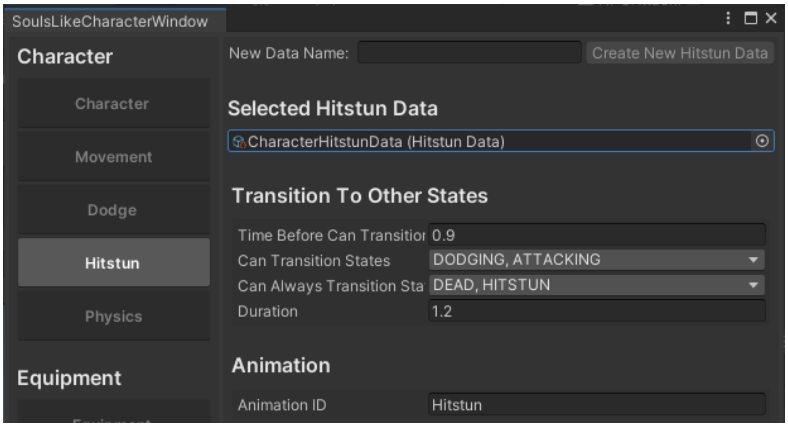

Figura 6.27. Configuració de les dades de l'estat d'atordiment.

Font: Elaboració pròpia.

6.5.1.4 Físiques

Les físiques del personatge tenen una gravetat diferent en el terra i en l'aire, ja que en el terra en mantenir la velocitat vertical fixa la gravetat funciona pel temps en el qual no es toca terra, però encara no s'ha començat a caure.

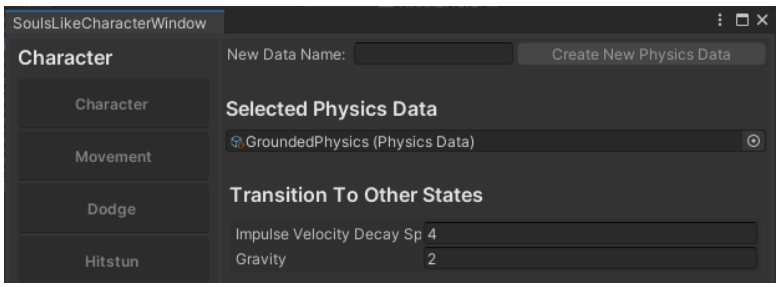

Figura 6.28. Configuració de les dades de l'estat de físiques en el terra.

Font: Elaboració pròpia.

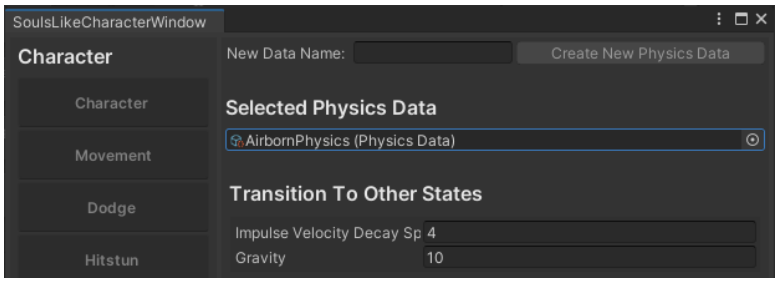

Figura 6.29. Configuració de les dades de l'estat de físiques en l'aire.

Font: Elaboració pròpia.

6.5.1.5 Equipament

Per les armes del personatge s'ha modelat amb *ProBuilder* una espasa i un escut per servir com a equipament del personatge.

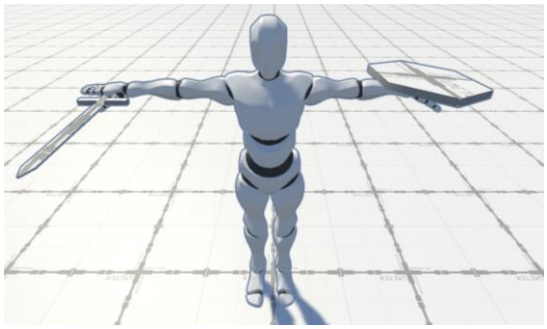

Figura 6.30. Personatge de l'escena amb l'espasa i l'escut. Font: Elaboració pròpia.

Per l'espasa s'ha fet una combinació d'atacs i s'ha assignat com a acció principal. Per poder veure dintre de l'escena la mida de les col·lisions de l'arma i quan poden fer dany s'ha creat un script de tipus *MonoBehaviour* el qual mostra la caixa de la col·lisió en gris i quan pot fer mal en vermell, aquesta opció es pot activar en qualsevol moment en la configuració de l'equipament. La configuració de l'espasa és:

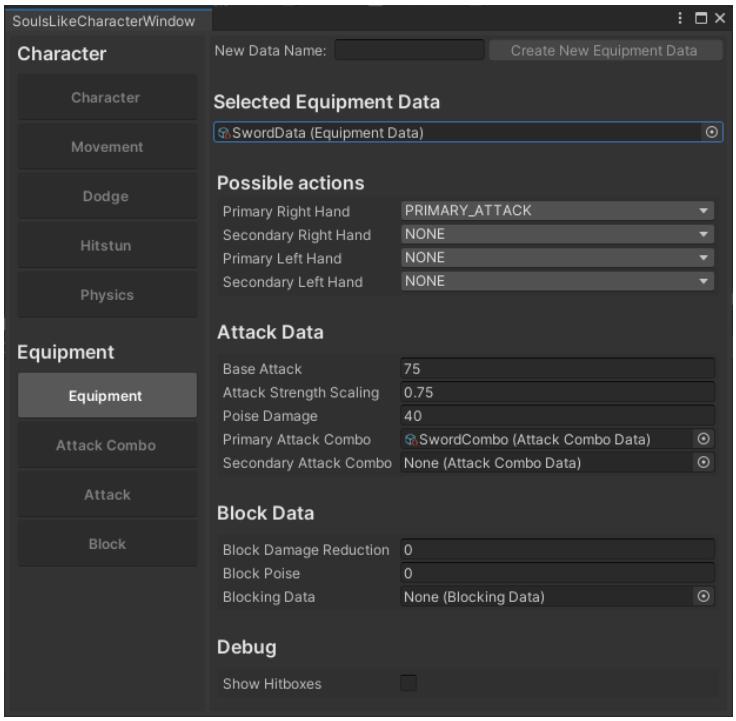

Figura 6.31. Configuració de les dades de l'espasa. Font: Elaboració pròpia.

Per l'escut s'ha fet que l'acció principal en la mà esquerra sigui la de bloquejar i l'acció de bloqueig és la de l'estat de bloquejar que s'explica en l'apartat de bloqueig.

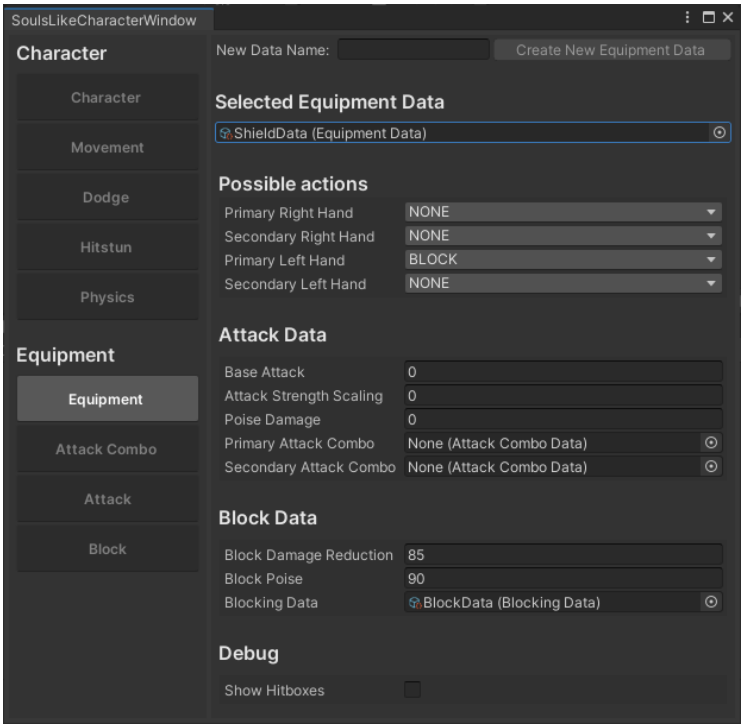

Figura 6.32. Configuració de les dades de l'escut. Font: Elaboració pròpia.

6.5.1.6 Combinació d'atacs

Per la combinació d'atacs del personatge s'ha escollit fer dos atacs, els quals un cop es facin els dos es torna al primer. Els atacs d'aquesta combinació s'expliquen a continuació.

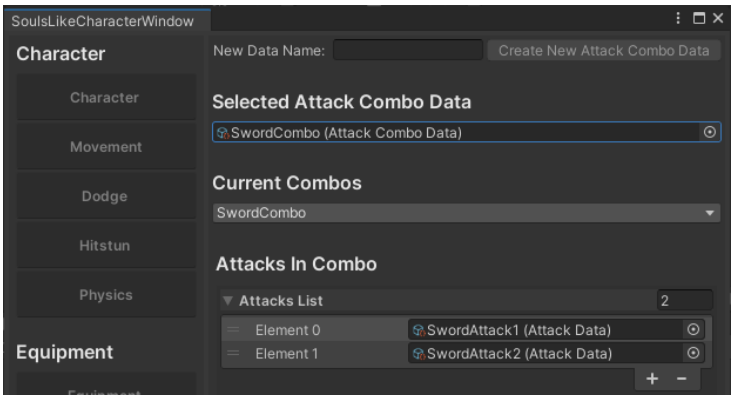

Figura 6.33. Configuració de la combinació d'atacs de l'espasa.
Font: Elaboració pròpia.

6.5.1.7 Atacs

Pels de la combinació d'atacs de l'espasa s'han creat dues dades d'atac les quals tenen els mateixos valors exceptuant el nom de l'animació i el moviment. L'animació escollida és una animació d'atac amb tres atacs diferents, la qual s'ha tallat per mostrar el primer atac i el segon en animacions diferents. En cada animació d'atac s'ha fet servir *AnimationEvents* per definir en quin punt de l'animació s'activa i es desactiva el dany de l'arma i també quan s'activa i es desactiva l'augment d'estabilitat durant l'atac.

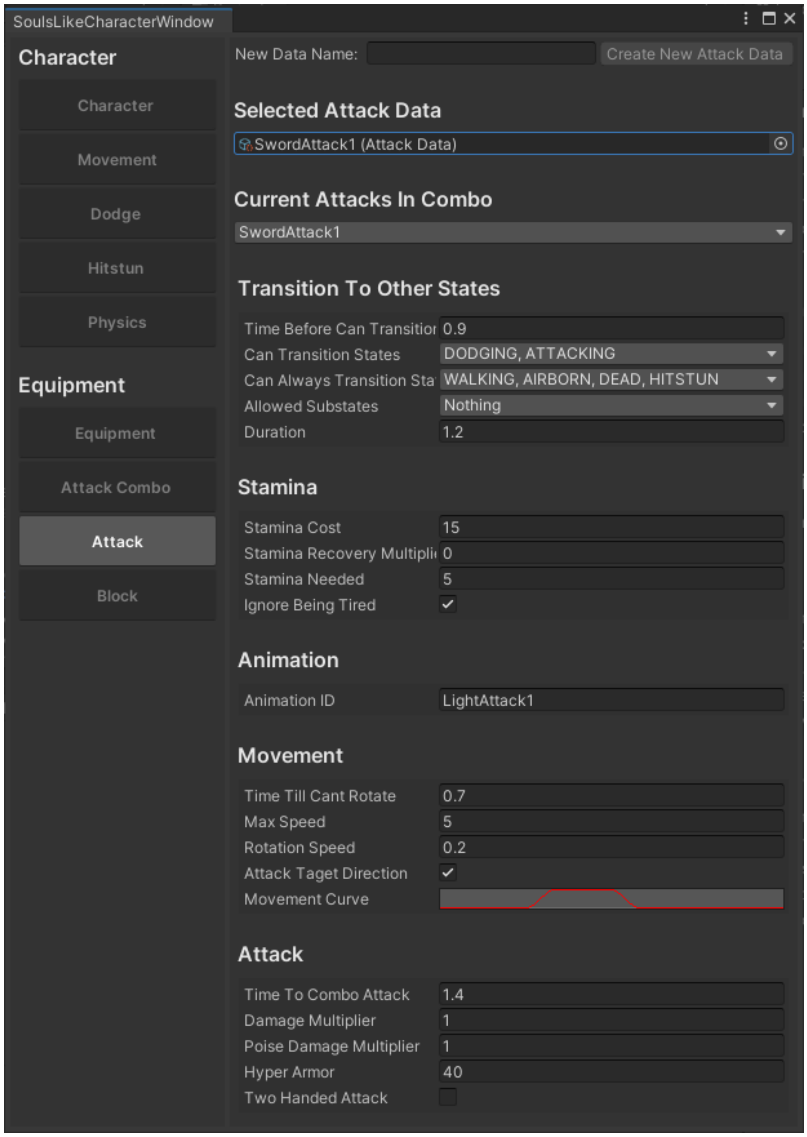

Figura 6.34. Configuració del primer atac de l'espasa. Font: Elaboració pròpia.

6.5.1.8 Bloqueig

Per poder fer l'animació de bloquejar s'ha agafat el braç i l'espatlla esquerra del personatge i s'ha posat en una *Layer* diferent de l'*Animator* per poder reproduir l'animació de bloqueig del braç sense canviar l'animació actual del personatge. La velocitat de moviment del personatge s'ha decidit no reduir-la quan es bloqueja i el temps en el qual tarda el personatge a començar a bloquejar impactes és el mateix que tarda el personatge a fer l'animació d'apujar el braç.

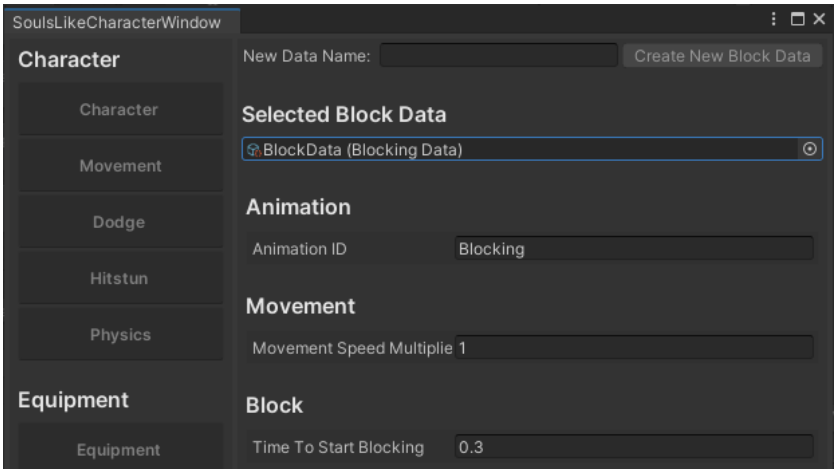

Figura 6.35. Configuració del primer atac de l'espasa. Font: Elaboració pròpia.

6.5.1.9 Interfície d'usuari

Per poder visualitzar la vida i la resistència del personatge s'han fet dos components, un amb dos *Sliders* (Unity Scripting API - Slider), un en vermell i l'altre en groc, per la vida i un slider verd per la resistència. Per cada component s'ha creat un script que controla aquests *Sliders*.

El script *HealthSystemUI* mira quan es perd vida i modifica el *Slider* vermell al de la vida actual, el *Slider* groc que està a sota del vermell s'espera un segon i es redueix gradualment fins a arribar a la vida actual. El script *StaminaSystemUI* canvia el valor del *Slider* a ser el de la resistència actual del personatge en tot moment.

Figura 6.36. Mostra de la representació visual de la vida i la resistència del personatge. Font: Elaboració pròpia.

S'ha creat un script *MonoBehaviour* nomenat *UILockedElementsManager* el qual gestiona els elements que ha de mostrar un *LockableTarget* quan està fixat per la càmera. Aquests elements són la vida de l'enemic i un punt blanc per tenir una millor visibilitat de l'enemic fixat.

Figura 6.37. Personatge amb barra de vida i un punt blanc. Font: Elaboració pròpia.

6.5.2 Creació de nous personatges

Per validar que l'eina pot funcionar per diferents tipus de personatge s'han creat 3 personatges més els quals el modelatge, animacions i controlador d'animacions ha sigut proporcionat per Montes (2023).

Per poder treballar conjuntament s'ha creat un repositori de GitHub amb el projecte de *Unity*. Montes ha clonat l'*AnimatorController* del personatge humà creat i ha canviat les animacions per les dels nous personatges.

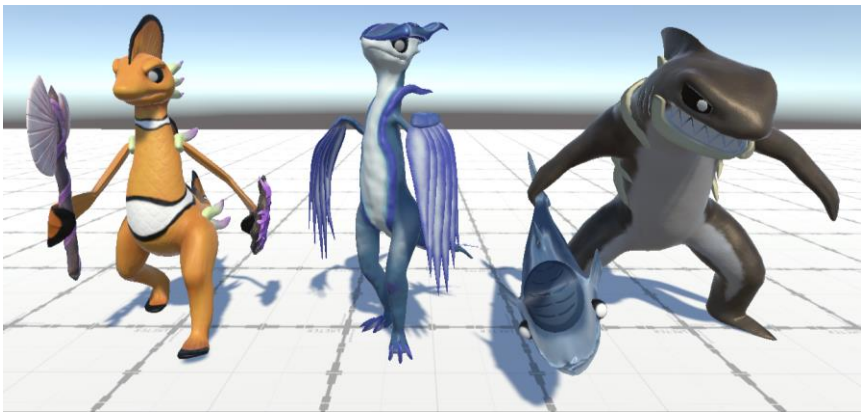

Figura 6.38. Captura dels tres nous personatges en l'escena de *Unity*.
Font: Montes (2023).

Un cop Montes ha pujat al GitHub els personatges amb les animacions s'ha procedit a crear les dades per cada personatge. Cada personatge consta de dues combinacions d'atacs, una igual que la del personatge humà i l'altre és una combinació d'un sol atac, aquests atacs són més forts i lents que els altres.

El primer personatge de tots ha sigut el peix pallasso. S'ha creat amb dades similars a la del personatge humà, exceptuant que el peix treu més vida a canvi de tenir-ne menys.

El segon personatge ha sigut el dragonet blau. Aquest personatge és molt més àgil que el personatge humà, per la qual cosa té menys vida, estabilitat i atac a canvi que els seus estats són més curts, els moviments més ràpids i consumeixen poca resistència.

El tercer personatge és el tauró tigre. Aquest personatge s'ha creat com a contraposició del dragonet blau, per la qual cosa és un personatge molt lent que consumeix molta resistència, però té molta vida, molta estabilitat i un gran dany amb els seus atacs.

6.5.3 Creació d'un combatent no controlat pel jugador

Es coneix com a personatge no jugable, NPC en anglès, tot personatge que no controla el jugador. Per poder comprovar que el funcionament de l'eina no funciona només a nivell teòric sinó que també en l'àmbit pràctic s'ha creat un comportament per NPCs. El comportament d'un NPC avalua la situació i efectua els mateixos inputs que podria fer un jugador si se li donés control del personatge que està controlant el NPC, per la qual cosa es regeix per les mateixes normes que un personatge jugable pel jugador, la diferència entre personatges resideix en els paràmetres establerts.

6.5.3.1 Estils de combat

Els NPCs s'han dissenyat per tenir tres estils de comportament depenent d'una fòrmula que s'ha nomenat factor d'agressivitat. Aquest factor és tres vegades el percentatge de la vida actual del NPC més el percentatge de la resistència del NPC i aquesta suma dividida entre quatre. Si el factor d'agressivitat supera o és inferior als

valors designats en el *ScriptableObject* de dades de comportaments de NPC llavors s'efectua un canvi d'estil de combat.

Els estils de combat del NPCs poden ser: agressiu, neutral i cautelós. Quan un NPC està en estat agressiu se centra principalment a atacar retallar distància amb l'objectiu, gairebé tots els moviments són cap endavant, fins i tot les evasions. En estat neutral el NPC juga de manera més observadora, apropant-se de tant en tant, però sent propens a fer moviments més laterals. En estat cautelós es prioritza l'evasió i generar distància amb el contrincant.

Els NPCs tenen sis estats de comportament depenent de la distància o l'estat de personatge del contrincant:

Estat de vagabundejant: En aquest estat el NPC espera al fet que hi hagi un objectiu el qual considera un enemic a una certa distància d'ell, quan detecta un objectiu passa directament a l'estat de persecució.

Estat de persecució: En estat de persecució l'objectiu del NPC és apropar-se al seu contrincant. Si l'estil de combat és agressiu, el NPC corre cap al contrincant, en cas de l'estil normal només s'aproxima caminant i en l'estil cautelós s'aproxima mentre es mou també cap als costats. En certa distància contra el contrincant el NPC passarà a l'estat d'atac i en cas que la distància contra el contrincant sigui major que un valor establert es passarà a l'estat d'observació.

Estat d'observació: En estat d'observació el NPC busca fer un moviment més lateral per regenerar resistència. En estil de combat agressiu el moviment lateral es mou també una mica cap endavant, en l'estil de combat neutral el moviment és lateral i en estil cautelós el moviment va una mica cap enrere.

Estat d'atac: En estat d'atac el NPC espera a acabar l'atac efectuat per avaluar si tornar a atacar o tornar a persecució depenent de la distància, en cas d'estar en estil cautelós també hi ha possibilitats després de cada atac, si l'enemic està prou a prop, que l'NPC rodi per evitar un possible atac del contrincant.

Estat d'esquiva: En l'estat d'esquiva el personatge selecciona una direcció aleatòria, dintre d'un rang, depenent del seu estil de combat, en estil agressiu l'esquiva és cap

endavant, en estil neutral l'esquiva és lateral i en estil cautelós l'esquiva és cap enrere.

Estat d'atordiment: En aquest estat el NPC entra quan està atordit, un cop acaba l'atordiment si l'enemic encara està atacant s'ha establert una possibilitat, a modificar en les dades de l'estat, d'intentar evadir l'atac, en cas contrari el NPC torna a l'estat de persecució.

Per la modificació d'aquests estats s'ha creat una altre finestra similar a la de creació de personatges, però per canviar els paràmetres de comportament dels NPCs.

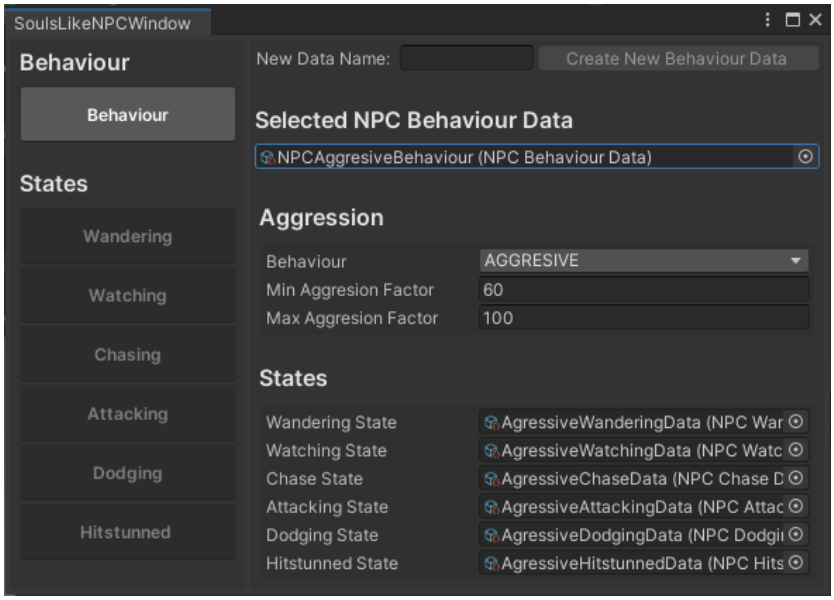

Figura 6.39. Captura de la finestra de modificació de variables dels NPCs.
Font: Elaboració pròpia.

Com el fixat de contrincants dels personatges funcionen agafant un possible objectiu a atacar es pot arribar a fer que dos NPCs o lluitin mútuament.

6.6 Cas d'ús

Serpent Blade és un videojoc creat per Alba, Gómez, Llovera, Montes i Raya el 2022. Aquest videojoc es va desenvolupar en tres dies per la GameJam "Mini Jam 113: Final Boss" (ZahranW, KingW, 2022). Aquest videojoc és un "bullet hell" en vista de càmera isomètrica on l'objectiu de la protagonista és derrotar a l'enemic del

centre de l'escenari tres cops, cada cop que l'enemic és derrotat el següent combat és més difícil.

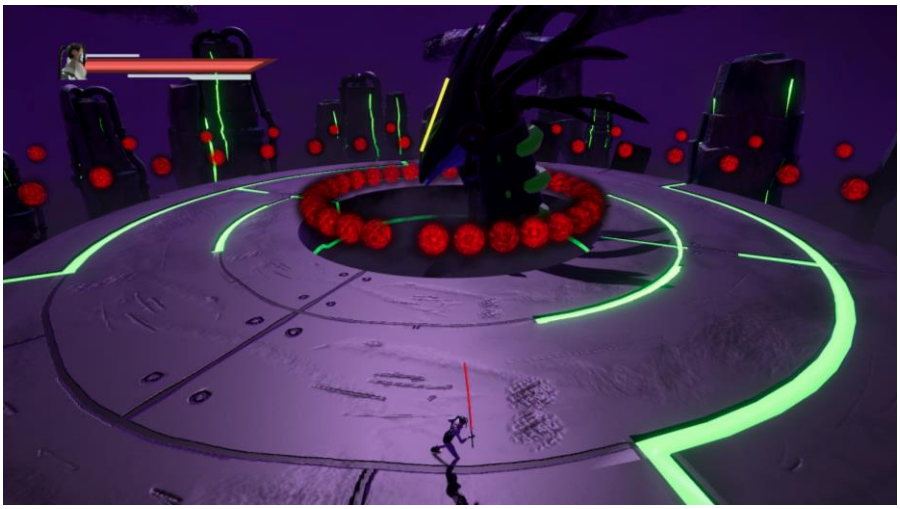

Figura 6.40. Captura del joc *Serpent Blade*. Font: Elaboració pròpia.

Les mecàniques que pot dur a terme la protagonista són:

- Moviment: Un moviment no gradual en la direcció de l'input. El moviment lateral es fa de forma circular donant així la volta a l'enemic del centre de l'escenari.
- Atacar: Pot atacar en moviment fins a tres vegades seguides, amb el tercer atac fent més dany que els dos anteriors.
- Evadir: L'esquiva és ràpida en línia recta l'input del jugador.
- Bloquejar: Fa una esfera la qual bloqueja atacs de l'enemic i retorna projectils.

La càmera segueix a la protagonista i canvia una mica d'alçada depenent de la distància del personatge al centre per poder mantenir un millor enquadrament de l'enemic.

Per adaptar aquest videojoc a l'eina el primer de tot que s'ha fet ha sigut treure la càmera existent i canviar-la pel prefab de la càmera feta en l'eina. Per poder veure a l'enemic i a la protagonista s'han afegit a la *Culling Mask* de la càmera les layers corresponents a renderitzar.

El primer inconvenient que es pot veure és el fet que la càmera està massa a prop de la protagonista relatiu a la seva mida, i com l'enemic és molt gran pot haver problemes de visibilitat.

Figura 6.41. Captura de l'escena amb la nova càmera. Font: Elaboració pròpia.

Per poder solucionar el problema s'ha allunyat un 50% la distància de la càmera per poder tenir millor visibilitat de l'entorn.

Figura 6.42. Captura de l'escena amb la distància de la càmera modificada. Font: Elaboració pròpia.

Per poder fixar a l'enemic del centre de l'escenari se li ha posat el script de *LockableTarget* a l'enemic i la referència d'on ha d'apuntar la càmera (*CameraRoot*) a l'os del cap. D'aquesta manera quan l'enemic fa un atac la càmera segueix al seu punt més important. Per poder mostrar el punt de fixat d'objectiu s'ha posat el script de *UILockedElementsManager* en el canvas de l'escena. No s'ha posat en fixar

objectiu que l'enemic mostri una barra de vida, ja que la té incorporada de manera diegètica en el cap.

Un cop preparada la càmera s'ha afegit a l'escena el prefab del personatge humà, del qual s'ha substituït la part visual del personatge per la de la protagonista del joc. Primer de tot s'ha modificat el *AnimatorController* per tenir una estructuració similar a la dels personatges ja creats i també s'ha afegit un *Blend Tree* en l'animació de moviment. Com aquest personatge té animació de mort el *RagdollActivator* s'ha tret de la protagonista.

Al *GameObject* del personatge el qual conté les animacions se li ha posat el script d'*AttacksHurtboxController* per poder controlar les col·lisions dels atacs en les animacions. S'ha posat el script de *HurtboxCollider* a l'arma de la protagonista per poder fer dany a l'enemic. Un cop afegits aquests scripts s'han fet els *AnimationEvents* per poder activar les col·lisions de l'atac, com les animacions són curtes i el que es fa és mantenir l'últim fotograma abans de tornar a l'animació de caminar no s'ha afegit un *AnimationEvent* per desactivar la col·lisió de l'atac, d'aquesta manera es desactiva a l'acabar l'estat d'atac i no per *event* de l'*animator*.

Perquè els atacs del jugador puguin fer mal a l'enemic s'ha modificat codi del *HurtboxCollider* perquè agafi el script gestor de vida de l'enemic i li faci mal.

Per fer que l'enemic pugui atacar al jugador s'ha canviat una línia de codi la qual buscava a la protagonista a través del component de controlador de personatge el qual ja no té. També s'han efectuat canvis en els atacs de l'enemic perquè cridin la funció de rebre dany del *HealthSystem*.

En l'estat d'esquiva s'ha afegit en l'*OnEnter* l'activació de l'efecte d'estela de l'esquiva.

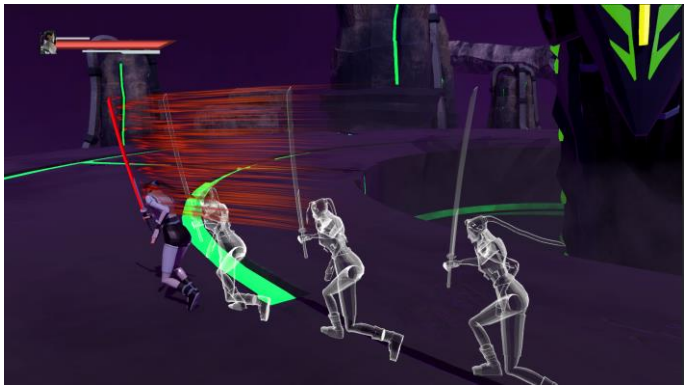

Figura 6.43. Efecte d'estela de l'esquiva del personatge. Font: Elaboració pròpia.

A diferència d'en el videojoc original en aquest s'ha fet que en mantenir el botó d'evadir el personatge vagi més de pressa. Aquest canvi s'ha fet perquè d'aquesta manera el moviment sense córrer del personatge s'ha pogut disminuir una mica, ja que resulta més còmode donat el nou estil de controlador.

Pel bloqueig s'ha fet que en l'*OnEnter* es cridi a la funció que activa l'escut per poder bloquejar i reflectir atacs. El bloqueig es pot efectuar en els estats de moviment, atacar i evadir. Quan es bloqueja s'és invulnerable durant el temps que dura l'escut activat, el qual és una duració fixa.

En l'*OnEnter* dels estats d'atac, bloqueig i esquiva s'ha cridat la funció de reproduir el so corresponent que tenien posat en el controlador anterior. En la càmera s'ha afegit un *FMOD Studio Listener* (FMOD Documentation - Game Components) per poder senitr el so.

Pel funcionament de la barra de vida de la interfície d'usuari s'ha afegit al canvas el script *HealthSystemUI* i *HealthBar* amb els *Sliders* de vida serialitzats.

Quan el personatge mor es crida la corutina la qual posa la pantalla en negre i reinicia l'escena per tornar a començar el combat.

Amb aquests canvis ja es pot jugar el joc de principi a fi amb el protagonista amb les mecàniques del controlador creat en aquest treball i es pot modificar tots els valors del personatge amb la finestra de l'eina.

7. Conclusions

En aquest apartat s’ha valorat els resultats aconseguits en referència als objectius marcats i a continuació s'expliquen les línies de futur per tal de poder millorar i sobrepassar les limitacions actuals del treball desenvolupat.

7.1 Valoració dels resultats

Aquest treball ha aconseguit crear un controlador amb mecàniques de combat Souls-like, així com una eina per a la creació i modificació d'aquests controladors. La modificació i creació de nous personatges es pot dur a terme utilitzant la finestra desenvolupada de l'eina.

L'anàlisi de diferents videojocs del gènere Souls-like ha servit per extreure el funcionament de la càmera, les mecàniques de combat i el game feel del gènere. Per les mecàniques de cada joc s'ha elaborat un llistat on s'han posat en comú i s'han agrupat per funcionament lògic totes aquelles mecàniques rellevants pel funcionament d'un combat. Amb la llista fina s'ha fet una definició del funcionament més comú de les mecàniques del combat del gènere Souls-like.

S'ha escrit els requisits del controlador i els paràmetres mínims necessaris que un dissenyador necessita per poder modificar les diferents mecàniques d'un personatge creat amb l'eina. Per aquests requisits i paràmetres s'han dividit els sistemes i les possibles mecàniques que pot fer el controlador en seccions a fi d'explicar el funcionament esperat de cada element definit.

Amb els requisits s’han creat els scripts amb els quals s'ha aconseguit programat els sistemes i les mecàniques dissenyades. Aquest apartat ha sigut el més llarg donat que el controlador del personatge ha de permetre múltiples sistemes funcionant simultàniament sense errors.

S’ha fet un estudi de com crear un inspector i una finestra personalitzada en *Unity*. Gràcies al disseny provisional fet en *Figma* s'ha implementat de manera ràpida la finestra personalitzada la qual aporta una millor organització de la informació d'un personatge i un millor flux de treball.

Per la validació del controlador exposada en el cinquè objectiu s'ha afegit tres tipus de personatges extra a part del fet servir pel desenvolupament del controlador els quals tenen característiques diferents, aquests personatges s'han pogut implementar gràcies als models i animacions fetes per Montes (2023). Per combatre contra un enemic s'ha dissenyat i programat un comportament computacional el qual depenent de l'estat del personatge té un estil de combat o un altre. Aquests personatges no controlats pel jugador es poden modificar a través d'una altra finestra personalitzada. Per portar l'eina a un escenari real s'ha agafat el projecte Serpent Blade i s'ha aconseguit aplicar l'eina sense complicacions.

Finalment, s'han creat vídeos explicatius per la implementació i l'ús correcte de l'eina amb la finalitat d'aconseguir que una persona, sense requerir nocions de programació, entengui com poder usar els controladors ja creats i com crear-ne de nous.

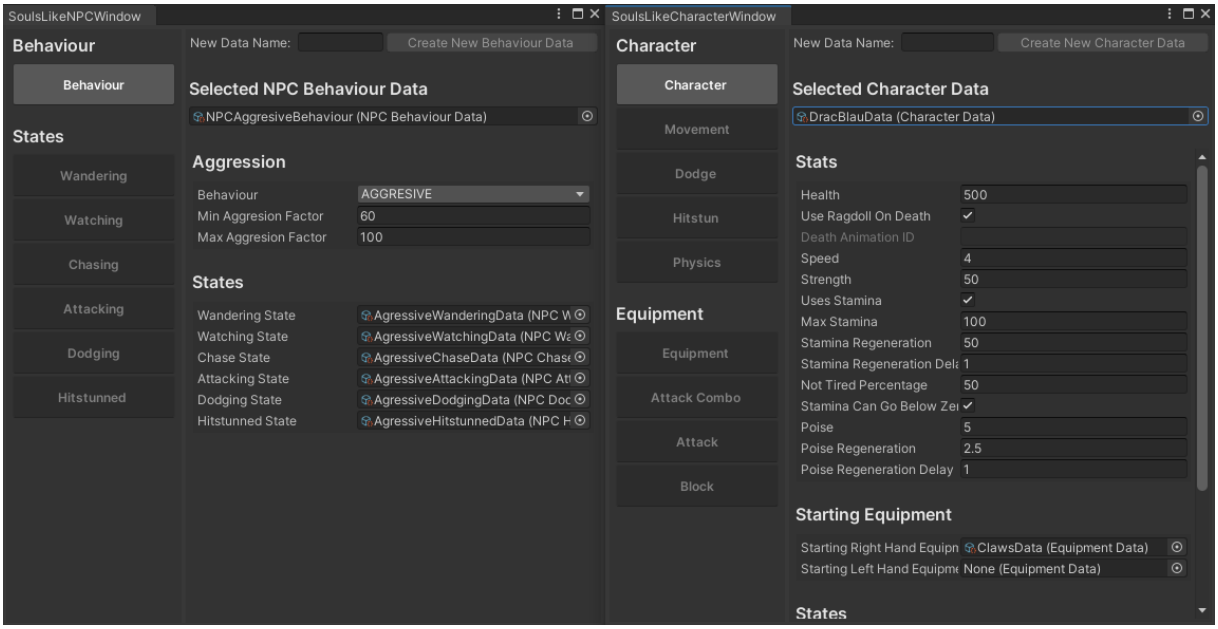

Figura 7.1. Resultat de les finestres desenvolupades en l'eina.

Font: Elaboració pròpia.

L'eina obtinguda en aquest treball s'ha nomenat com a "Souls-like Character Controller" tenint en compte els noms que han fet servir els referents del treball. Amb l'eina acabada s'ha fet una taula comparativa de l'eina desenvolupada amb les característiques de les eines agafades com a referents.

Característiques	Starter Assets - Third Person Character Controller	3rd Person Controller + Fly Mode	Character Controller SUPER	Souls-like Character Controller
Documentació d'ús				
Escena demostrativa				
Scripts diferents per cada mecànica				
Ús d'estats en el controlador				
Inspector personalitzat				
Modificació d'elements de Game Feel				
Suport de mecàniques de combat				
Ús de finestra personalitzada				
Script de gestió d'inputs				

Taula 5: Taula comparativa de característiques de les eines referents i l'eina desenvolupada. Font: Elaboració pròpia.

7.2 Línies de futur

Una millora de l'eina és que en l'estat actual del controlador es pot combatre amb aliats i enemics, però no té suport total en combat contra múltiples objectius tant pel jugador com pels personatges no jugables. Quan un objectiu mor els NPCs passen a atacar un altre, però no en mig d'un combat. El jugador només pot començar a fixar objectiu a l'enemic més proper i un cop fixat no es pot canviar amb l'input de la càmera com es pot fer en els Souls-like.

Un altra millora de l'eina seria la possibilitat de poder agafar els *GameObjects* de l'escena i modificar-los directament en la finestra de l'editor, d'aquesta manera es podrien modificar alguns elements que només es poden modificar d'un objecte en escena en la mateixa finestra.

En un futur es pot desplegar l'eina en l'Asset Store de *Unity* (*Unity Technologies*). En la documentació de *Unity* (Manual – Asset Store Publishing) es llista que per poder fer aquest desplegament es requereix:

1. Crear un compte de publisher.
2. Crear un esborrany de package.
3. Pujar els assets al package
4. Omplir els detalls del package.
5. Enviar el package per aprovació.

Encara que hi hagi molts aspectes en el que el controlador pot millorar el resultat ha sigut satisfactori i s'han assolit més objectius dels previstos inicialment en començar aquest treball.

8. Referències

8.1 Bibliografia

Adams E. (2014) *Fundamentals of Game Design.* doi: https://dl.acm.org/doi/10.5555/2544002

Alamia, M. (2013). Article - world, view and projection transformation matrices. Coding Labs. Recuperat el 12 de gener de 2023 de: http://www.codinglabs.net/article_world_view_projection_matrix.aspx

Autodesk. (2023). *3ds Max 2024 Help* | *Clipping Planes*. Recuperat el 6 de Maig https://help.autodesk.com/view/3DSMAX/2024/ENU/?guid=GUID-C2E55F9A-4F78-4D48-B5C5-1127AA5FBF2C

Bocanegra J. (2011). Countdown To Resident Evil's 15th Anniversary (Part 1). Recuperat el 18 de gener de 2023 de: https://www.relyonhorror.com/articles/countdown-to-resident-evils-15th-anniversary-part-1/

Card, S., Moran, T. i Newell, A. (1986). *The model human processor: An engineering model of human performance*. doi: https://doi.org/10.1177/107118138102500180

Clement J. (2023). Elden Ring cumulative units sold worldwide 2023. Recuperat el 12 de gener de 2023 de: https://www.statista.com/statistics/1300663/elden-ring-sales-worldwide/

Crawford C. (2003). *Chris Crawford on Game Design*. (1st Edition). New Riders Pub.

Dodd A. (23 de març 2013). *[Horror Declassified] An Examination Of Tank* Controls. Recuperat el 14 de gener de 2023 de: https://bloody-disgusting.com/news/3224958/horror-declassified-an-examination-of-tank-controls/

Firelight Technologies Pty, Ltd. (1995). *Game Components*. Recuperat el 2 de Juny de 2023 https://www.fmod.com/docs/2.01/unity/game-components.html

Gorisse, G., Christmann, O., Amato, E. A., i Richir, S. (2017). *First- and third-person perspectives in immersive virtual environments: Presence and performance analysis of Embodied Users*. Frontiers in Robotics and AI, 4. https://doi.org/10.3389/frobt.2017.00033

Graves, A. (2019). *Character Controller SUPER*. Unity Asset Store. Recuperat el 16 de gener de 2023 de: https://assetstore.unity.com/packages/templates/systems/3rd-person-controller-fly-mode-28647

Guzsvinecz, T. (2022). *The correlation between positive reviews, playtime, design and game mechanics in souls-like role-playing video games*. Multimedia Tools and Applications, 82(3), 4641–4670. doi: https://doi.org/10.1007/s11042-022-12308-1

Hoory, L. (2022). *What is waterfall methodology? here's how it can help your project management strategy*. Forbes. Recuperat el 17 de gener de 2023 de: https://www.forbes.com/advisor/business/what-is-waterfall-methodology/

Jordan J. (2021). *What's the First 3D Game in the World?* Recuperat el 14 de gener de 2023 de: https://narrasoft.com/what-is-the-first-3d-game-in-the-world/

Ketonen M. (2016). *Designing a 2D fighting game*. Recuperat el 5 de març de 2023 de: https://www.theseus.fi/bitstream/handle/10024/118514/Thesis_Miikka_Ketonen_KAT13PT.pdf?sequence=1&isAllowed=y

Kucic, M. (2005). *How to Prototype a Game in Under 7 Days*. Recuperat el 16 de gener de 2023 de: https://www.gamedeveloper.com/disciplines/how-to-prototype-a-game-in-under-7-days

Lutkevich, B., & Lewis, S. (2022). *What is the waterfall model? - definition and guide. Software Quality*. Recuperat el 17 de gener de 2023 de: https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model

Marques, V. (2015). *3rd person controller + fly mode. Unity Asset Store*. Recuperat el 16 de gener de 2023 de: https://assetstore.unity.com/packages/templates/systems/3rd-person-controller-fly-mode-28647

Microsoft Corporation. (15 de setembre, 2021). *Capitalization Conventions*. Recuperat el 2 de Juny de 2023 https://learn.microsoft.com/en-us/dotnet/standard/design-guidelines/capitalization-conventions

Microsoft Corporation. (8 d'abril de 2023). *Enumeration types (C# reference).* Recuperat el 2 de Juny de 2023 https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

Microsoft Corporation. (s.d). *FlagsAttribute Class*. Recuperat el 2 de Juny de 2023 https://learn.microsoft.com/en-us/dotnet/api/system.flagsattribute?view=net-7.0

Montes M. (2023). *Disseny i animació de personatges basats en animals marins pel gènere Souls-like: Memòria.* (Treball de fi de Grau no publicat). TecnoCampus Mataró.

Naftis, M., Tsatiris, G., i Karpouzis, K. (2021). *How Camera Placement Affects Gameplay in Video Games*. doi https://doi.org/10.48550/arXiv.2109.03750

NASA. (2022). *Aircraft rotations - glenn research center*. NASA. Recuperat de: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/aircraft-rotations/

Parente, D. (2014). *Los 3 Cs del Diseño de videojuegos*. Daniel Parente Blog. Recuperat el 14 de gener de 2023 de: https://www.danielparente.net/es/2014/05/27/los-3-cs-del-diseno-de-videojuegos/

Rouse, A. (2020). *Programming Tool*. Recuperat el 17 de gener de 2023 de: https://www.techopedia.com/definition/8996/programming-tool

Rouse, R. (1999). *What's your perspective?* ACM SIGGRAPH Computer Graphics, 33(3), 9–12. doi: https://doi.org/10.1145/330572.330575

Rutter, J., i Bryce, J. (2006). *Understanding Digital Games*. doi: https://doi.org/10.4135/9781446211397

Sanglard F. (2013). *DOOM ENGINE CODE REVIEW*. Recuperat el 18 de gener de 2023 de: https://fabiensanglard.net/doomIphone/doomClassicRenderer.php

Swink, S. (2007). *Game Feel: The Secret Ingredient. Game Developer*. Recuperat el 10 de gener de 2023 de: https://www.gamedeveloper.com/design/game-feel-the-secret-ingredient

Swink, S. (2008). *Game Feel: A Game Designer's Guide to Virtual Sensation*. (1st Edition). CRC Press.

Tadeusz Stach, T.C. Graham N., Brehmer M., Hollatz A. (n.d.). *Classifying input for active games*. Recuperat el 12 de gener de 2023 de: https://mattbrehmer.ca/pubs/stach_ace09.pdf

Tyler D. (2023). *Video Game Mechanics for Beginners*. Recuperat el 17 de gener de 2023 de: https://www.gamedesigning.org/learn/basic-game-mechanics/

Unity Technologies,. (14 d'abril de 2023). *Unity 2021.3.23*. Unity. Recuperat el 2 de juny de 2023 https://unity.com/releases/editor/whats-new/2021.3.23

Unity Technologies,. (s.d). *Manual - AnimatorController*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity Technologies,. (s.d). *Manual – Input System*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/Packages/com.unity.inputsystem@1.6/manual/index.html

Unity Technologies,. (s.d). *Manual - ProBuilder*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/Packages/com.unity.probuilder@4.0/manual/index.html

Unity Technologies,. (s.d). *Manual - RagdollWizard*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/Manual/wizard-RagdollWizard.html

Unity Technologies,. (s.d). *Scripting API – CharacterController*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/CharacterController.html

Unity Technologies,. (2023). *Scripting API – FindProperty*. Unity. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/SerializedObject.FindProperty.html

Unity Technologies,. (2023). *Scripting API – GetComponent*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html

Unity Technologies,. (2023). *Scripting API – MonoBehaviour*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Unity Technologies,. (2023). *Scripting API – serializedObject*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/Editor-serializedObject.html

Unity Technologies,. (2023). *Scripting API – SerializedProperty*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/ScriptReference/SerializedProperty.html

Unity Technologies,. (2023). *Scripting API – Slider*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Slider.html

Unity Technologies,. (2023). *Asset packages. Unity*. Recuperat el 2 de juny de 2023 https://docs.unity3d.com/560/Documentation/Manual/AssetPackages.html

Unity Technologies,. (2023). *Manual - Cinemachine*. Unity. Recuperat el 2 de juny de 2023

https://docs.unity3d.com/Packages/com.unity.cinemachine@2.9/manual/index.html

Unity Technologies,. (2023). *Manual - Input Manager*. Recuperat el 2 de Juny de 2023 https://docs.unity3d.com/Manual/class-InputManager.html

Unity Technologies,. (2023) *Starter assets - third person character Controller*. Unity Asset Store. Recuperat el 2 de juny de 2023: https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526

Unity Technologies. *Unity Documentation Transform - Scripting API*. Recuperat el 2 de juny de 2023 de: https://docs.unity3d.com/ScriptReference/Transform.html

Wood, A. (23 de febrer de 2022). *What the hell is a souls-like? game devs break down FromSoftware's accidental genre*. gamesradar. Recuperat el 4 de Febrer de: https://www.gamesradar.com/what-is-a-souls-like-developers-explain/

ZahranW, KingW. (agost de 2022). *Mini Jam 113: Final Boss*. Recuperat de: https://itch.io/jam/mini-jam-113-final-boss

8.2 Ludografía

Alba J., Gómez S., Llovera G., Montes M. i Raya A. (2022) *Serpent Blade* (PC) [Videojoc]. Recuperat el 4 de Juny de 2023: https://andrew-raya.itch.io/serpent-blade

Argonaut Games, (1986). *Starglider* (Atari ST) [Videojoc]. London, England: Telecomsoft.

Argonaut Games, (1988). *Starglider II* (Atari ST) [Videojoc]. London, England: Telecomsoft.

Behaviour Interactive, (2016). *Dead By Daylight* (PC) [Videojoc]. Montreal, Canada: Behaviour Interactive.

Blizzard Entertainment, (2012). *Diablo III* (PC) [Videojoc]. Irvine, CA: Blizzard Entertainment.

Capcom, (1994). *Street Fighter 2* (PC) [Videojoc]. Osaka, Japan: Capcom

Capcom, (1996). *Resident Evil* (PC) [Videojoc]. Osaka, Japan: Capcom

FromSoftware Inc., (2011). *Dark Souls* (PC) [Videojoc]. Tokyo, Japan: FromSoftware Inc.

FromSoftware Inc., (2014). *Dark Souls II* (PC) [Videojoc]. Tokyo, Japan: FromSoftware Inc.

FromSoftware Inc., (2016). *Dark Souls III* (PC) [Videojoc]. Tokyo, Japan: FromSoftware Inc.

FromSoftware Inc., (2019). *Sekiro: Shadows Die Twice* (PC) [Videojoc]. Tokyo, Japan: FromSoftware Inc.

FromSoftware Inc., (2022). *ELDEN RING* (PC) [Videojoc]. Tokyo, Japan: FromSoftware Inc.

id Software, (1993). *DOOM* (MS-DOS) [Videojoc]. Richardson, TX: id Software.

Nintendo, (1996). *Super Mario 64* (Nintendo 64) [Videojoc]. Kyoto, Japan: Nintendo.

Respawn Entertainment, (2019). *Star Wars Jedi: Fallen Order* (PC) [Videojoc]. Sherman Oaks, California, U.S: Respawn Entertainment.

Russell S., Graetz M., Samson P. i Witaenem W., (1962). *Spacewar!* (DEC PDP-1 minicomputer) [Videojoc]. Massachusetts Institute: Steve Russell, Chris Diamond.

Technōs Japan, (1984). *Karate Champ* (Arcades) [Videojoc]. Suginami City, Tokyo, Japan: Data East.

Valve Corporation, (2012). *Counter-Strike: Global Offensive* (PC) [Videojoc]. Bellevue, WA: Valve Corporation.

8.3 Software a tercers

Adobe. (2021) *Adobe Photoshop* (2021) [Software]. Recuperat de: https://www.adobe.com/products/photoshop.html

Adobe. (2023). *Mixamo* [Software]. Recuperat el 20 d'Abril de: https://www.mixamo.com/

Figma, Inc. (2016). *Figma* (9.0) [Software]. Recuperat de: https://www.figma.com/

Microsoft Corporation (1997). *Visual Studio* (2022) [Software]. Recuperat de: https://visualstudio.microsoft.com/

JetBrains. (2017). *Rider* (2022.3.2) [Software]. Recuperat de: https://www.jetbrains.com/rider/

Unity Technologies. (2022a). *ProBuilder* (5.0.6) [Software]. Recuperat de: https://unity.com/es/features/probuilder

Unity Technologies. (2022b). *Cinemachine* (2.8.9) [Software]. Recuperat de: https://unity.com/unity/features/editor/art-and-design/cinemachine

Unity Technologies. (2023a). *Unity Engine 5* (2021.3.23) [Software]. Recuperat de: https://unity.com

Unity Technologies. (2023b). *Input System* (1.5.1) [Software]. Recuperat de: https://unity.com/features/input-system