Universitat de Girona
Escola Politècnica Superior

Grau en Disseny i Desenvolupament de Videojocs

PROJECTE FINAL DE GRAU

Creació d'un videojoc Hack'n&Slash de món obert

Autors:
Abel Navarro Rosell
Daniel Vázquez Samos
Sergio Chinchilla Córdoba

Tutor:
Gustavo Patow

MEMÒRIA

Convocatòria:
Setembre 2023

Departament:
Informàtica, Matemàtica Aplicada i Estadística

Índex

1 Introducció i objectius

1.1 Introducció

Entre la gran quantitat de videojocs *indie* que apareixen cada any, hi ha diversos gèneres que en destaquen, però el *Hack and Slash* no és un d'ells. Aquesta escassetat ve causada per la gran complexitat i precisió que requereixen els jocs d'aquest gènere. D'entre els jocs *indie* que apareixen cada any, n'hi ha pocs que siguin complexos i tècnicament profunds.

Deixant de costat els jocs *indie*, els *Hack and Slash* que apareixen segueixen tots un esquema molt convencional. La major part d'aquests tenen mapes lineals i mecàniques de millora a través de l'obtenció de noves armes. Sent així, gairebé mai existeix una mecànica de millora de l'equipament inicial del jugador.

Tenint en compte tot això, s'ha volgut crear un joc *Hack and Slash* que incorpori mecàniques de món obert, i millora d'armes, donant un gran pes al sistema d'obtenció de runes. Es donarà molta importància al disseny i programació de nivell, combat i mapa. L'objectiu del joc serà explorar el mapa, descobrir la trama a poc a poc i acabar matant l'enemic final.

Tenint tot això en compte, al final de la implementació, s'espera tenir un joc profund en les mecàniques, que requereixi exploració del món, i on el jugador hagi d'esforçar-se a comprendre el comportament dels enemics i del mateix protagonista, per a poder tenir un bon control sobre aquest.

1.2 Motivacions

La major motivació per nosaltres, actualment, seria intentar superar el repte de realitzar un joc tan complex mecànicament tenint pocs recursos i un temps bastant limitat. A més, volem tractar de refinar-ho tot el possible per tal de crear una experiència enriquidora pel jugador.

A més, com a estudiants de videojocs que gaudim de l'entreteniment que ofereix aquesta indústria, volem experimentar el procés de creació i disseny d'un videojoc, especialment en els següents punts:

- Creació d'un entorn on el jugador es senti atret per conèixer la història i el context i que no es faci repetitiu, com acostuma a passar en alguns jocs de món obert.
- Disseny d'uns enemics que puguin suposar un repte pel jugador, però sense fer-lo arribar a un nivell elevat d'estrès perquè requereixen una gran habilitat i experiència per derrotar-los.
- Comprensió del grau de complexitat que requereix un *Hack and Slash* per ser desenvolupat i que faci una sensació de fluïdesa.
- Aplicació de tots els coneixements que hem adquirit durant els anys que hem estat al grau.
- Adquirir experiència en la creació de videojocs i obtenir un producte que es pugui publicar per tal de compartir-lo de forma gratuïta amb la resta.

1.3 Objectius

El nostre objectiu principal amb aquest projecte és desenvolupar una *demo* jugable d'un videojoc de nivell comercial. Es vol crear un videojoc *Hack and Slash* de món obert, basat en mecàniques d'exploració, moviment i combat. Es vol desenvolupar un joc considerat *indie* amb característiques de profunditat i amplitud que se'ls hi atribueix als jocs AAA.

Quant al prototip del joc dissenyat, tenim els següents objectius:

- Treballar en un videojoc que contingui un sistema de combinacions de moviments pel jugador, que permeti combinar diferents atacs.
- Crear uns enemics amb una intel·ligència artificial independent que sigui capaç de prendre diferents decisions, juntament amb un enemic final que tingui diferents fases, és a dir, que el comportament canviï segons avança el combat.
- Desenvolupar un sistema de guardat i càrrega de partida.
- Establir una gestió de runes, que permeti guardar-les en un inventari i incrustar-les dins l'arma, juntament amb una millora de diferents estadístiques del jugador.
- Programar uns NPC que tinguin un diàleg propi, per tal de posar al jugador en context de la narrativa.
- Dissenyar un espai que motivi al jugador a explorar tot el mapa abans d'enfrontar-se a l'enemic final, tot i tenir l'opció de fer-ho des del principi.

Com a extra, es vol aprofundir i millorar l'ús i els coneixements sobre Unreal Engine 4 (UE4) i el llenguatge de Blueprints, juntament amb el servei de control de versions anomenat Github. Per al bon funcionament d'aquest projecte és important també treballar la gestió d'un equip petit i la planificació per tal de desenvolupar de forma individual, però connexa i complementària.

1.4 Distribució de Tasques

L'equip de treball està format per tres integrants, tots estudiants del Grau en Disseny i Desenvolupament de Videojocs a la UdG. Els integrants són:

- Chinchilla Córdoba, Sergio.
- Navarro Rosell, Abel.
- Vázquez Samos, Daniel.

Dels percentatges que es veuen a continuació, tots els integrants tenen un 15% a les mecàniques ja que el disseny del videojoc s'ha realitzat de forma conjunta a través de reunions i *brainstorming*. La resta de percentatges s'expliquen individualment per cada cas.

El Sergio va aportar una part a l'estètica, especialment en la presa de decisions, i en la narrativa, la qual es va decidir inicialment amb el disseny del videojoc. La part on ha dedicat més temps és la tecnologia, per això té més pes. Quant a la tecnologia, ha desenvolupat el sistema de compatibilitat entre controls de teclat i ratolí i controlador, la intel·ligència artificial, l'inventari, el sistema de runes i el balanceig de les estadístiques del joc.

Estètica	5%
Narrativa	5%
Mecàniques	15%
Tecnologia	75%

L'Abel es va encarregar principalment a dissenyar i dur a terme la majoria de l'estètica general del joc i de les interfícies. Per aquesta raó la major part del seu treball es troba a l'apartat d'estètica. Combinant l'apartat de tecnologia i estètica, ha dut a terme l'edició, adaptació i ordre de les animacions de combat de la protagonista i dels enemics, i el sistema de partícules dels atacs de la protagonista. De la mateixa manera que en Sergio, s'ha involucrat a la part conjunta de la narrativa.

Estètica	60%
Narrativa	5%
Mecàniques	15%
Tecnologia	20%

En Daniel va aportar també una petita part a la part d'estètica, sobretot per a l'estructura de l'inventari i aspecte dels diàlegs. A més, va desenvolupar la narrativa a més de les decisions inicials fetes en conjunt. Per això el seu percentatge és una mica major a la resta. Per acabar, on més temps ha dedicat ha estat a la part de tecnologia, treballant en l'inventari, el sistema de cofres, els diàlegs, el moviment, el combat, i mapa.

Estètica	5%
Narrativa	10%
Mecàniques	15%
Tecnologia	70%

2 Estudi de Viabilitat

Per a fer possible la creació del projecte, s'ha hagut de fer un estudi preliminar de tots els requisits. S'han hagut de tenir diversos aspectes en consideració. Cal parlar de la tecnologia, els pressupostos, els recursos utilitzats, i el temps requerit, a més de crear un perfil a qui anirà dirigit el projecte.

2.1 Viabilitat Tecnològica

Per a poder portar a terme el projecte, cal tenir a disposició una sèrie de programari i maquinari adequat.

2.1.1 Hardware

Els recursos físics utilitzats són, senzillament, l'ordinador personal de cadascun dels integrants de l'equip. Aquests ordinadors, però, ha estat necessari que fossin suficientment potents per a poder utilitzar un motor de videojocs professional de forma fluida. A més, cal tenir com a mínim, un comandament de consola per a cadascú.

Els ordinadors disponibles han estat:

- MSI GL65 Leopard (1500 €)
    - CPU: Intel Core i7-10750H
    - RAM: 16 Gb
    - GPU: NVIDIA GeForce RTX 2070 SUPER
- PC de sobretaula (1100 €)
    - CPU: AMD Ryzen 7 2700X
    - RAM: 32 GB
    - GPU: NVIDIA GeForce RTX 3070
- PC de sobretaula (950 €)
    - CPU: AMD Ryzen 7 1700X
    - RAM: 16 GB
    - GPU: AMD Radeon RX 580

El preu total amb els ordinadors i els comandaments, que són 70 € per cada comandament, seria de 3670 €.

2.1.2 Software

Com a recursos de software, es requereix un motor de videojocs, en aquest cas, d'UE4, el qual és completament gratuït; eines de planificació, com *HacknPlan*; eines de modelatge 3D com *Blender*, que també és gratuït; *Substance Painter 3D*, mitjançant una llicència d'estudiant proporcionada per la universitat, i eines de tractament d'imatges, com *Photoshop* i *Illustrator* (dels quals en podem fer ús gràcies al fet que un integrant del grup ja tenia les llicències prèviament a causa d'estudis previs).

A més, es requereix la utilització de qualsevol paquet gratuït necessari de l'*Epic Store* per al desenvolupament del projecte, a més a més de qualsevol recurs que es pugui extreure d'Internet de forma gratuïta i se'n pugui fer ús sense copyright, per tal de facilitar al màxim la realització del projecte. Entre altres elements, poden entrar en aquesta categoria: models 3D, personatges, animacions, textures, etc.

2.2 Recursos Humans

Donat el tipus de projecte, el desenvolupament del joc en la seva totalitat requeriria els següents rols:

- Dissenyador de nivells
- Dissenyador de so
- Artista 3D
- Dissenyador de UI /UX
- Animador 3D
- Artista VFX
- Programador de la IA
- Programador de *gameplay*
- Programador de UI/UX
- Tester

No obstant això, el desenvolupament del prototipus es realitzarà per un equip format per tres persones, cadascuna realitzant tasques de diversos rols.

2.3 Viabilitat Econòmica

Pel que fa als recursos econòmics, com ja s'ha dit són completament nuls. Tot el treball a fer cal que sigui a través d'eines gratuïtes i sense cap problema amb el copyright. Tot i així, el cost requerit pels recursos humans es podria estimar utilitzant les eines payscale.com i salarios.infojobs.net, que dona el resultat dels següents sous bruts a l'any:

- Dissenyador de nivells (24.960 €)
- Dissenyador de so (36.851 €)
- Artista 3D (24.306 €)
- Dissenyador de UI /UX (29.485 €)
- Animador 3D (24.306 €)
- Artista VFX (31.572 €)
- Programador de la IA (67.632 €)
- Programador de *gameplay* (30.723 €)
- Programador de UI/UX (32.000 €)
- Tester (38.345 €)

El temps de desenvolupament d'un joc AAA estàndard seria d'uns tres anys.

2.4 Públic Objectiu i Perfil del Jugador

El perfil de jugador objectiu es basa en un adolescent o jove adult que tingui els videojocs com a passatemps habitual, de forma que estigui habituat a jugar i pugui aprendre a controlar el personatge de forma ràpida. Ha d'estar interessat en la mitologia o el coneixement de la religió. Com a afegit, li ha d'agradar que li expliquin històries, siguin amb llibres o amb series i pel·lícules. A més, ha d'estar interessat en els jocs ràpids i que demanin concentració, i que no tingui problemes amb què se li ofereixin reptes complicats. Ha de tenir paciència i moltes ganes de millorar.

La fitxa d'un Jugador Ideal seria: Adult, d'uns 24 anys, a qui li agradi conèixer la mitologia i li agradi llegir o veure metratges amb una trama profunda, amb capacitat de concentració i passió pels videojocs.

2.5 Estudi de Mercat

Un cop s'ha estudiat com de viable és tecnològicament i econòmicament el projecte, la quantitat i cost dels recursos per dur-ho a terme i quin és el públic objectiu i el perfil de jugador ideal, cal realitzar i analitzar l'estat del mercat dels *Hack and Slash*.

2.5.1 Estat de l'Art

Els jocs de tipus *Hack and Slash* estan caracteritzats per la intensa acció, el combat estilitzat i els elements sobrenaturals. Els títols més icònics i reconeguts d'aquest gènere són *Devil May Cry* i *Bayonetta*, que han establert un estàndard alt en termes de jugabilitat, estil i narrativa. Un nou joc del tipus *Hack and Slash* que capturi l'essència d'aquests títols té el potencial de guanyar-se, o almenys cridar l'atenció, als fanàtics d'aquests dos gegants del gènere.

A continuació es farà una anàlisi general a la competència en aquest sector:

Figura 1, Videojoc Devil May Cry

Devil May Cry Series: La sèrie de videojocs *Devil May Cry* és coneguda per la acció frenètica, estil visual únic i una mescla de combat cos a cos i a distància. Ha estat elogiada per la jugabilitat polida i la història intrigant.

La història d'aquesta saga comença amb Dante, un caçador de dimonis que és meitat humà meitat dimoni, que va veure com quan era jove com Eva, la seva mare, va ser assassinada per dimonis (d'aquí el seu odi i perquè es va fer caçador de dimonis) Dante té un germà bessó anomenat Vergil, amb qui comparteix una gran rivalitat per la visió oposada de com tractar amb els dimonis. Durant tota la saga en la narrativa es pot apreciar diferents conflictes que involucren la cerca de venjança i al mateix temps de redempció de Dante i el conflicte entre humans i dimonis.

Figura 2. Videojoc Bayonetta

Bayonetta Series: *Bayonetta* ofereix un estil de joc similar, però agrega una protagonista carismàtica i un to més extravagant. El joc s'ha destacat per la estètica cridanera i mecàniques de combat profundes.

La història parla sobre Bayonetta, una bruixa amnèsica amb poders màgics extraordinaris, que just es desperta d'un llarg son. A mesura que recupera els records, es veu envolta en una lluita entre diferents faccions, tant faccions angelicals com faccions demoníaques entre altres, a la recerca del control d'un artefacte misteriós anomenat *Eyes of the World*. També durant el joc apareix Jeanne, una amiga i rival de Bayonetta, i es descobreix que també està involucrada en la lluita per l'artefacte misteriós. Durant tota la saga podem veure com té una narrativa i personatges molt complexos, essent el millor exemple i el personatge més característic la mateixa Bayonetta per la seva personalitat carismàtica.

Figura 3. Videojoc Nier:Automata

Nier:Automata: Encara que no és un joc purament *Hack and Slash*, *Nier:Automata* ha estat reeixit en aquest gènere en combinar una narrativa emocional i filosòfica, amb acció intensa. Ha demostrat que la mescla de gèneres pot ser reeixida.

La història de *Nier:Automata* posa al jugador en un futur distòpic en el qual Terra ha estat envaïda per màquines alienígenes. Els humans han fugit a la lluna i han creat uns androides de combat que intenten reclamar la Terra de nou. El jugador es fica en el paper de l'androide protagonista 2B, i el seu company 9S, que han de lluitar amb les màquines invasores i descobrir la raó del conflicte. A mesura que el joc avança, els personatges comencen a qüestionar-se tota mena de coses, fins a tal punt de qüestionar-se la seva pròpia humanitat i el significat de tot. Un dels punts forts de *Nier:Automata* és els seus múltiples finals, i com per entendre la trama del joc completament, s'ha de jugar múltiples cops i obtenint els diferents finals. Una cosa que també destaca molt és com la banda sonora del joc és una part essencial i ajuda a la construcció de l'atmosfera commovedora del joc.

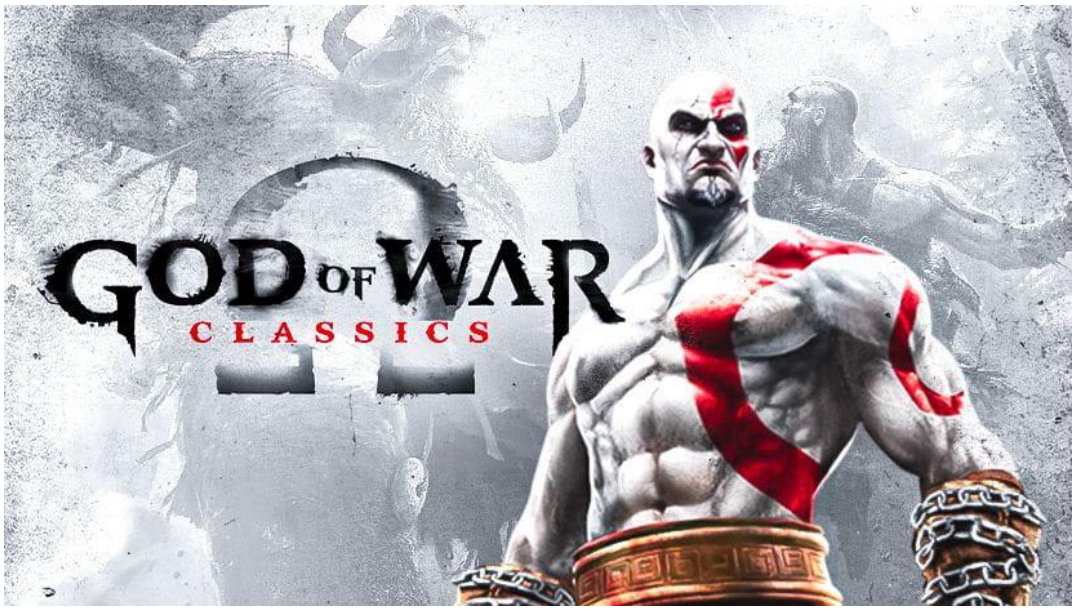

Figura 4. Videojoc God of War

God of War Series: Aquesta saga de videojocs és coneguda en gran part gràcies al carismàtic protagonista, Kratos, un antic guerrer espartà, que lluita contra deïtats i tota classe d'éssers mitològics en cerca de venjança i redempció.

La història comença amb Kratos, buscant venjança contra els Déus de l'Olimp, especialment contra Ares, el Déu de la guerra, per enganyar-lo quan es va convertir en súbdit seu amb la finalitat d'acabar amb els seus enemics, fent-lo assassinar a la seva dona i filla. A mesura que avança la història i els jocs, Kratos posa fi a la vida de diferents Déus de la mitologia grega, fins i tot posa fi a la vida de Zeus, el Déu de l'Olimp. Un cop acaba amb Zeus, Kratos apareix en una zona completament diferent amb un fill, Atreus, que va tenir amb la seva segona dona Faye, també ja morta. En aquest punt la història de *God of War* que s'havia centrat en la mitologia grega, es comença a barrejar amb la mitologia nòrdica, ja que apareixen personatges com Thor, Odin, o com es desvela a *God of War (2018)*, Atreus és en realitat Loki.

God of War ha estat molt reconeguda a causa de la seva extensa narrativa i al desenvolupament de personatge durant tota la saga de Kratos.

Figura 5. Videojoc Hades

Hades: És un videojoc *indie* que ha guanyat moltíssima popularitat. Una d'aquestes raons és la de barrejar elements *Hack and Slash* amb elements *roguelike* en un món mitològic.

Hades segueix la història del príncep de l'inframon, Zagreus, que vol escapar d'aquest món reinat pel seu pare Hades i conèixer a la seva mare Perséfone.

Aquest videojoc, igual que *Nier:Automata*, té diversos finals i, a mesura que es juga i s'obtenen els diversos finals, es pot arribar a comprendre millor la trama.

Un dels elements més destacats d'*Hades* és la generació procedural de masmorres, creant així una experiència de joc diferent cada cop que es juga i s'hi intenta aconseguir un final alternatiu.

2.5.2 Comparació Estat de l'Art

Un cop analitzats individualment els gran referents del mercat de *Hack and Slash*, ara compararem diferents aspectes generals d'aquests videojocs:

- Dificultat [1-10]
- Apartat estètic [1-10]
- Moviment [1-10]
- Narrativa [1-10]
- Duració aproximada [Hores]

Joc	Dificultat	Estètica	Moviment	Narrativa	Duració
Saga Devil May Cry	8	9	10	7	56h-70h
Saga Bayonetta	7	10	10	8	37h-42h
Nier:Automata	6	9	8	10	20h-30h
Saga God of War	7	10	9	9	68h-89h
Hades	8	9	9	10	20h-30h

Taula 1. Taula comparativa dels jocs rellevants actuals del gènere Hack and Slash

Cal destacar que la saga *Devil May Cry* té diversos jocs de duració aproximada d'unes 10 hores cadascun, *Bayonetta* conté tres jocs d'una duració aproximada d'unes 12 hores cadascun, dins la saga *God of War,* els primers tenen una duració aproximada d'unes 10 hores i els dos últims d'unes 25 hores i l'*Hades* cada intent de fugida té una duració aproximada d'uns 20-40 minuts, però el que es veu a la Taula 1 és la duració aproximada del joc sencer amb tots els finals.

Observant la Taula 1, es pot veure el següent:

- La saga *Devil May Cry*, la saga *Bayonetta* i la saga *God of War* tenen puntuacions bastant similars en quant a l'enfoc que donen, essent *Devil May Cry* qui potser aposta una mica més per l'acció del joc mentre que *Bayonetta* i *God of War* aposten més per la narrativa i estètica.
- *Nier:Automata* aposta molt més per la seva narrativa i estètica que no pas per l'acció.
- *Hades* al tenir elements *roguelike* fa que la dificultat del joc augmenti bastant.
- *Nier:Automata*, *Hades* i els últims *God of War* són els jocs que més duració té si tenim en compte els jocs per separat i no la saga al sencer, això pot ser degut a la seva aposta per la narrativa.

Tenint això en compte, i el nostre perfil de jugador ideal, el producte final s'hauria d'enfocar al mercat d'una manera similar als jocs *Devil May Cry* i *Bayonetta*, pot ser també com a factor diferencial un enfoc més profund i potser madur com fa *Nier:Automata*.

2.5.3 Model de Negoci

Per entrar de la millor manera possible al mercat, s'hauria de fer com han fet la gran majoria d'aquest sector i és seguir el model tradicional de venta de videojocs, és a dir, un únic pagament per poder gaudir de l'experiència de joc.

També seguint la tendència general del mercat dels videojocs, i com per exemple fa *Devil May Cry 5*, és treure un DLC (Contingut Addicional Descargable) expandint la història del joc.

3 Planificació

3.1 Diagrama de Gantt

El diagrama es va pensar per a començar a implementar el joc el desembre de 2022 i tenir-lo acabat per agost de 2023. El diagrama està dividit en quinzenes i seccions. Cada secció té un color, que és on s'ha planificat treballar.

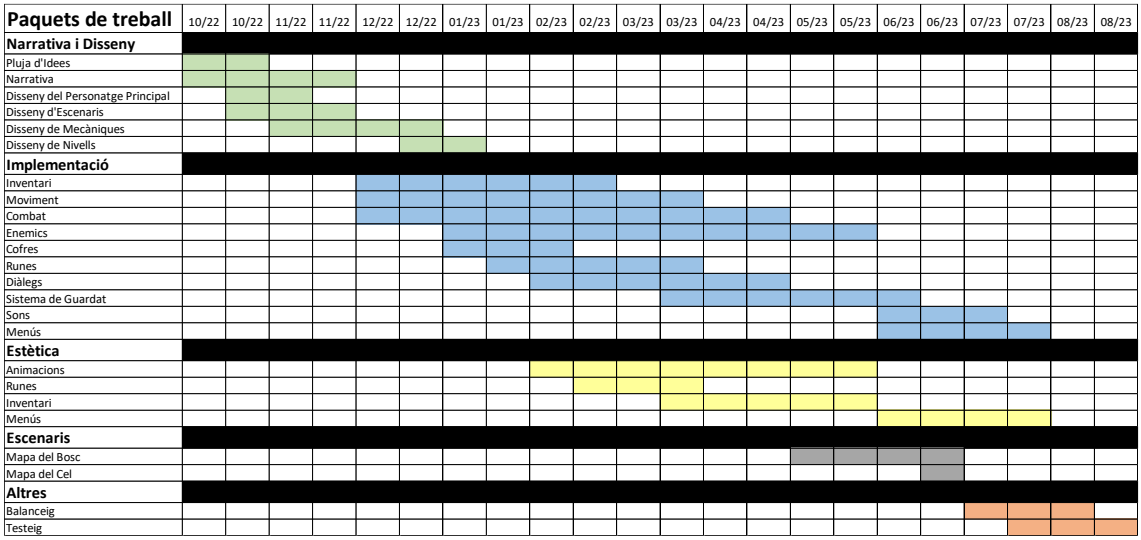

Figura 6. Diagrama de Gantt

3.2 Metodologia

La metodologia seguida ha estat en tot moment orientada a seguir la planificació de la millor forma possible i poder dividir les tasques de forma paral·lela sense generar problemes. Les eines utilitzades per a portar a terme aquest objectiu han estat *GitHub* i *HacknPlan*.

HacknPlan és una eina de planificació i distribució de tasques. Aquesta l'hem utilitzada per a mantenir un registre de les diverses tasques a realitzar, en progrés i acabades de cadascun dels integrants.

GitHub és una eina que serveix per a mantenir un projecte comú en un emmagatzematge remot i treballar de forma local, i tenir un sistema de gestió de canvis i versions. Els canvis fets es van actualitzant al repositori en remot, i es van descarregant els canvis fets pels altres en el projecte local. D'aquesta forma s'ha pogut treballar en paral·lel sense gaires problemes.

4 Marc de Treball

4.1 Conceptes Previs

En aquest apartat es comenten tots els elements i detalls d'UE4 que s'ha cregut oportú per a poder entendre la memòria de forma correcta.

4.1.1 Vocabulari específic

Les següents paraules s'utilitzen diverses vegades a l'escrit, i en ell es dona per fet el coneixement del significat. Aquí es dona l'explicació de cadascuna d'elles:

- *Blueprint*: cadascun dels objectes i nodes propis d'UE4.
- *Event*: node que inicia l'execució d'un fil de codi.
- *Input*: qualsevol senyal d'entrada donada pel jugador.
- *Widget*: subtipus de Blueprint utilitzat per al disseny d'interfícies i l'apartat visual.

4.1.2 ActionMappings

A l'hora de definir els controls del joc, es pot programar la funcionalitat i fer que cada funció s'executi quan es clica una tecla en concret. Aquesta manera, tot i ser útil, no és gaire còmoda, ja que si després es vol canviar els controls s'ha d'anar a buscar el tros de codi concret on està. A més, s'haurien de crear nodes de diferents *Events* per executar un mateix codi si es vol crear un joc que suporti teclat i ratolí i controlador, per exemple.

UE4 permet gestionar els controls d'una forma més còmoda: els *ActionMappings*. Consisteixen a crear una acció (la qual pot tenir un nom escollit pel jugador) i assignar-li diferents botons o tecles. Sempre que algun dels controls de la llista sigui clicat, s'executarà l'*Event* d'aquesta acció. A la Figura 7 es poden apreciar tots els *ActionMappings* que s'han definit per aquest projecte. Per arribar aquí cal anar a *Project Settings*, a l'apartat d'*Input*. Es pot definir un nou *ActionMapping* clicant la icona de més al costat del títol que es veu a la figura. Un cop es crea, es poden afegir noves tecles amb la icona de més del costat del respectiu *ActionMapping*. Com es pot apreciar en aquest cas, per l'acció *OpenRunesMenu* s'han definit dos controls: la tecla I i *Gamepad Special Right* (el nom dels botons del controlador es poden comprovar a l'Apartat 11.3).

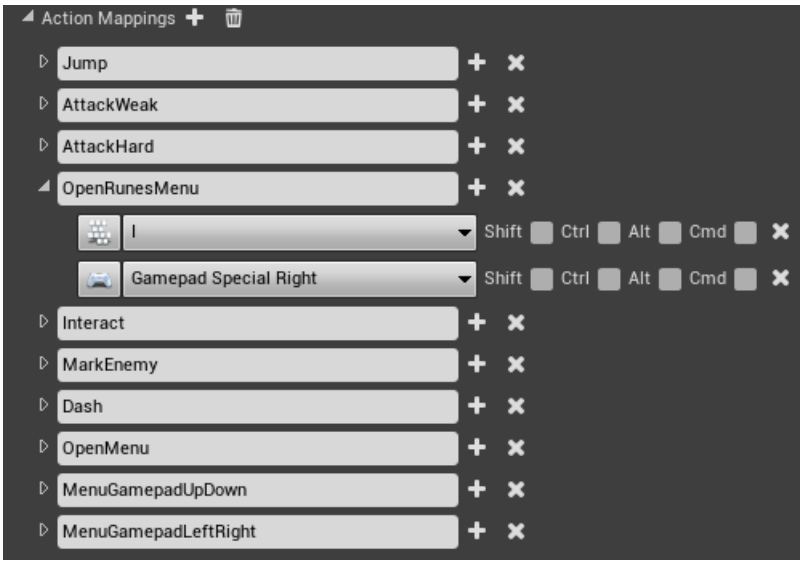

Figura 7. Action Mappings del projecte

4.1.3 NavMesh

La *NavMesh* és un component del mateix UE4 que permet calcular la superfície transitable. D'aquesta manera, quan s'utilitzen personatges que no controla el jugador, els *Non Playable Character* (NPC) es poden calcular el camí que han de seguir quan han d'anar a un punt concret, tenint en compte que pel camí pot haver-hi obstacles i no necessàriament haurà de ser en línia recta. Per tal de calcular la *NavMesh* només cal crear l'objecte a l'escena i donar-li les mides necessàries. Un exemple seria el mostrat a la Figura 8. Com es pot apreciar, s'ha creat una caixa al voltant del pla per tal d'encapsular el terreny. Quan es crea es pot veure com es genera aquest color verd que indica la superfície navegable (en cas que no es mostri cal clicar la tecla P). Una bona característica d'aquest component és que calcula la superfície un cop i l'emmagatzema per fer servir les dades de forma estàtica. Per aquest mateix motiu, es pot crear una caixa que sigui molt més gran que el mapa, ja que no afectarà en res al rendiment del joc.

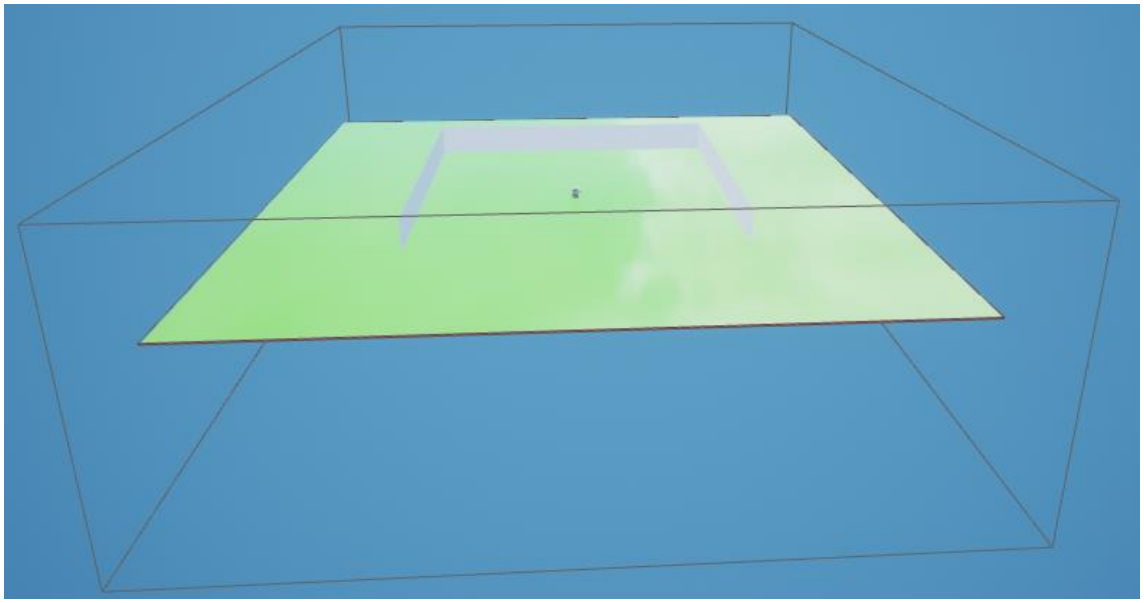

Figura 8. Exemple NavMesh

Tot i que no es sap al 100% quin és l'algorisme utilitzat pel *pathfinding*, existeix una gran probabilitat que aquest sigui l'A*, o alguna petita variant. A l'algorisme A* es creen dues llistes de nodes: una de nodes ja explorats i una altra de nodes sense explorar. Cada node té dos valors associats: g i h. La "g" és el cost acumulat des del node inicial fins l'actual. L'"h" és una estimació del cost des del node actual fins el final. Per cada node es calcula una puntuació, fruit de la suma de g i h. El node amb la puntuació més baixa s'afegeix a la llista d'explorats. Finalment, es comproven els valors g de tots els veïns. Si algun d'aquests és inferior a l'actual, s'actualitza el node amb el nou cost per tal d'intentar obtenir el camí més eficient possible. S'itera fins haver explorat tots els nodes o fins arribar a l'objectiu.

4.1.4 Behavior Tree

Aquest *Blueprint* és una de les múltiples formes que té UE4 per desenvolupar una intel·ligència artificial. La part bona d'aquesta manera és que permet crear un arbre de comportament que és molt senzill de debugar i és molt més visual que, per exemple, un

arxiu de codi en C++. Té dues parts principals: el *Blackboard* i el *Behavior Tree*, BT. El *Blackboard* permet crear *Keys*, unes variables que s'utilitzen al BT per prendre decisions i canviar comportaments del personatge. El BT és l'arbre que permet crear diferents seccions per executar comportaments, segons els valors que tinguin les *Keys* que creem al *Blackboard*.

En el cas del BT, es parteix del node *Root*, que és la base i no es pot modificar. Té l'aspecte de la Figura 9. A partir d'aquest es poden utilitzar els *Selector* i *Sequence* per triar el comportament, explicats a continuació, i les *Task*, que executen una funció específica i es poden utilitzar tant els d'UE4 com uns propis del desenvolupador.

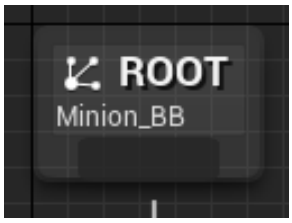

Figura 9. Node Root

Cada cop que es realitza una *Task*, aquesta retorna si s'ha completat correctament. Això s'utilitza tant per les *Sequence* com pel *Selector*, però es gestionen de manera diferent. En el cas del *Selector* (que acostuma a ser el node que penja del *Root*), si se li pengen dues *Sequence*, sempre executarà la primera. Si les tasques s'executen correctament tornarà a executar el mateix comportament, però si alguna falla sortirà de la *Sequence* i executarà la segona. Es pot apreciar l'aspecte del *Selector* a la Figura 10.

Figura 10. Selector

D'altra banda, el node *Sequence*, quan una tasca es completa amb èxit, passa a la següent en comptes de repetir-la. A més té una característica que, si bé es pot utilitzar amb el *Selector*, en aquest cas és més comú. Si es fa clic dret sobre el node es pot afegir un *Decorator*. El *Decorator* permet afegir comportament sobre el node. Un exemple seria el mostrat a la Figura 11. En aquest cas el *Decorator* permet decidir si executar les tasques de la seqüència només si es compleix la condició, que ve donada per una de les *Keys* que s'han esmentat anteriorment. A la Figura 12 es poden veure les opcions que es poden modificar.

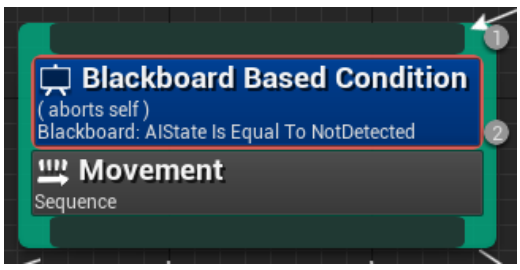

Figura 11. Sequence amb Decorator

L'*Observer aborts* habitualment té com a valor *None*, però això implica que la condició es comprova només abans d'entrar. Canviant a *Self*, com en aquest exemple, implica que encara que estigui en mig de l'execució, si *AIState* deixa de ser *NotDetected* deixarà d'executar aquesta *Sequence* al moment. D'altra banda, la condició s'expressa a l'apartat de *Blackboard* on permet triar la *Key* definida a *Blackboard Key*, el valor que es vol comparar a *Key Value* i el tipus de comparació a *Key Query* (igual, més gran, més petit, més gran o igual...).

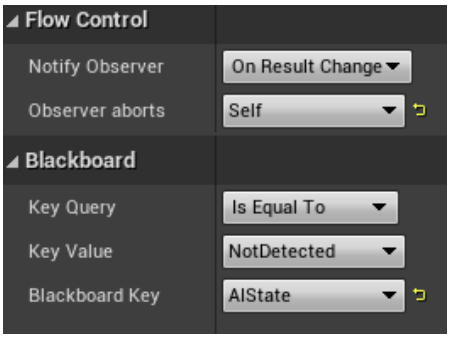

Figura 12. Opcions del Decorator

Algunes de les *Task* més utilitzades són *MoveTo* i *Wait*. La tasca *MoveTo* es pot veure a la Figura 13 amb els dos paràmetres més importants: *Acceptable Radius* i *Blackboard Key*. El primer permet indicar a quina distància ha d'estar de l'objectiu. És especialment útil quan es vol seguir a un altre personatge ja que no es podrà arribar exactament a la seva posició. El segon és la *Blackboard Key*, que necessàriament ha de ser una variable de tipus vector per indicar la posició objectiu. La tasca *Wait* es pot apreciar a la Figura 14 amb els dos paràmetres que es poden modificar: *WaitTime* i *RandomDeviation*. El primer és el temps que es vol fer esperar com a tal. El segon s'utilitza quan es vol que aquest temps variï, així doncs la suma i resta d'aquest valor al *WaitTime* serà el màxim i mínim, respectivament. Es veu clarament com aquest valor s'indica dins del mateix node amb els caràcters "+-", que permeten fer-ho més gràfic i comprensible.

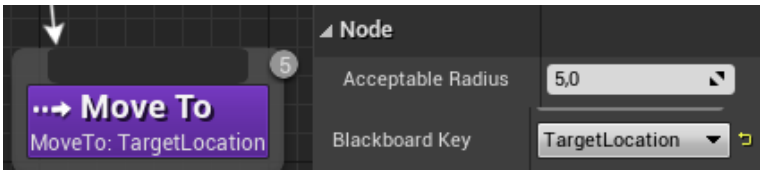

Figura 13. Node MoveTo amb paràmetres destacats

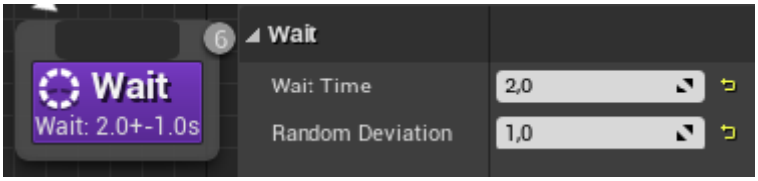

Figura 14. Node Wait amb paràmetres

4.2 Referències

Les majors referencies utilitzades per a la creació del treball han estat dos sagues de jocs: *Bayonetta*, veure Figura 15, i *Devil May Cry*, veure Figura 16. D'aquests jocs, s'han agafat com a inspiració la part conceptual del moviment i combat del personatge, a més de

l'estètica. És a dir, s'ha agafat la fluïdesa que mostren tots dos jocs en el moviment i s'ha intentat fer quelcom similar i adaptada a la implementació.

La part més diferenciadora és la del món obert, la obtenció d'objectes, i el sistema de diàlegs. Per començar, el món ideat pel joc és molt obert, permetent el jugador anar allà on vulgui sense restriccions. En contrapartida, *Bayonetta* per exemple, té un món molt lineal i separat per nivells. A més, els objectes no cobren gaire importància en cap dels jocs esmentats, mentre que en el dissenyat tenen un protagonisme elevat per a la millora del personatge i facilitats al jugador. Com a tema completament afegit, cal remarcar el sistema de diàlegs. Aquest no existeix en cap dels dos jocs comentat, i genera una gran profunditat a l'hora d'idear les diferents zones i objectes importants a aconseguir, ja que obliga al jugador a recopilar informació i contrastar-la amb el què es va trobant al món.

Per tant, tot i tenir una base similar en la part de moviment i combat, la resta d'apartats fan que es generi una manera de jugar al joc molt diferent dels jocs inspiració.

Figura 15. Bayonetta

Figura 16. Devil May Cry

5 Disseny del Videojoc

5.1 Narrativa

5.1.1 Guió

Fa molt de temps, hi va haver una gran guerra entre els àngels per la successió al tro ancestral. Existien dos bàndols: un creia en la puresa de la sang angelical, i l'altre acceptava als Àngels amb meitat sang humana, els mixtes. Els Àngels mixtes tenien un gran poder, i van ser una de les armes més importants per la guerra en aquest segon bàndol. Tot i així, els denominats "Sang Pura" van acabar vencent. Com a resultat de la derrota, el bàndol perdedor va ser desterrat a l'Infern, i se'ls hi va denominar "Dimonis".

Temps després, una Dimoni de sang mixta anomenada Aria, va tenir un fill. El petit nadó va néixer amb poders sobrenaturals, molt per sobre de les capacitats normals dels Àngels. Déu, atemorit pel poder del Sang Mixta, va ordenar el seu assassinat. Des de llavors, Aria, consumida per la ira i el dolor, té com a únic objectiu a la seva vida el venjar-se de Déu.

Aria comença el seu viatge a les profunditats de l'infern, on s'ha d'enfrontar amb Àngels infiltrats que intenten eliminar els Dimonis. Aria lluita amb habilitat i determinació, sabent que cada Àngel que derrota l'apropa més al seu objectiu final: venjar la mort del seu fill. Una vegada en el món terrenal, creua valls i muntanyes, rius i oceans, i enfronta criatures temibles i perilloses, trepitjant a tot ésser, Àngel o criatura, que es creui en el seu camí.

Per fi, Aria arriba al cel, custodiat per Àngels poderosos que intenten aturar el seu avanç. Aria s'enfronta amb ells, sense importar-li els sacrificis que hagi de fer. S'enfronta Àngels cada vegada més poderosos, cadascun més difícil de vèncer que l'anterior.

Finalment, Aria arriba al tro. Allà, s'enfronta a la batalla final contra Déu. Aria sap que acabar amb ell és una tasca gairebé impossible, però la seva determinació és irrompible. Només pensa en la seva venjança i en matar a Déu.

5.1.2 Escenaris

Els diversos escenaris que hi apareixen estan molt relacionats amb la trama, i tenen una rellevància real a l'hora d'explicar la història i comunicar al jugador en quina situació es troba en cada moment.

5.1.2.1 Infern

L'Infern és l'escenari inicial del joc. En aquesta part, el jugador comença a fer-se als controls del personatge, i pot investigar una mica com és la historia del joc. Aquest és un escenari ple de caos, que reflecteix el càstig imposat sobre els Dimonis per perdre la batalla angelical. En aquest espai de joc, el jugador farà les primeres passes, i tindrà els primers combats amb àngels.

5.1.2.2 Món Terrenal

El Món Terrenal és l'escenari més neutre. Representa el punt mig entre el Cel i l'Infern, allà on es poden trobar tot tipus de criatures i espècies. En aquesta zona del joc, el

jugador podrà aprendre més sobre la historia ocorreguda després de la guerra, i podrà veure diferents punts de vista del fi de la guerra i els problemes que aquesta va portar. Aquesta zona representa la balança entre diferents idees i pensaments. També és la part on el jugador ha d'aprendre la major part d'habilitats per a poder passar a la zona final i més difícil del joc.

5.1.2.3 Poble

El Poble és una subzona dins el Món Terrenal. Aquesta és una zona segura, on el jugador pot comerciar, obtenir missions, buscar informació, etc. En aquest espai s'ajunten espècies de tot tipus, i es pot interactuar i dialogar tant amb humans, com Dimonis i Àngels.

5.1.2.4 Cel

El Cel és la zona final del joc. Representa la majestuositat i la perfecció, com a contrast de les idees radicals de molts Àngels. És la zona més difícil del joc, a més de la més hostil. En aquest espai es pressuposa que el jugador està ben preparat i sap com controlar bé el personatge. Mentre es trobi en el Cel, el jugador només es trobarà Àngels.

5.1.2.5 Espai de Joc

Cada escenari és un món obert separat dels altres, i només es pot canviar d'un a l'altre a través dels punts de connexió concrets. En qualsevol moment es permet el pas, ja que el jugador sempre té l'habilitat. Això és útil per a l'obtenció de materials concrets, la possibilitat de completar missions, i qualsevol cosa que requereixi canviar de món. L'estructura seguida és pot veure de forma visual a la Figura 17.

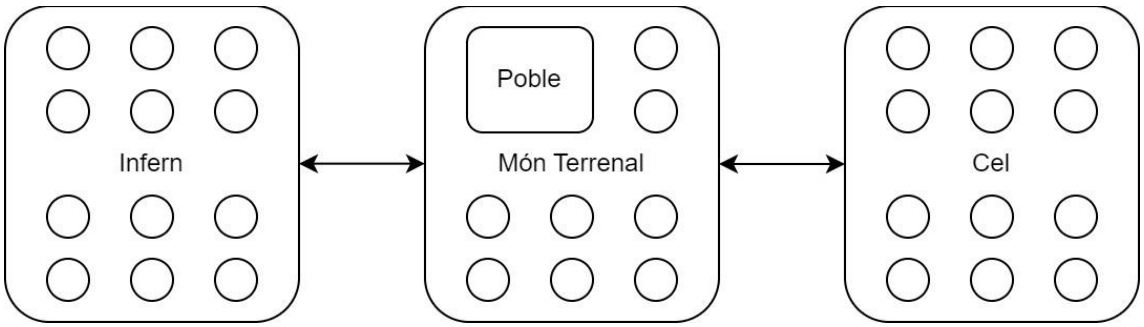

Figura 17. Espai de Joc

5.1.2.6 Connexió entre la trama i els escenaris

Com ja s'ha dit, cada escenari té un significat i funcionalitat propis. A mesura que s'avança en els escenaris, s'avança en la historia. Per a poder seguir la trama, no queda altra que anar completant els objectius principals del joc. Per tant, la connexió és molt profunda. Tot i que es pot canviar d'escenari sense problemes en cap moment, el jugador no es podrà sentir còmode fins que no hagi superat els reptes que pertoquen a cada zona, i no podrà acabar el joc. És a dir, l'escenari evoluciona poc a poc de la mateixa forma que ho fa la trama, i totalment connectat a ella.

5.1.3 Personatges

Els personatges principals (Aria, la protagonista, i Sabaoth o Déu, que és l'enemic principal) estan connectats a causa de la narrativa del joc, com s'ha explicat anteriorment a l'Apartat 5.1.1. A continuació s'analitzaran diferents característiques de cadascun.

5.1.3.1 Aria

Nom	Aria
Raça	Àngel Mixta / Dimoni Mixta / Sang Mixta
Característiques	És una dona jove, d'uns 28-32 anys, amb els cabells obscurs i les puntes del cabell vermelles. Te la pell extremadament pàl·lida i la pell al voltant dels ulls de color vermella, donant així aquest aire "demoníac" en un cos humà. Per combatre porta un vestit de cuir completament negre amb unes botes i guants negres i espases a dos mans, per obtenir una aparença ofensiva màxima.
Anàlisi psicològic	Tenim tres aspectes a tenir en compte en quant a l'anàlisi d'Aria: **Trauma**: La pèrdua del seu fill ha deixat una marca profunda en Aria i és el desencadenant d'una sèrie de reaccions emocionals i psicològiques, com la ira, la tristesa, la culpa i la desesperació. Aquests sentiments son la força darrere la seva venjança. **Motivació**: Donat el seu trauma, la motivació d'Aria per a buscar venjança contra Déu és comprensible. El desig de fer justícia la portarà a prendre decisions extremes i a enfrontar qualsevol desafiament per a aconseguir l'objectiu: acabar amb Sabaoth. **Personalitat**: Donat el seu passat traumàtic i la seva motivació de venjança, Ària és un personatge complex i fosc. És una persona reservada, intensa i decidida en la seva cerca. A causa del seu passat, l'hostilitat serà la part més dominant de la seva personalitat, especialment contra personatges relacionats amb Sabaoth.

5.1.3.2 Sabaoth

Nom	Sabaoth
Raça	Àngel/ Deïtat
Característiques	És un ens, d'edat indeterminada a causa del seus estatus de Déu. Té una vestidura blanca molt distingida per fer reconegut aquesta distinció i superioritat sobre la resta d'àngels. També té un gran martell amb el que elimina a tot ésser que es fica al mig del seu camí.
Anàlisi psicològic	Tenim diversos aspectes a tenir en compte en quant a l'anàlisi de Sabaoth: **Condescendència**: A causa de l'estatus de Deïtat, Sabaoth és un ésser condescendent. Es sent superior a altres criatures, com els humans i els Sang Mixta com a Ària. Aquesta actitud el porta a menysprear les vides d'aquells als qui considera inferiors. **Implacable**: Sabaoth no dubta a l'hora de prendre mesures extremes per a aconseguir els objectius, com per exemple el fet

	d'ordenar als àngels acabar amb el fill d'Aria per frenar el creixement de poder i al mateix temps acabar amb un ésser que tenia potencial per superior a ell per part dels Sang Mixta. **Falta d'empatia**: Sabaoth manca per complet d'empatia cap als altres, la qual cosa ho fa indiferent al sofriment que causa i el fa especialment perillós, ja que no té objeccions a causar dolor i sofriment a uns altres. **Poder i control**: La sensació de poder i control és una part important de la personalitat de Sabaoth. Com un déu que no dubta a prendre vides i que és condescendent, busca mantenir el seu domini i autoritat sobre el món en el qual viu.

5.1.4 Objectes

Dins del joc es podran trobar dos tipus principals d'objectes: les runes i les pocions. A part d'aquests n'hi ha d'altres, com per exemple les dues espases que porta el personatge, però aquestes no varien cap tipus d'estadística durant la partida ni es pot interactuar amb elles. Només s'utilitzen per atacar als enemics.

5.1.4.1 Runes

Les runes són unes pedres màgiques que es poden trobar per tot el mapa. Aquestes aporten al jugador estadístiques extra si les incrusta a les espases. Existeixen tres rareses: comuns, èpiques i llegendàries. Les rareses s'ordenen d'esquerra a dreta i principalment indiquen la facilitat amb la que es troben al mapa i la quantitat de millora que donen.

- Les comuns es troben molt fàcilment per tot el mapa, inicialment ajuden al jugador a tenir un extra, però són poc útils de cares a la recta final del joc ja que aquest augment d'estadístiques resulta insuficient.
- Les èpiques costen més de trobar, a la part inicial serà casi impossible trobar-les perquè aporten un bon augment de les estadístiques del jugador i, ben combinades, es poden arribar a fer servir inclús a la recta final.
- Les llegendàries són les més difícils de localitzar del joc. A diferència de les anteriors, no es repeteixen. Només existeix una de cada tipus en tot el mapa. A part d'oferir les millors estadístiques, tenen un atac afegit a causa del mal elemental que aporten.

Totes les runes esmentades anteriorment es poden vendre, però només les comuns es poden obtenir a través de la compra. Tot i així, la forma més habitual de trobar les runes és obrint cofres i completant missions secundàries parlant amb els NPC.

5.1.4.2 Pocions

Durant la partida la protagonista pot conèixer alguns personatges que tenen habilitats màgiques. Aquesta capacitat és la que els permet elaborar pocions que ajuden al jugador a superar més fàcilment els enemics més complicats. Aquests objectes són més fàcils de trobar que les runes, explicades a l'apartat anterior, ja que sempre es poden obtenir parlant amb algun d'aquests personatges. La principal diferència respecte les runes és que els seus efectes són temporals.

Les pocions es poden classificar per nivell (entre un i tres). L'augment d'estadística serà sempre igual, però la duració d'aquest efecte ve marcada pel nivell. Com més elevat sigui, més temps es pot aprofitar. El problema que tenen les pocions de més nivell és que tenen menys estoc i resulten molt més cares que les de menor qualitat.

5.2 Estètica i Disseny d'Interfícies

5.2.1 Estètica

Des d'un primer moment es va decidir que el videojoc tindria elements propis de la religió, així doncs, per poder fer una millor adaptació d'aquests elements, es juga tant amb els colors com amb els elements del joc que s'explicaran a continuació:

5.2.1.1 Colors

La gamma de colors emprada tant en el personatge principal, en les diferents interfícies i en els efectes de partícules dels atacs de la protagonista són uns colors vermells foscos i jugant molt també amb les tonalitats fosques. S'han escollit aquests colors perquè es considera que representen molt bé el que volem transmetre: la protagonista del joc és un semi-dimoni. Els podem veure a la Figura 18 i Figura 19.

Figura 18. Tonalitats de la protagonista i dels menús

Figura 19. Personatge principal amb les tonalitats prèviament mencionades

Per altra banda, els enemics més bàsics i també l'enemic final Déu tenen una tonalitat molt més clares, pròximes al color blanc. Aquests colors representen la puresa d'aquests éssers, ja que tenen la gràcia de Déu. Es poden veure els colors a la Figura 20 i un exemple aplicat amb Déu a la Figura 21.

Figura 20.Tonalitats dels enemics i Déu

Figura 21. L'enemic final Déu amb les tonalitats prèviament mencionades

5.2.1.2 Elements del món

Durant el joc es troben diferents elements que segueixen aquesta tendència religiosa, especialment la relacionada amb la gràcia de Déu, per tal de aconseguir un contrast dels diferents elements envers la protagonista, com per exemple les diferents estàtues que ens permeten guardar la partida (Figura 22) o l'entrada al cel (Figura 23, Figura 24).

Figura 22. Estàtua per guardar partida

Figura 23. Escales i entrada al cel

Figura 24. Portes d'entrada al cel

5.2.2 Disseny d'Interfícies

Per al disseny d'interfícies s'ha decidit que estèticament tindran relació amb la protagonista, pel que predominen les tonalitats vermelles fosques i els negres.

5.2.2.1 Menús

Tant per al menú d'inici com per al menú de pausa, el menú d'opcions i les interfícies de crèdits i de mort del jugador s'ha optat pel mateix fons, que consisteix d'una taca vermella en un fons negre que varia la tonalitat de vermell en el temps.

Figura 25. Començament de l'animació amb el vermell fosc

Figura 26. Punt de l'animació just abans de fer la transició al vermell inicial

Tots els menús tenen la mateixa font *Demon Wings*, que ha permès així donar una millor immersió dins d'aquest món d'àngels i dimonis.

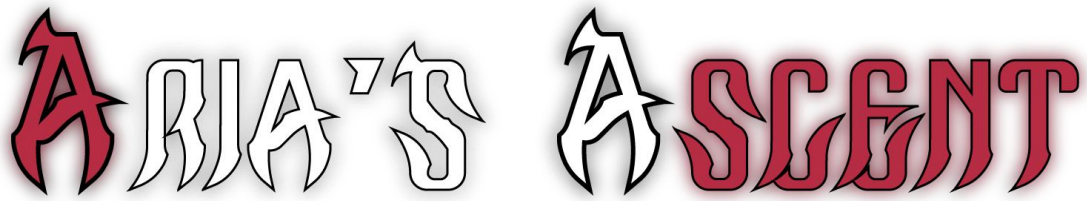

Figura 27. Exemple d'aquesta font amb el títol del joc

A més a més, al menú de pausa dins del joc es troben les espases del personatge principal que es troben al voltant de les diferents opcions disponibles, mantenint així una continuïtat amb la relació menús/interfícies i protagonista.

Figura 28. Espasa que porta la protagonista

Figura 29. Menú principal del joc

Figura 30. Menú d'opcions

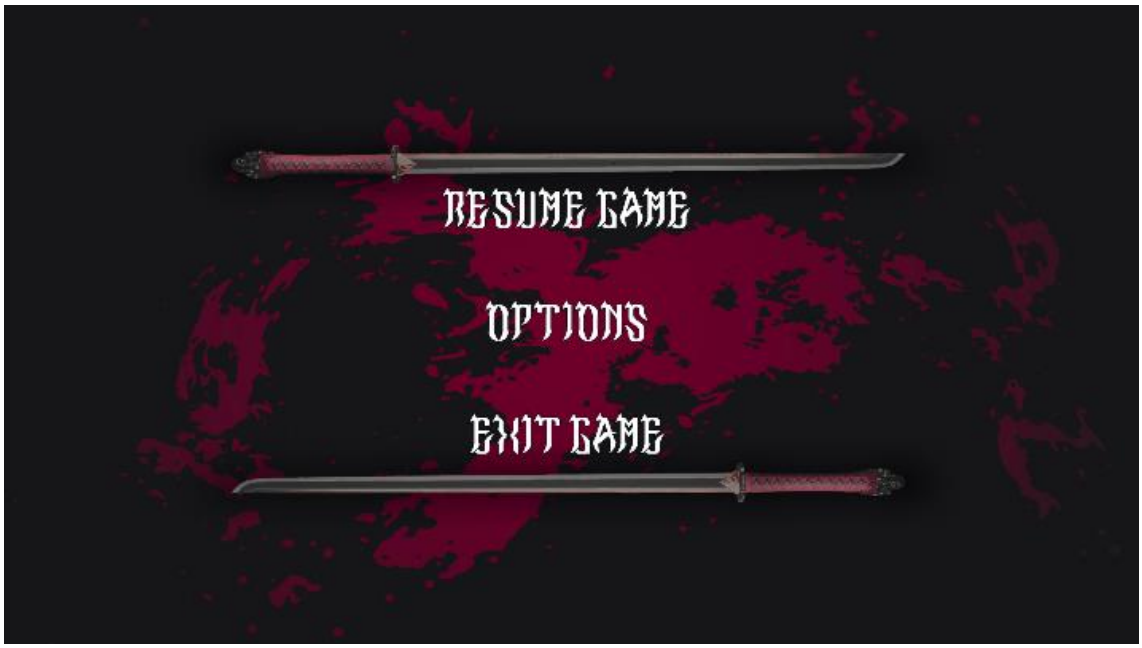

Figura 31. Menú de pausa del joc

5.2.2.2 Inventari

A l'inventari es troben diversos elements que, com els mencionats anteriorment, formen part de la relació de menús/interfícies i protagonista. Aquí es poden trobar diferències amb els altres menús mencionats anteriorment.

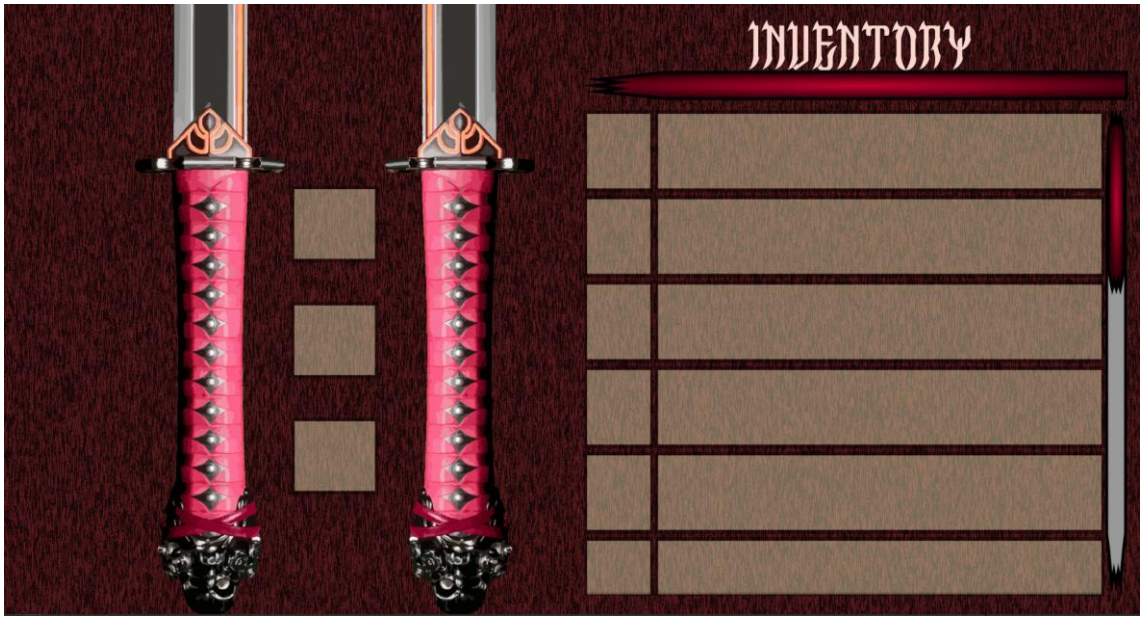

Figura 32. Inventari del joc, en aquest cas l'inventari és buit

Com es pot observar a la Figura 32, l'inventari té diversos elements a desglossar a continuació:

- A l'esquerra es poden observar els mànecs de les espases de la protagonista (Figura 33) amb unes caixes (Figura 34) on es veuran les runes equipades en aquell moment pel jugador, fins a un màxim de tres runes alhora.
- A la part dreta superior es pot apreciar el títol de la interfície, en aquest cas *inventory* (Figura 36).
- A la part dreta central i inferior s'observen les diferents caixes on es situaran les runes no equipades amb el nom i descripció. (Figura 34 i Figura 35).
- A la part dreta podem veure una barra lateral que permetrà una navegació més senzilla quan el jugador tingui molts elements a l'inventari. Veure Figura 37.

Figura 33. Mànecs de les espases de la protagonista

Figura 34. Caixa on es veurà la imatge de la runa dins l'inventari.

Figura 35. Caixa on es veurà la descripció de la runa associada

Figura 36. Títol de l'inventari

Figura 37. Barra lateral que facilitarà la navegació per l'inventari

5.2.2.3 Runes

La base de les runes del joc són uns models 3D gratuïts agafats d'internet i les figures esculpides sobre d'aquestes han estat fetes a mà utilitzant *Blender*. Per donar-li color i textura s'ha utilitzat el *Substance Painter 3D*.

Com hi ha diferents tipus de runes respecte a característiques principals (runes ofensives, defensives i de velocitat) i raresa (comú, èpic i llegendari), s'ha optat per combinar tipus i raresa per a cada runa i així poder dir ràpidament quina runa és ofensiva comú o èpica o llegendària, o una runa és defensiva, etc.

Per poder distingir les runes segons tipus s'ha optat per a que les runes ofensives tinguin tallades a la pedra una espasa (Figura 38, Figura 39 i Figura 40), mentre que les runes defensives tinguin tallades un escut (Figura 41, Figura 42 i Figura 43) i les de velocitat tinguin tallades una figura amb línies d'aire al darrere la silueta (Figura 44, Figura 45 i Figura 46).

Per distingir les runes per raresa cal fixar-se en el color brillant d'aquesta. Si la pedra no brilla, sembla desgastada i quan es posa el ratolí a sobre té una aura blava, això vol dir que és comú (Figura 38, Figura 41 i Figura 44, part dreta). Si la pedra brilla una mica de color púrpura i té una aura també púrpura, això indica que és una runa èpica (Figura 39, Figura 42 i Figura 45, part dreta). Finalment, si la runa brilla d'un color daurat i l'aura és també daurada això vol dir que es tracta d'una runa llegendària (Figura 40, Figura 43 i Figura 46, part dreta).

A continuació es veuen les diferents rareses de les runes i les seves diferències dividides en el tipus principal de runa:

- Runes ofensives

Figura 38. Runa ofensiva comú

Figura 39. Runa ofensiva èpica

Figura 40. Runa ofensiva llegendària

- Runes defensives

Figura 41. Runa defensiva comú

Figura 42. Runa defensiva èpica

Figura 43. Runa defensiva llegendària

- Runes de velocitat

Figura 44. Runa de velocitat comú

Figura 45. Runa de velocitat èpica

Figura 46. Runa de velocitat llegendària

5.2.2.4 Mort del jugador

Quan la vida del jugador arriba a 0, la partida s'acaba i el protagonista mor. Quan això passa, es mostrarà una imatge amb el missatge de que s'ha mort abans de passar al menú principal. Veure Figura 47.

Figura 47. Imatge informant al jugador de que ha mort.

Es pot veure com el fons, encara que és la mateixa base que a la Figura 25 però més fosca per donar una sensació d'inquietud i nerviosisme al jugador.

5.2.2.5 Crèdits del final del joc

Un cop el jugador acaba amb Déu, sortirà una imatge final indicant al jugador que el joc s'ha acabat i un missatge amb els crèdits finals del joc. Veure Figura 48.

Figura 48. Imatge de crèdits finals

5.3 Mecàniques

D'entre les diverses mecàniques del joc, se'n poden destacar diversos grups. Hi ha mecàniques de moviment, de combat, de diàleg i d'obtenció i ús de materials.

5.3.1 Moviment

El moviment és un dels aspectes més importants en el joc, ja que és necessari que sigui molt còmode i fluït, per a donar al jugador un control altíssim i proporcionar una bona sensació. Els moviments interaccionen entre ells i es poden sobreposar uns als altres, per a poder proporcionar un alt control sobre els personatge. Les possibles accions són:

- Moviment horitzontal: es refereix al canvis de posició del jugador a través d'una direcció donada. Aquests són tan al terra com a l'aire.
- Salt: impuls vertical. Es pot donar sempre que el jugador toqui el terra.
- Doble salt: segon impuls vertical. Es pot fer sempre que el jugador estigui a l'aire i no s'hagi fet ja un Doble salt.
- Dash: impuls horitzontal donat en la direcció en la que mira el jugador. Es pot fer sempre que s'estigui al terra. Quan s'està a l'aire, es pot fer una primera vegada sense condicions, i després de cada Doble Salt.

5.3.2 Combat

El combat és un altre dels aspectes importants del joc. Aquest està molt relacionat al moviment, ja que està pensat per a poder fer cadenes complexes de cops. El combat es basa en cops inicials i cops encadenats. Els cops poden ser forts o febles, sent els forts més lents i provocant més mal, i els febles molt més suaus i ràpids. Sempre que no s'estigui atacant, es podrà fer un cop inicial. Aquest aspecte funciona igual que el *Dash*: es pot iniciar sempre que s'estigui al terra, quan s'estigui a l'aire i es faci per primera vegada, o després de cada doble salt. A partir del cop inicial, la resta de cops seran encadenats. La cadena es pot seguir fins que aquesta acabi. Existeixen moltes cadenes diferents, i van definides segons els cops febles i els forts. Les possibles accions són:

- Cop feble: cop feble i molt ràpid.

- Cop fort: cop fort i lent.
- Encadenar cop: es pot fer executant un cop de qualsevol tipus quan s'està executant un altre. Això seguirà una cadena de cops existent. Si la cadena actual no té l'opció de cop indicada, s'acabarà. Totes les cadenes tenen un fi.

5.3.3 Diàleg

El sistema de diàlegs és important per al jugador per a poder conèixer la historia, obtenir missions, obtenir informació interessant dins el joc o aconseguir objectes. En el sistema, el jugador podrà veure el què diuen els personatges, el què diu el protagonista automàticament i podrà donar respostes a escollir quan se li doni la opció. D'aquesta manera, es crearan cadenes de diàleg que siguin d'interès pel jugador. Les possibles accions són:

- Iniciar diàleg: iniciar el diàleg amb un personatge.
- Seguir diàleg: veure la següent seqüència del diàleg, ja sigui del personatge principal o de amb qui s'està dialogant.
- Respondre: escollir una resposta d'entre les diverses donades per a seguir un camí del diàleg o un altre.
- Acabar diàleg: acabar el diàleg actual.

5.3.4 Objectes

Els objectes són importants dins el joc, ja que poden tenir efectes positius en el jugador. Cal comentar que existeixen uns objectes anomenats Runes que permeten ser incrustats a l'espasa del jugador i atorguen un benefici mentre es porten incrustades. Les possibles accions són:

- Obrir cofre: hi ha cofres a través del mapa que es poden obrir. Aquests donen un objecte.
- Ús d'objecte: utilització d'un objecte per a obtenir un benefici, instantani, temporal o definitiu.
- Incrustació de runa: incrustar una runa a l'espasa.
- Desincrustació d'una runa: eliminar una runa de l'espasa.
- Llençar objecte: desfer-se d'un objecte en concret.

5.3.5 Comerç

El sistema de comerç és una de les vies que existeixen en el joc per obtenir els objectes esmentats a l'apartat anterior. Si bé les runes de major qualitat s'obtenen a través de cofres o missions especials, pel mapa es trobaran mercaders amb els quals es poden obtenir runes comuns i els objectes que donen extres a les estadístiques del jugador. Les possibles accions són les següents:

- Comprar: a través de les monedes que es guanyaran fent encàrrecs es poden obtenir els objectes que ofereix el venedor.
- Vendre: si s'aconsegueix un objecte del qual no se'n donarà ús, el jugador pot vendre'l per guanyar algunes monedes (el preu sempre serà inferior al preu que costa adquirir-lo).

5.4 Economia

Abans d'explicar com funciona l'economia, cal esmentar quatre conceptes que s'utilitzen per classificar els tipus de recursos del videojoc:

- *Sources*: creen recursos i els guarden en una entitat (creació).
- *Drains*: consumeixen recursos (drenatge).
- *Converters*: transformen recursos d'un tipus en un altre (conversió).
- *Traders*: mouen el recurs d'una entitat a una altra (intercanvi).

Pel que fa a la economia del joc, els recursos principals són les monedes, la sang, la vida, l'atac i la velocitat.

Les monedes són un *Source* ja que s'obtenen a través d'encàrrecs que donaran alguns NPC repartits pel mapa. Aquestes permeten obtenir els objectes a través del comerç, tal i com s'ha explicat a l'apartat anterior, per tant es podrien considerar un *Trader*.

La sang també és un *Source* s'obté cada cop que s'acaba amb la vida d'un enemic. Com més fort sigui l'enemic més sang oferirà. Aquesta permet pujar de nivell, cosa que es pot considerar un *Converter* ja que donarà al personatge una millora tant en la vida com en l'atac.

La vida pot ser un *Drain* i un *Source* al mateix temps, ja que pot ser restada per part dels atacs dels enemics, però es pot recuperar resant a les estàtues que es poden trobar pel mapa. A més, de mateixa manera que l'atac i la velocitat, es pot millorar notablement a través de la incrustació de runes a l'espasa, és a dir, que les runes també són un *Source*.

Finalment, una altra opció per tal de millorar les tres estadístiques (vida, atac i velocitat) seria a través de la utilització d'objectes (com a *Converter*), que donen un extra temporal. Aquesta economia es veu reflectida de forma esquemàtica a la Figura 49.

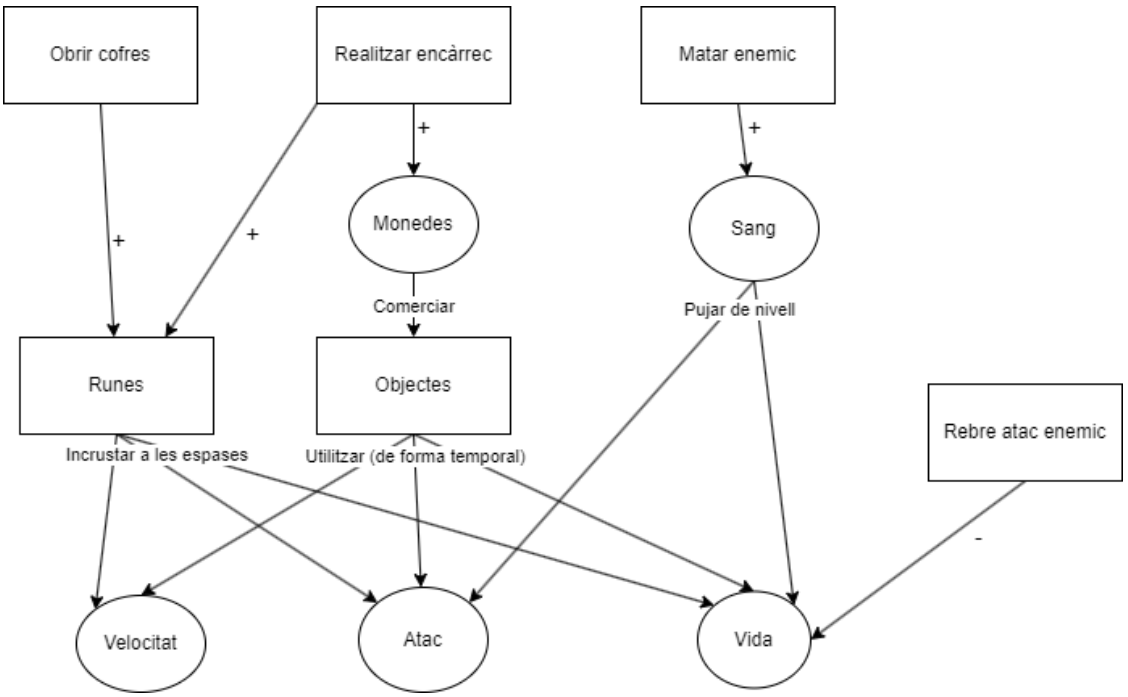

Figura 49. Esquema de l'economia

5.5 Nivells de Joc (Jerarquia de Reptes)

Existeixen 4 dificultats diferents dins el joc. Cada dificultat està associada a un moment del joc i escenari concret (els esmentats a l'Apartat 5.1.2). Tot i així, cada escenari té subzones en què la dificultat és major, a les quals s'hi pot accedir més endavant en el joc. A més, cada dificultat és progressiva. Aquesta divisió es fa per a tenir una referència, però la idea és que la dificultat sigui progressiva tot el joc.

Les dificultats assignades són:

- Infern: Nivell 1
- Món Terrenal: Nivell 2
- Cel: Nivell 3
- Combat Final: Nivell 4

El nivell de dificultat també indica les capacitats en moviment i ofensives que ha de tenir el jugador abans d'enfrontar-se a els diversos combats. Òbviament, cada dificultat tindrà unes recompenses millors, les quals milloren també de forma progressiva.

La idea, per tant, és fer que el jugador vagi adquirint experiència en un nivell en concret, i hagi d'assolir les parts més difícils d'aquell nivell abans d'accedir al següent escenari, on li esperarà un nou nivell de dificultat. A més, com ja s'ha dit, s'obligarà al jugador a anar de davant a darrera per a poder accedir a les zones més difícils de cada escenari.

5.6 Game Layout Charts

L'estructura general del joc al final ja s'ha comentat a l'apartat anterior. L'estructura que segueix el joc es pot entre millor visualment mirant la Figura 50.

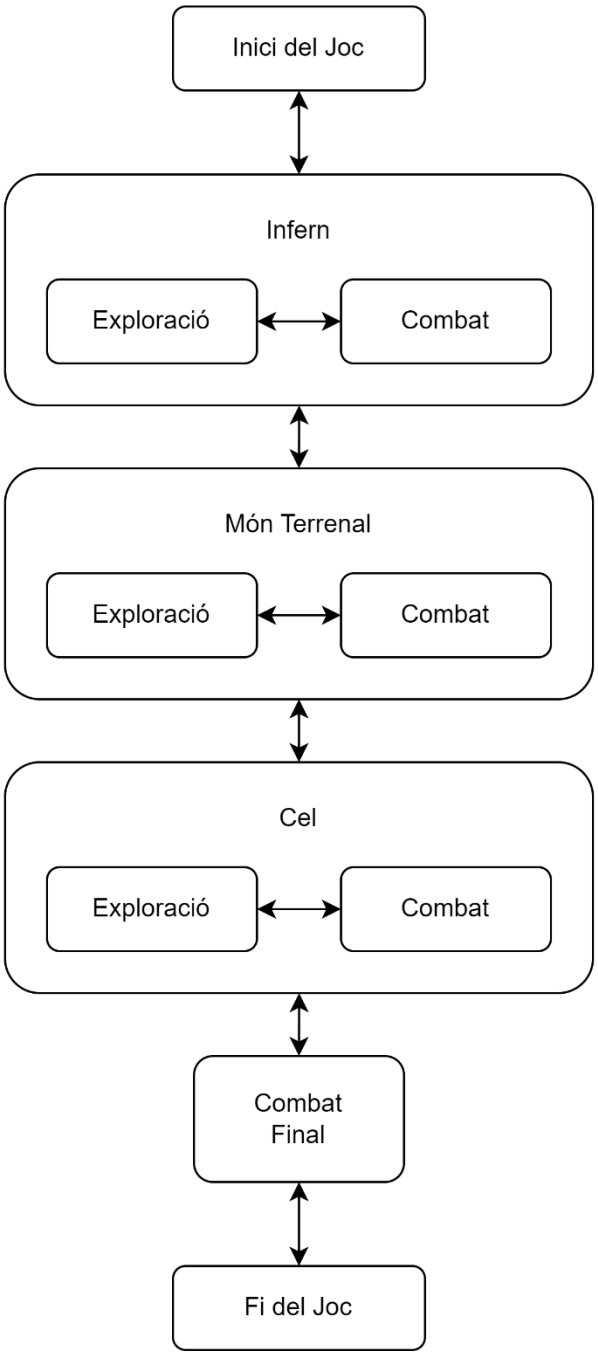

Figura 50. Game Layout Charts

5.7 Màrqueting

5.7.1 Plataformes de Distribució

El joc estarà dissenyat sobretot per a jugar des d'ordinador, però estarà disponible també a les consoles estàndard del mercat (*Play Station, Xbox* i *Switch*).

5.7.2 Comunicació

L'objectiu del màrqueting seria promocionar el videojoc en el mercat, arribant a un alt nivell de reconeixement i generant ventes.

Per aconseguir-ho, es crearia publicitat en línia. Primer de tot, es crearien anuncis en vídeo per a destacar la jugabilitat i característiques úniques del joc, relacionades amb el seu gènere. S'utilitzarien plataformes digitals de publicitat i xarxes socials com *Instagram* i *Twitter* per arribar al jugador objectiu, mostrant-li justament les característiques que li poguessin interessar. Com a afegit, es buscarien creadors de contingut famosos a plataformes digitals en el gènere del joc. Es podrien enviar còpies del joc de forma anticipada per a generar expectativa en els futurs jugadors.

Pel què fa a les relacions amb mitjans de comunicació, als medis especialitzats, se'ls hi enviaria material exclusiu, a més de fer esdeveniments i demostracions privades, per tal de crear notícies en portals famosos i especialitzats que afavorissin la publicitat del joc. Als medis generals, se'ls hi enviaria informació més general per arribar als medis de comunicació tradicionals. Es podrien ressaltar aspectes interessants del joc per atraure un públic més ampli, com aspectes narratius o visuals. Tot i així, aquests tindrien una menor influència en la publicitat.

6 Implementació

6.1 Inventari

Per a crear un inventari, es necessiten, primer de tot, un tipus de dades que es consideri un objecte, i una estructura de dades composada per aquest "objecte".

6.1.1 ST_Item

Aquest component representa el conjunt de dades que composa un objecte. Es tracta d'un *Blueprint* anomenat *Structure*. En ell, es creen diverses variables i se'ls hi assigna un tipus de valor. Aquesta llista es pot veure a la Figura 51.

Figura 51. Variables de l'estructura ST_Item

La major part de les variables són tipus ja existents a UE4, però n'hi ha tres que no ho són, tots tres enumeradors, o *Enumeration* a UE4: *EN_Item_Types*, *EN_Item_Rarity_Types* i *EN_Element*.

Als *Enumeration*, s'han de crear els tipus de valor assignables a les variables. A aquests, se'ls hi ha de posar un nom i, opcionalment, una descripció. Els possibles valors de cadascun d'ells són els de les figures: Figura 52, Figura 53 i Figura 54.

Figura 52. EN_Item_Types

Figura 53. EN_Item_Rarity_Types

Figura 54. EN_ Element

6.1.2 DT_Items

A partir de l'*Structure* acabat d'explicar, es crea el conjunt de dades que es volen tenir. Per a aquest propòsit, s'utilitza un tipus de *Blueprint* anomenat *Data Table*. Explicat de forma ràpida i resumida, es tracta d'una taula on els elements són d'un tipus *Structure* concret. És a dir, que les fileres seran sempre un objecte *Structure* escollit a l'hora de crear la *Data Table*. Aquesta taula consistirà en: un índex, un nom únic identificatiu, i tota la informació determinada a l'objecte *ST_Item*.

Per a crear les runes, s'ha fet una divisió inicial segons l'*EN_Item_Rarity_Types*, veure Figura 53. Com bé s'ha esmentat anteriorment, les runes comuns són les que tenen pitjor qualitat i les llegendàries són les millors. Això s'ha de veure reflectit a les estadístiques. Cada runa donarà una millora a, com a mínim, una estadística (augmentar la vida màxima, l'atac o la velocitat).

6.1.2.1 Runes comuns

En aquest cas hi ha tres tipus, una per cada estadística que es pot millorar. Com es pot apreciar a la Figura 55, només tenen una estadística i els valors que dona són baixos en comparació a les altres.

Attack Rune I	Rune that adds power to a sword by embedding it in it	Attack	25.000000	None	0.000000	Common
Defense Rune I	Rune that adds defense to a sword by embedding it in it	Defense	25.000000	None	0.000000	Common
Velocity Rune I	Rune that adds velocity to a sword by embedding it in it	Velocity	31.250000	None	0.000000	Common

Figura 55. Estadístiques de les runes comuns

6.1.2.2 Runes èpiques

Per aquest cas, la gran majoria aporten dues característiques (menys les que aporten el doble d'una) i es pot observar una gran millora respecte a les anteriors. Tant les estadístiques d'atac com de defensa es dupliquen i la velocitat es triplica. Tot i així, per obtenir un millor balanceig, s'ha fet que el tipus secundari aporti un valor inferior al primari (tot i que continua sent superior a les estadístiques de les comuns). Veure Figura 56.

Attack Defense Rune II	Rune that adds power and defense to a sword by embedding it in it	Attack	50.000000	Defense	30.000000	Epic
Double Attack Rune II	Rune that adds huge power to a sword by embedding it in it	Attack	100.000000	None	0.000000	Epic
Attack Velocity Rune II	Rune that adds power and velocity to a sword by embedding it in it	Attack	50.000000	Velocity	93.750000	Epic
Double Defense Rune II	Rune that adds huge defense to a sword by embedding it in it	Defense	100.000000	None	0.000000	Epic
Defense Velocity Rune II	Rune that adds defense and velocity to a sword by embedding it in it	Defense	50.000000	Velocity	93.750000	Epic
Double Velocity Rune II	Rune that adds huge velocity to a sword by embedding it in it	Velocity	187.500000	None	0.000000	Epic

Figura 56. Estadístiques de les runes èpiques

6.1.2.3 *Runes llegendàries*

Finalment, aquest es el llistat de les millors runes del joc. Aquestes són exclusives, per això només n'hi haurà una de cada tipus al mapa. Per reafirmar la idea d'exclusivitat i qualitat, tal i com es pot observar, aquestes runes no tenen un nom segons la seva estadística, si no que utilitzen un nom propi i una descripció que s'allunya de la genèrica. Quant a les estadístiques, comparat amb les comuns, aquestes aporten cinc vegades més velocitat i el triple de defensa i atac (excepte amb el tipus secundari, que és lleugerament inferior).

A més, cada runa té un element (foc, electricitat o crític). Aquesta part no s'ha implementat de cares al prototip, però s'utilitzarà al joc final per crear el sistema de debilitats i fortaleses dels enemics. Veure Figura 57.

Fire Fang	Powerful and savage	Attack	75.000000	Defense	70.000000	Legendary	Fire
Thunder Slash	Anyone is able to see it	Attack	75.000000	Velocity	125.00000(	Legendary	Electric
Heavy Stab	Induced by an overwhelming force	Defense	75.000000	Attack	70.000000	Legendary	Critic
Flame Determination	Exalt your inner blaze	Defense	75.000000	Velocity	125.00000(	Legendary	Fire
Lightning Shock	Match the speed of light	Velocity	156.250000	Defense	70.000000	Legendary	Electric
Wild Heart	Feel the power of instinct	Velocity	156.250000	Attack	70.000000	Legendary	Critic

Figura 57. Estadístiques de les runes llegendàries

6.1.3 GI_Inventory

Amb tota la informació de les runes ja creada, ara ja es pot començar a implementar el que veritablement serà l'inventari.

Primer de tot, cal crear una instància accessible des de tot arreu que contingui la informació de les runes. Aquesta és el *GI_Inventory*, un Blueprint que hereta de *Game Instance* i que es pot assignar com a *Game Instance Class* a les propietats del projecte.

Els atributs del *GI_Inventory* són els de la Figura 58.

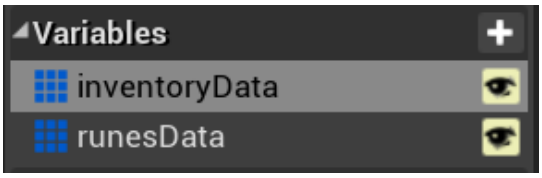

Figura 58. Atributs del GI_Inventory

L'atribut que ens interessa ara és *inventoryData*, que es tracta d'una llista d'*ST_Item*. *runesData* té la mateixa estructura, però ara mateix no ens interessa.

6.1.4 InventoryScroll

El següent que cal crear és l'element visual per a la pantalla. El necessitem per a poder fer tota la lògica d'assignació, ordenació i eliminació de les diverses runes. Aquí entrarem només en la estructura seguida, la qual es pot veure a la Figura 59. La part de l'estètica s'explica a l'Apartat 5.2.

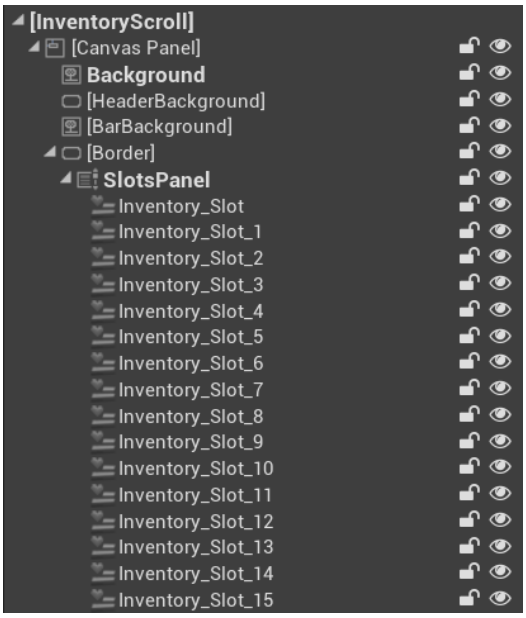

Figura 59. Estructura de l'InventoryScroll

L'estructura de components és senzilla. Per començar, té l'objecte pare i un *Canvas Panel*, que és obligatori per poder afegir la resta d'elements. Després té les diverses imatges de fons, i el *Border*, que marca a quin lloc de la pantalla estarà exactament situada la part on es veuran les diferents runes. *Inventory_Slot* són cadascun dels espais que hi ha per a les runes, i és un *Widget* diferent, amb l'estructura de la Figura 60.

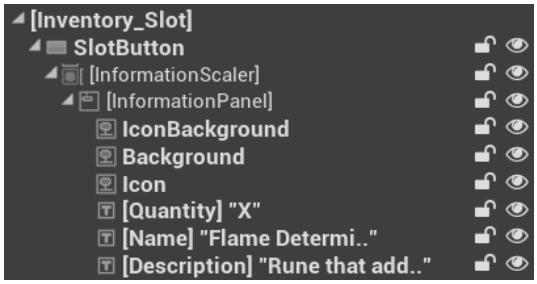

Figura 60. Estructura de l'Inventory_Slot

De nou, l'estructura comença amb el pare. Després fem que sigui tot l'element un botó (*SlotButton*), i li donem mida amb l'*InfomationScaler*. Després li posem l'*InformationPanel* (que és un *CanvasPanel* amb el nom canviat), per poder afegir les imatges i els texts que hi ha a continuació.

6.1.5 MainHUD

Abans d'explicar la implementació de la lògica, cal mencionar un altre *Widget* anomenat *MainHUD*. Aquest està compost simplement per un *CanvasPanel* buit. S'assigna a la pantalla quan s'inicia la partida, i serà el pare de tot l'*InventoryScroll*. Veure Figura 61.

Figura 61. Inicialització del MainHUD

6.1.6 BPFL_InventoryScroll

Amb tota la infraestructura creada, ja es pot, per fi, començar a implementar la lògica. S'explicarà funció per funció: primer les funcions útils que s'utilitzaran en alguns moments en altres llocs i que es poden explicar de forma independent, i després s'explicaran les funcions que donen funcionalitat a l'inventari.

6.1.6.1 GetInventoryGameInstance

GetInventoryGameInstance fa exactament el que diu, obté el *GI_Inventory* explicat abans. Veure Figura 62.

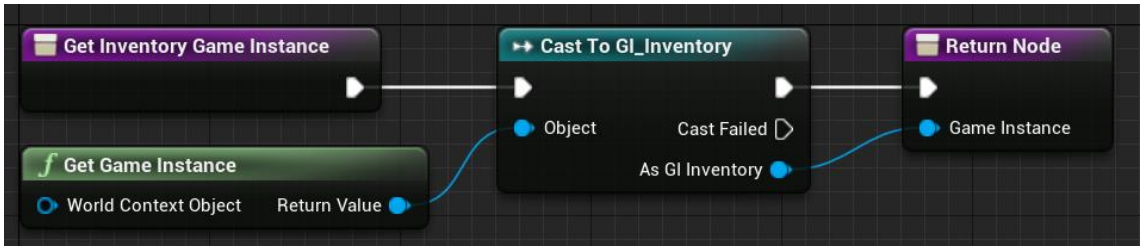

Figura 62. GetInventoryGameInstance

6.1.6.2 GetHUD

Aquesta funció obté el *MainHUD* explicat abans. Veure Figura 63.

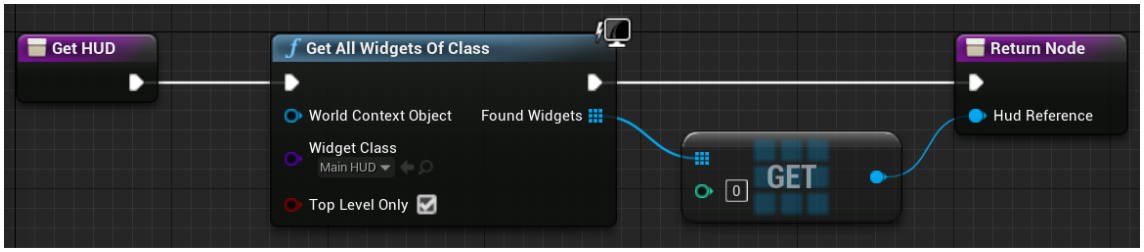

Figura 63. GetHUD

6.1.6.3 GetInventoryReference

Aquesta funció obté l'*InventoryScroll* explicat abans. Veure Figura 64.

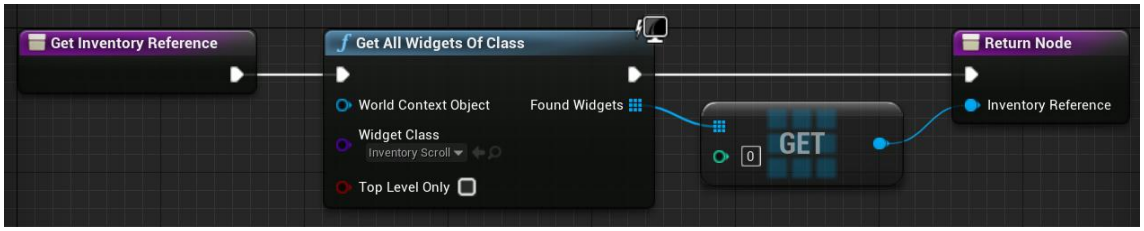

Figura 64. GetInventoryReference

6.1.6.4 *IsStringAlphabeticallyFirst*

Aquesta funció ordena alfabèticament i per mida del text. Veure Figura 65 i Figura 66.

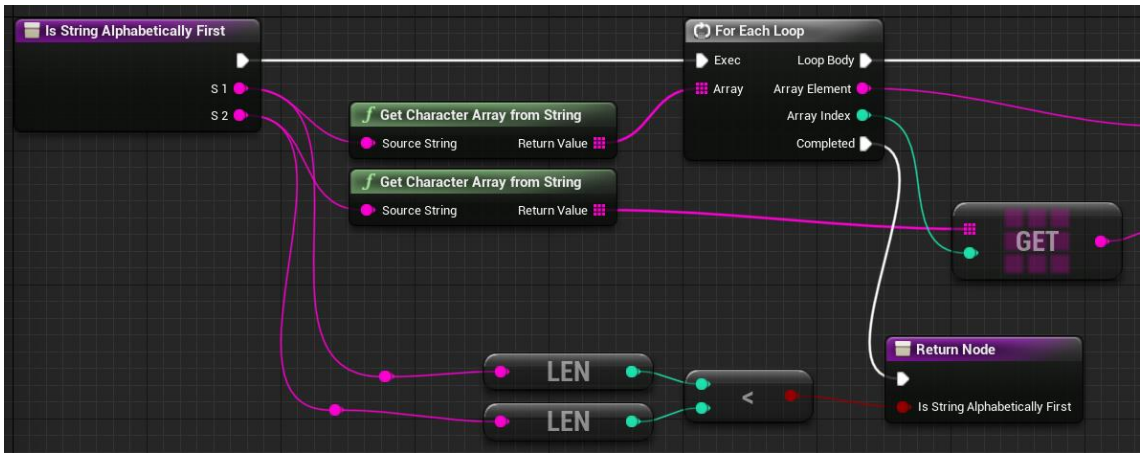

Figura 65. IsStringAlphabeticallyFirst - Part 1

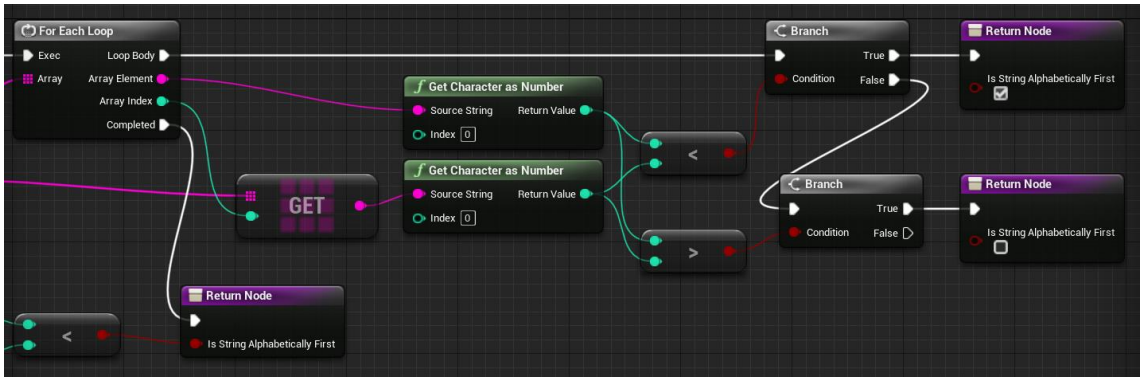

Figura 66. IsStringAlphabeticallyFirst - Part 2

Primer hi ha una comparació de caràcters per posició, i l'*script* diu si la primera és primera alfabèticament. Si es dona la casualitat que una cadena de caràcters conté l'altra completament, retorna que la més curta va primera.

6.1.6.5 *InitializeInventory*

Aquesta funció inicialitza tot l'inventari. Veure Figura 67.

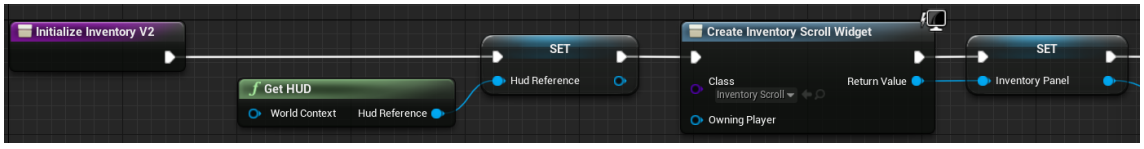

Figura 67. InitializeInventory - Part 1

Primer de tot, agafa les referències del HUD i de l'*InventoryScroll* i les guarda com a variables locals. Veure Figura 68.

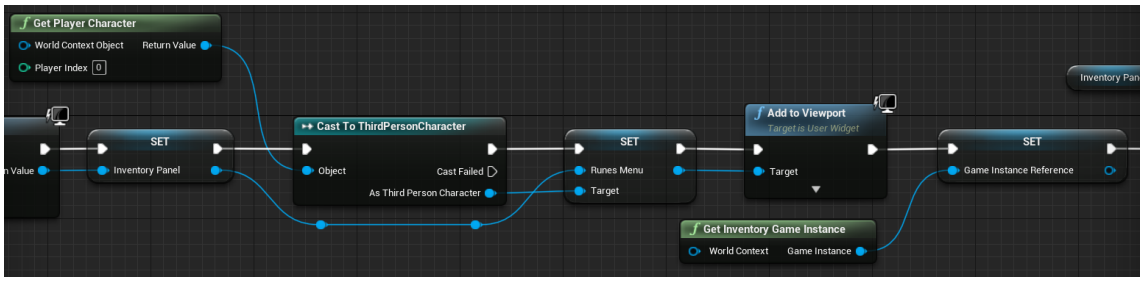

Figura 68. InitializeInventory - Part 2

Just després li assigna al jugador l'inventari a l'atribut *RunesMenu*, afegeix l'inventari a la pantalla, i guarda com a variable local una referencia al *GI_Invetory*. Veure Figura 69.

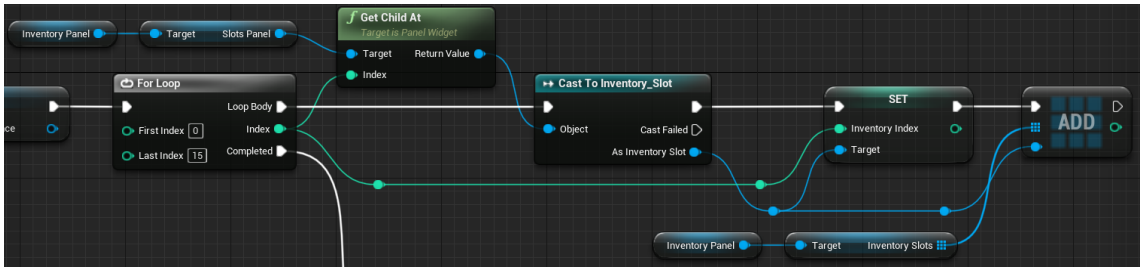

Figura 69. InitializeInventory - Part 3

A continuació es fa un recorregut per obtenir tots els *Inventory_Slot* explicats abans per assigna'ls-hi l'índex dins l'inventari i guardar les referències. Veure Figura 70.

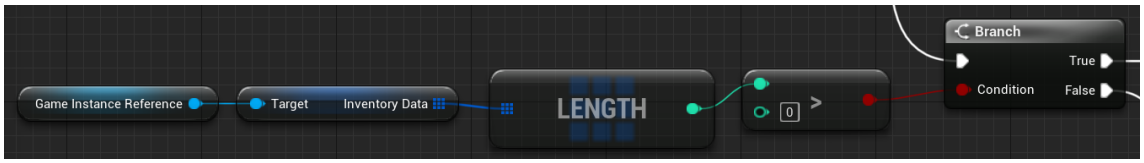

Figura 70. InitializeInventory - Part 4

A l'acabar, es creen dos camins segons si hi ha informació guardada de l'inventari. Veure Figura 71.

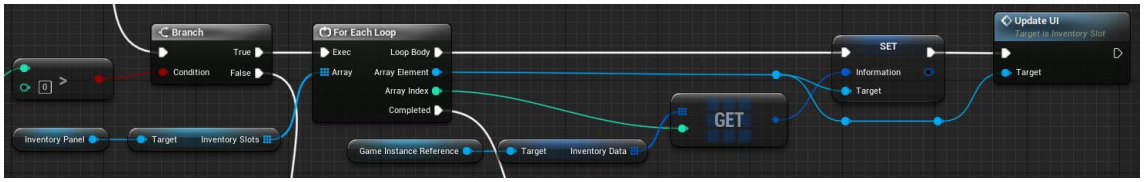

Figura 71. InitializeInventory - Part 5

En cas d'haver-hi, s'assigna i s'actualitza l'apartat visual amb la funció *UpdateUI* (aquesta s'explicarà més endavant). Veure Figura 72.

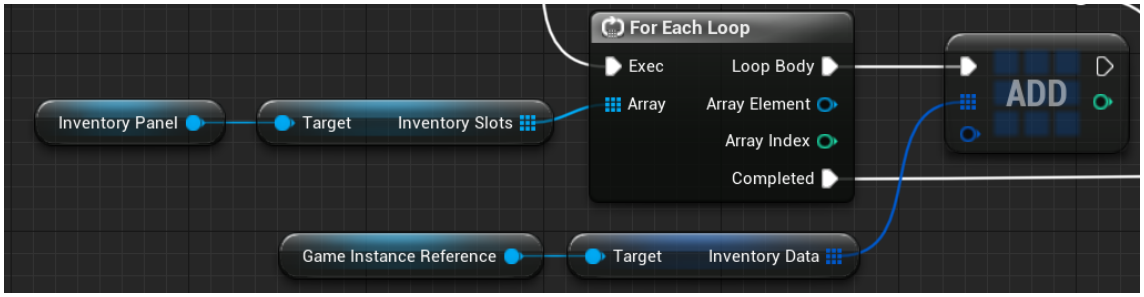

Figura 72. InitializeInventory - Part 6

En cas de no existir informació, es crea amb elements buits. Veure Figura 73.

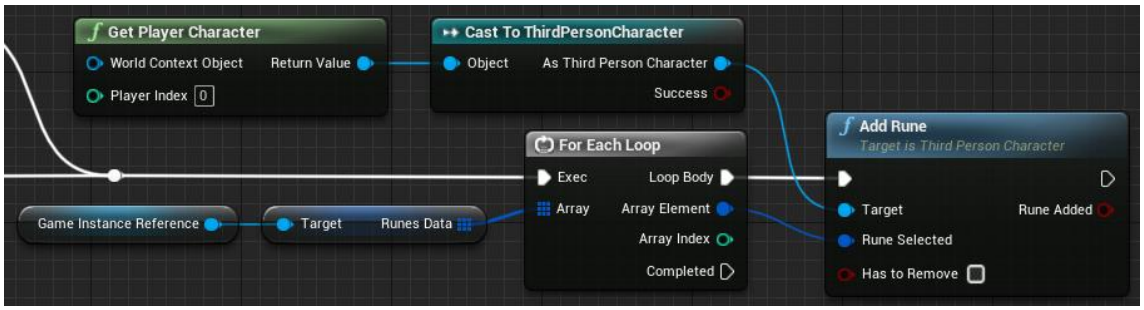

Figura 73. InitializeInventory - Part 7

En qualsevol dels dos casos, a l'acabar, s'afegeixen les runes incrustades que estaven guardades (aquest pas s'entendrà millor quan s'expliqui la part de la incrustació de runes), a l'Apartat 6.1.7.

6.1.6.6 ScanForItem

Aquesta funció busca i retorna la informació d'un objecte dins l'inventari a partir del seu nom. Si no existeix, retorna *FoundItem* a fals. Veure Figura 74.

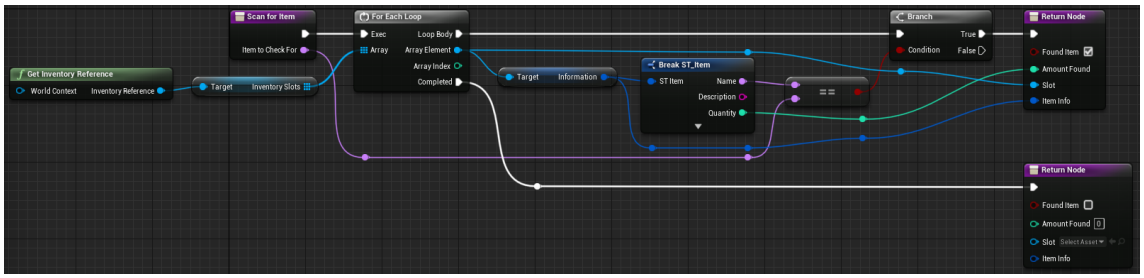

Figura 74. ScanForItem

6.1.6.7 UpdateQuantity

Aquesta funció actualitza la quantitat d'un objecte de l'inventari segons el valor *QuantityToAdd*, que pot ser tan positiu, per afegir més de l'objecte, com negatiu, per extreure'n. Veure Figura 75.

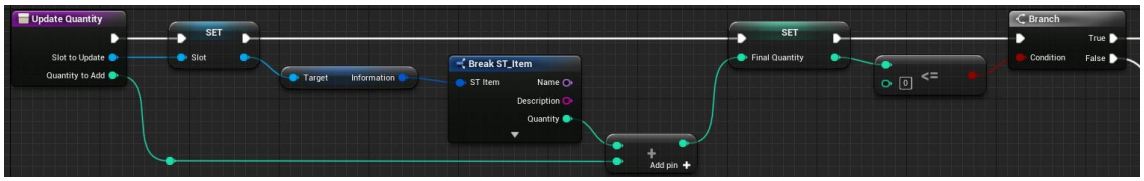

Figura 75. UpdateQuantity - Part 1

El primer que es fa és obtenir la quantitat de l'objecte dit, sumar-li el *QuantityToAdd*, i guardar-lo a una variable local anomenada *FinalQuantity*. A més, es segueixen camins diferents segons si aquest és major a 0, o no. Veure Figura 76.

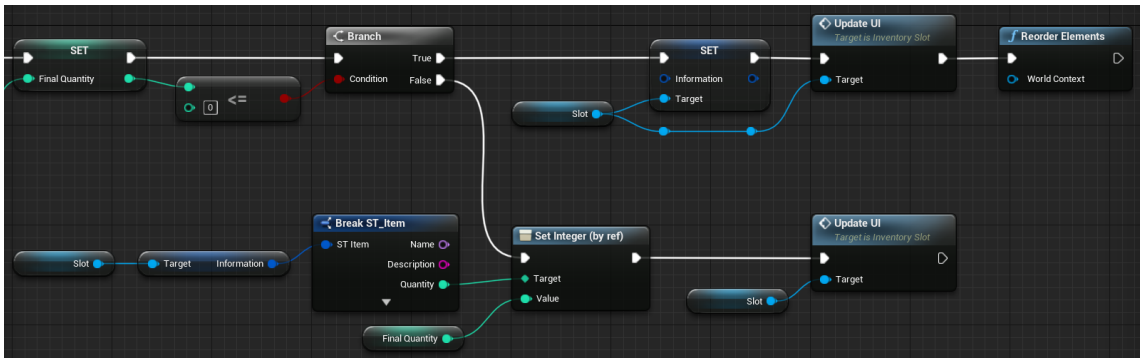

Figura 76. UpdateQuantity - Part 2

En cas de ser major a 0, actualitza la quantitat a l'apartat visual. En cas de no ser-ho, elimina l'element i reordena l'inventari amb la funció *ReorderElements* (la qual s'explicarà a continuació) per eliminar el buit que queda a l'eliminar.

6.1.6.8 ReorderElements

Aquesta funció serveix per reordenar els elements de l'inventari i eliminar el buit que deixa l'eliminació d'un element explicat a la funció anterior. Veure Figura 77.

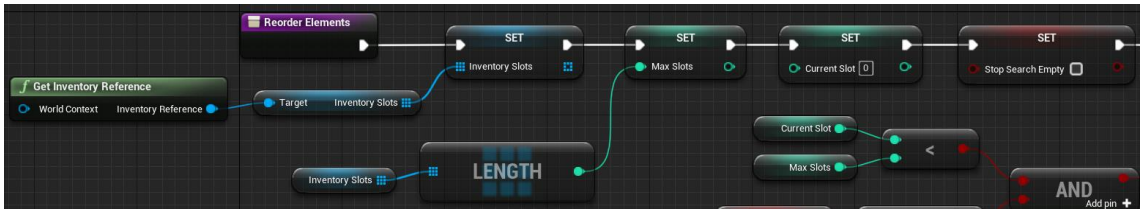

Figura 77. ReorderElements - Part 1

El primer que es fa és guardar en variables locals la referencia a l'inventari, la quantitat d'*Inventory_Slot* a l'inventari, l'índex actual a 0, i el parar la cerca a fals. Veure Figura 78.

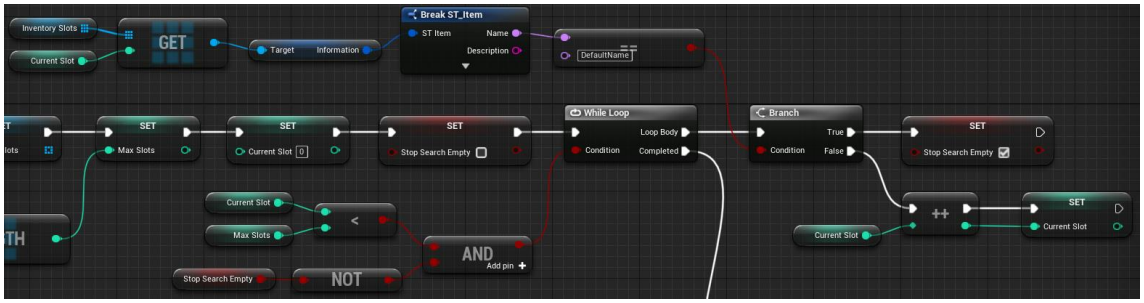

Figura 78. ReorderElements - Part 2

A partir d'aquí, es fa un bucle per buscar l'element de l'inventari que ha quedat buit, i deixar el *CurrentSlot* amb l'índex. Amb la condició *Name == DefaultName*, es determina si l'*Inventory_Slot* està buit. Veure Figura 79.

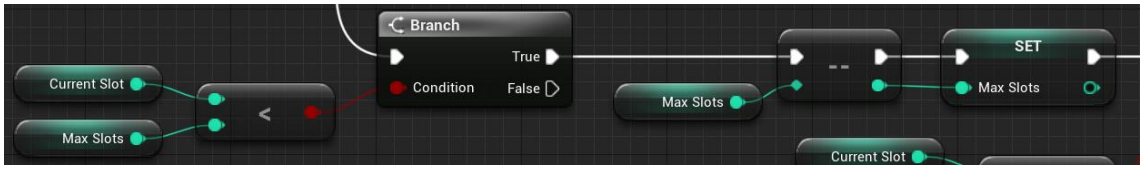

Figura 79. ReorderElements - Part 3

Es comprova si no s'ha quedat fora de la llista, perquè si és així, no cal fer res. Si segueix, resta 1 al valor de *MaxSlots* (no perquè disminueixi la quantitat d'*slots*, sinó per conveniència del codi). Veure Figura 80.

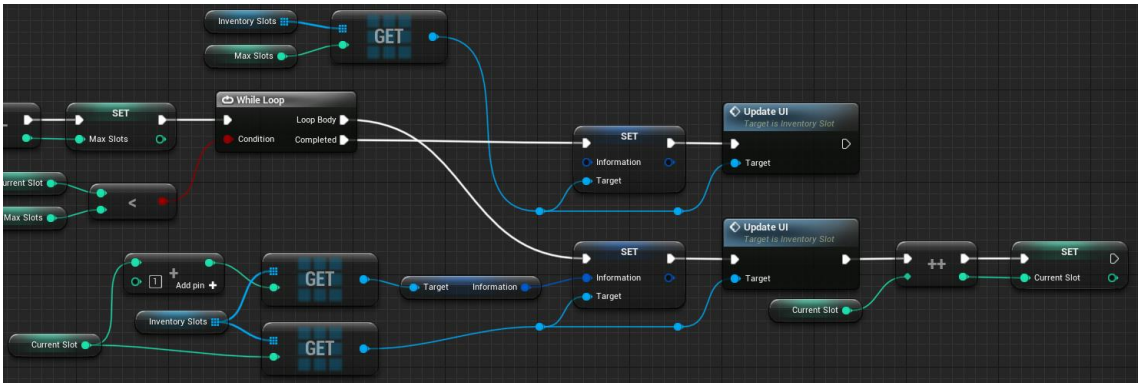

Figura 80. ReorderElements - Part 4

Per acabar, es fa un bucle per col·locar la informació del següent *slot* en l'actual. Quan el bucle acaba, s'elimina la informació de l'últim *slot*, ja que s'ha guardat a l'*slot* anterior.

6.1.6.9 InsertItem

Aquesta funció insereix una nova runa dins l'inventari. Si ja hi existeix una d'igual, la busca i li augmenta la quantitat, si no, la insereix de forma ordenada.

Aquesta funció està preparada per retornar si l'ha inserit o no, a causa de la mida de l'inventari. Tot i que aquest cas no es pot donar en la nostra *demo*, podria passar perfectament, que, si es té l'inventari al màxim de capacitat, no s'aconsegueixi inserir la runa i, per tant, no es pugui agafar, fent s'hagi de desfer-se d'alguna que ja es porti a sobre. Tot això no està implementat dins el joc, però és una possibilitat que aquesta funció permet.

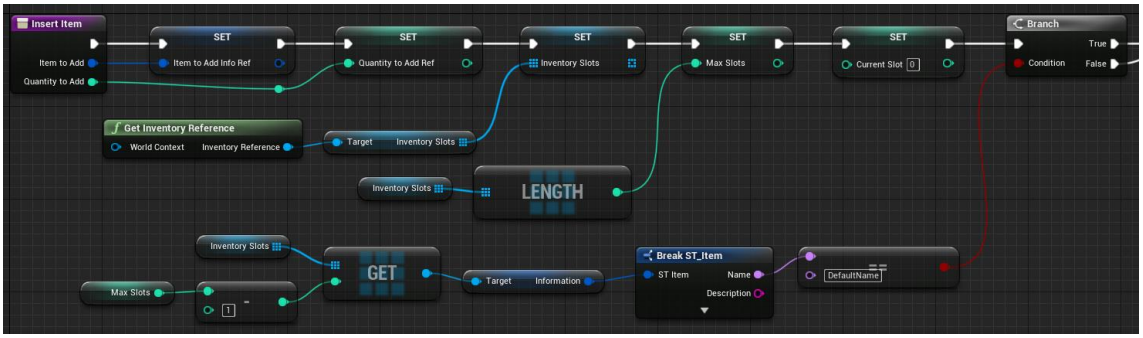

Figura 81. InsertItem - Part 1

Primer de tot, es guarden unes quantes referencies per a un accés més fàcil: l'objecte a afegir, la quantitat a afegir, la llista d'*Inventory_Slot*, la mida de la llista, i l'índex de l'*slot* actual, que és 0. A més es creen dos camins segons aquesta condició. Aquesta condició simplement diu si hi ha espai per a una runa nova, ja que si no n'hi ha, l'única opció és actualitzar la quantitat d'alguna ja existent.

Comencem pel cas en què no hi ha més espai. Veure Figura 82.

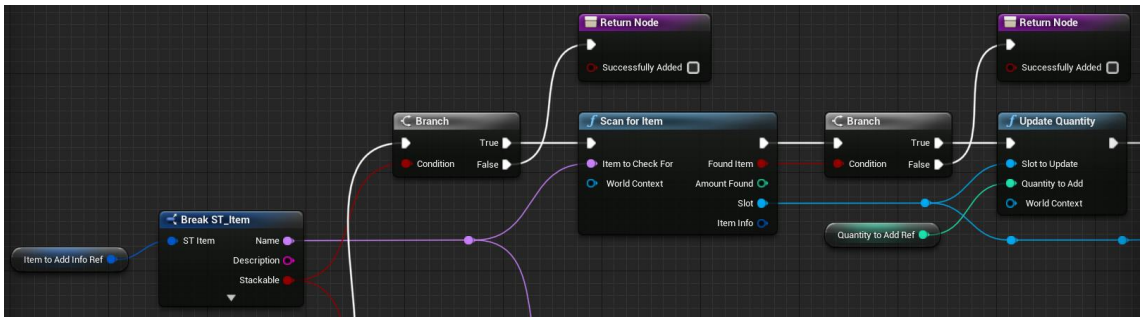

Figura 82. InsertItem - Part 2

El primer que es mira és si els objectes es poden *stackejar* (acumular, a partir d'ara). Si no és així, ja es termina el procés, retornant que no s'ha pogut guardar. Si sí que es poden acumular, es mira si ja existeix aquest objecte dins l'inventari amb la funció *ScanForItem*. De nou, si no és possible, es termina el procés retornant que no s'ha pogut guardar. Veure Figura 83.

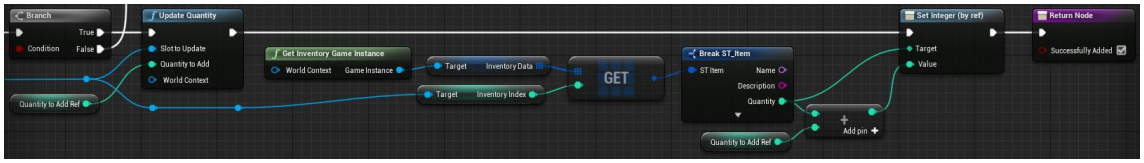

Figura 83. InsertItem - Part 3

En cas que sí existeixi ja un objecte igual, s'actualitza la quantitat a l'apartat visual amb la funció *UpdateQuantity* i a l'estructura de dades amb les següents instruccions. Es retorna que sí s'ha pogut guardar.

Continuem ara pel cas en què si que hi ha espai. Veure Figura 84.

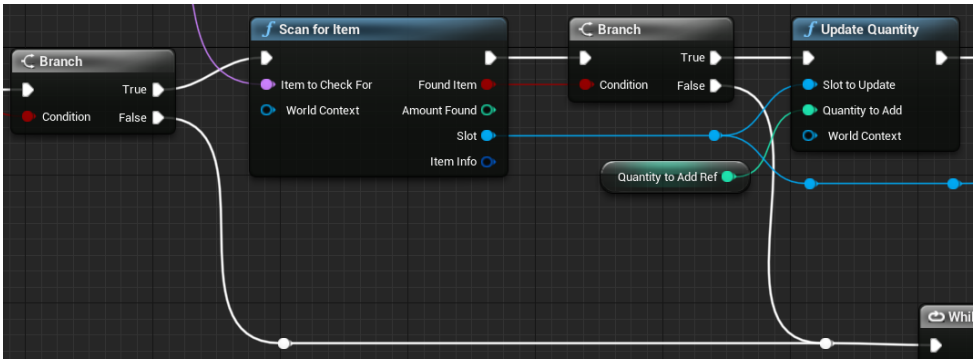

Figura 84. InsertItem - Part 4

De la mateixa forma que abans, es comprova primer si són acumulables. Si ho són, es busca si ja hi ha un objecte igual. Si existeix, es fa exactament el mateix procés que s'acaba d'explicar. En cas que alguna de les dues condicions sigui falsa, s'uneixen els camins i segueix el procés. Veure Figura 85 i Figura 86.

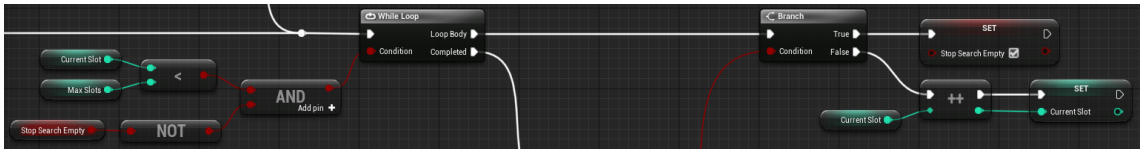

Figura 85. InsertItem - Part 5

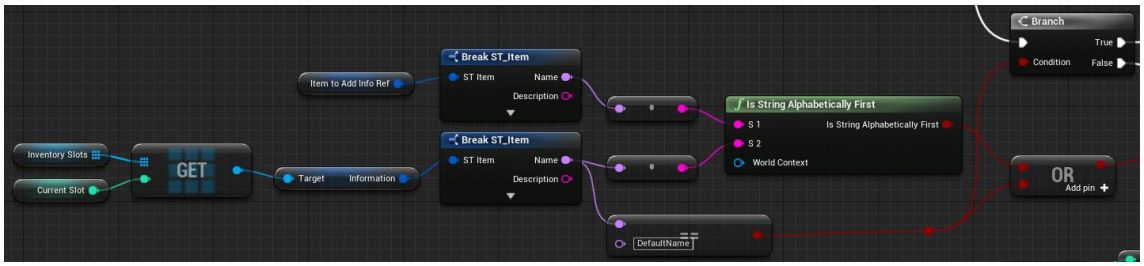

Figura 86. InsertItem - Part 6

El següent que es fa és un bucle per buscar el lloc en què caldrà inserir l'objecte. La cerca deixa el valor de *currentSlot* exactament en el lloc en què caldrà inserir. Les condicions per aturar la cerca són que el *currentSlot* no passi la mida de la llista o que s'hagi trobat el lloc on caldrà posar el nou objecte. Aquesta segona part es fa amb el conjunt d'instruccions de la segona imatge (sense el *Branch*). Ara que ja tenim el lloc on col·locar l'objecte, continuarem per dos camins diferents, segons si a l'acabar es compleix la condició dita (ara cal tenir en compte el *Branch*). Veure Figura 87 i Figura 88.

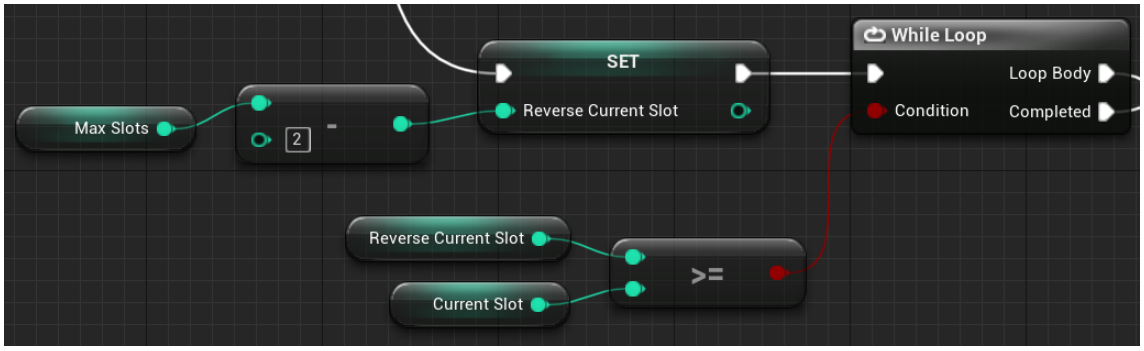

Figura 87. InsertItem - Part 7

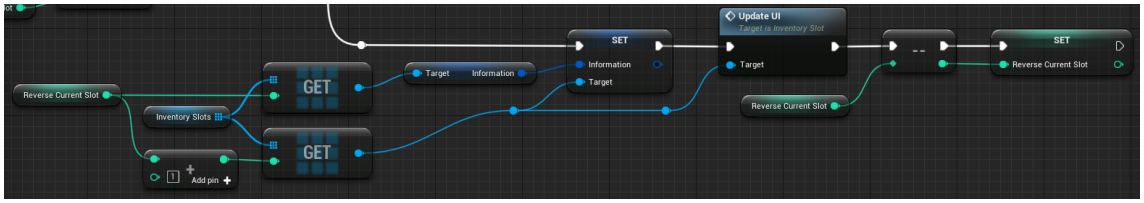

Figura 88. InsertItem - Part 8

En cas de no complir la condició, caldrà fer un bucle que mourà tots els objectes que vagin després del nou, una posició cap a baix. Tot això es fa amb el codi de les figures Figura 87 i Figura 88.

La continuació del procés quan acaba aquest bucle, o en cas de sí complir la condició anterior, és exactament la mateixa. Veure Figura 89 i Figura 90.

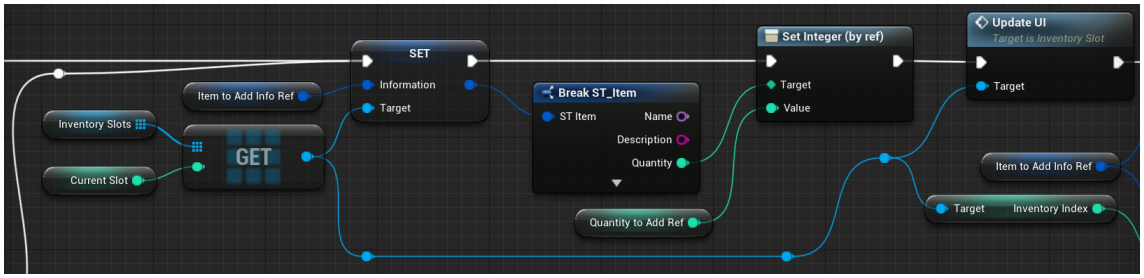

Figura 89. InsertItem - Part 9

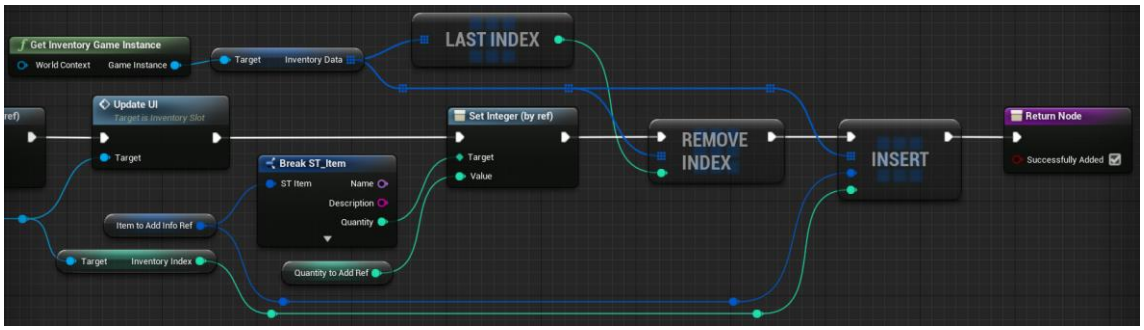

Figura 90. InsertItem - Part 10

El què fa aquesta última part és actualitzar la informació de l'apartat visual (Figura 89) i de l'estructura de dades (Figura 90) amb el nou objecte i la quantitat determinada. També retorna que sí ha pogut inserir l'objecte.

6.1.6.10 RemoveItem

Aquesta funció elimina un element de l'inventari en una quantitat concreta. Si la quantitat restant és igual o inferior a 0, l'elimina completament. Veure Figura 91.

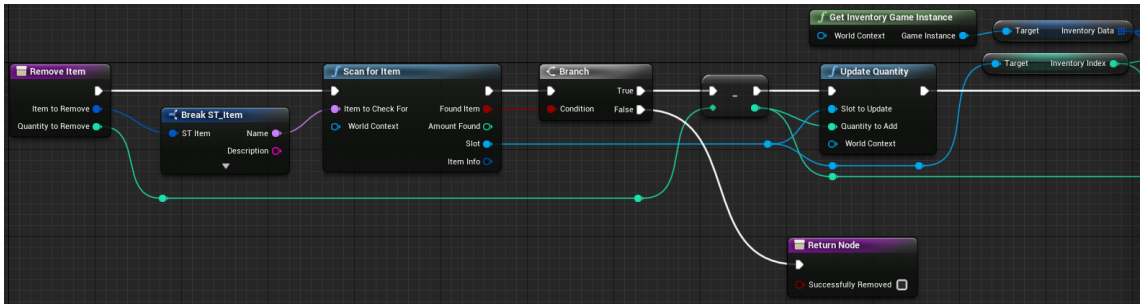

Figura 91. RemoveItem - Part 1

Primer de tot, busca l'element a l'inventari, perquè si no existeix, ja no cal fer res. Si existeix l'element, se li resta la quantitat indicada, i s'actualitza l'apartat visual. En cas que s'eliminin tots els objectes restants d'aquell tipus, simplement l'elimina completament de l'inventari (d'això s'encarrega l'*UpdadeQuantity*, com ja s'ha explicat abans). Veure Figura 92.

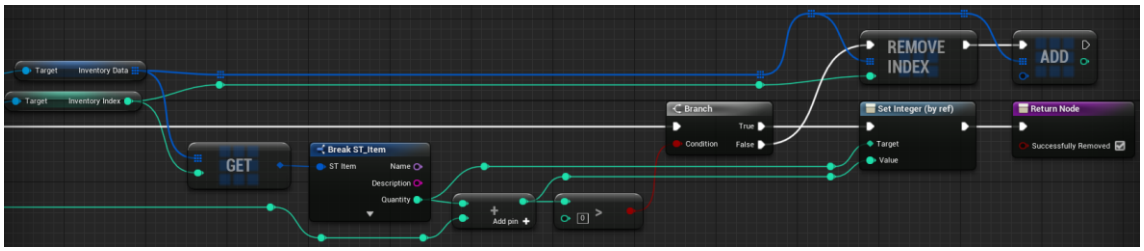

Figura 92. RemoveItem - Part 2

Just després, s'actualitza també a l'estructura de dades. De la mateixa manera, actualitza el valor o elimina l'objecte, segons si la quantitat restant és major a 0 o no.

6.1.7 Incrustació de runes

Per a implementar aquesta part intervenen tres elements: el *ThirdPersonCharacter*, l'*Inventory_Scroll* i els *Inventory_Slot*, els quals es troben dins del segon *Blueprint*, esmentat a l'Apartat 6.1.7.2.

6.1.7.1 Lògica de les runes dins el ThirdPersonCharacter

Els atributs del personatge per tal de poder guardar les runes són els mostrats a la Figura 93.

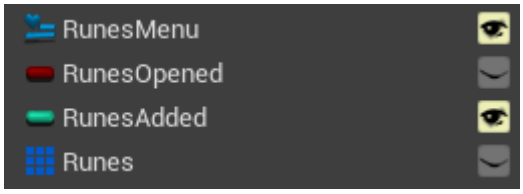

Figura 93. Atributs de les runes al ThirdPersonCharacter

El primer és una referència a l'*Inventory_Scroll* que permet obrir i tancar l'inventari, el segon un booleà que indica si està obert, el tercer indica quantes runes té equipades actualment i l'últim és una llista d'*ST_Item*, on realment es guardaran les runes. Cal dir que *Runes* s'inicialitza afegint tantes runes buides com es desitgi (en aquest cas tres).

6.1.7.1.1 Obrir i tancar inventari

Per realitzar aquesta acció, es recull un *Event* quan el jugador pressiona el botó per obrir o tancar l'inventari, tal i com es pot apreciar a la Figura 94. S'utilitza un *InputAction*, explicat a l'Apartat per poder recollir tant teclat com controlador. Per triar si obrir o tancar, es fa servir el node *FlipFlop* que alternarà sempre entre A i B.

Pel cas A (obrir inventari), primer comprova que s'ha creat l'inventari (tot i que no hauria de donar problemes ja que es fa al principi) i executa la funció per obrir-lo. El cas B és el contrari, ja que utilitza una altra funció per tancar-lo.

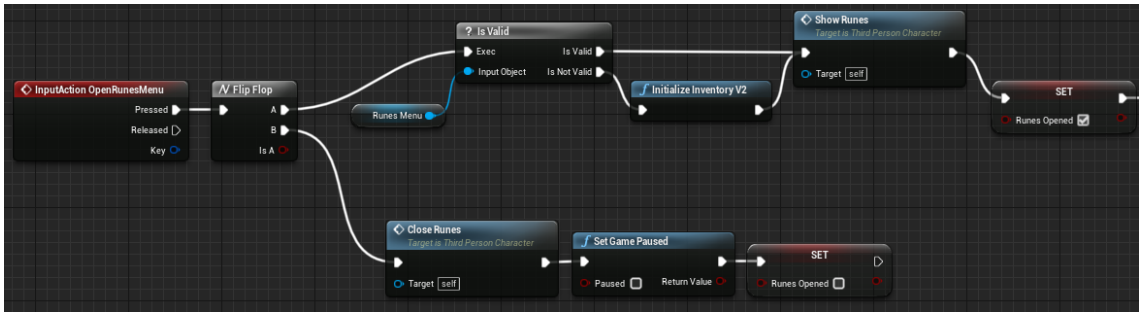

Figura 94. Obrir inventari - Part 1

La Figura 95 mostra una petita peculiaritat de la part d'obrir. Pel cas del controlador cal esperar una mica i executar les funcions de navegació per assegurar que sempre que s'obre l'inventari està seleccionat el primer *Inventory_Slot*. Les funcions de navegació es troben explicades als Apartats 6.1.7.2.8 i 6.1.7.2.1.

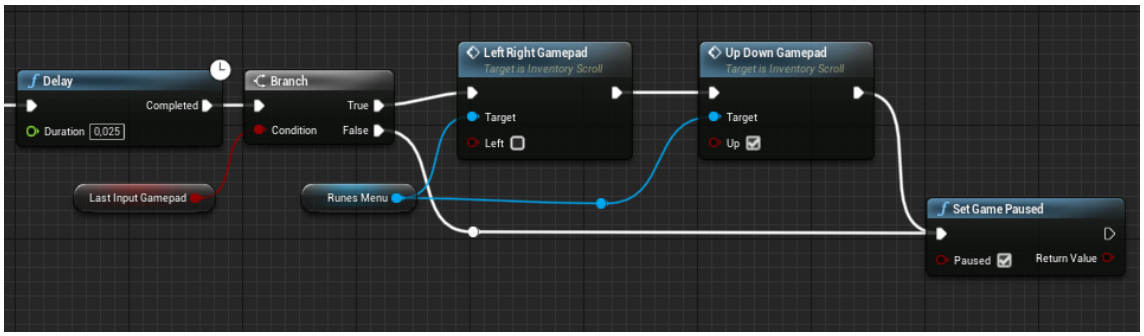

Figura 95. Obrir inventari - Part 2

6.1.7.1.1.1 Show Runes

Aquesta funció mostra l'inventari i el cursor (només si el jugador està utilitzant teclat i ratolí) i activa els inputs per la interfície d'usuari. Veure Figura 96.

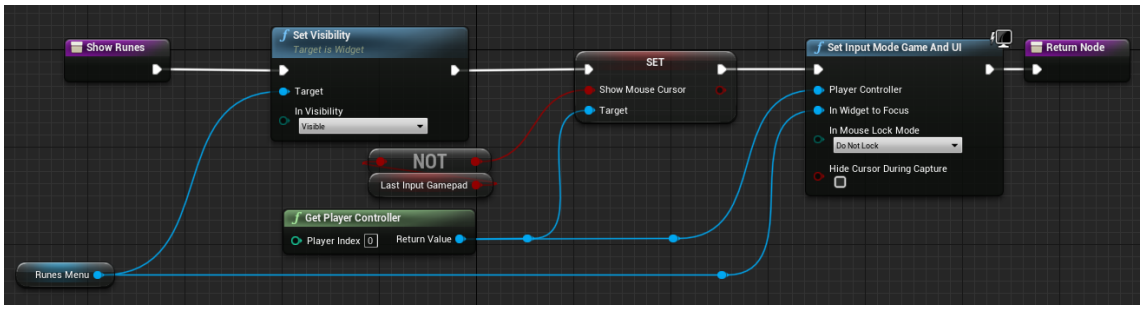

Figura 96. Implementació ShowRunes

6.1.7.1.1.2 Close Runes

CloseRunes és la funció antònima de *ShowRunes* (veure Apartat 6.1.7.1.1.1). En aquest cas s'oculta l'inventari i el cursor i es bloquegen els inputs per la UI. Veure Figura 97.

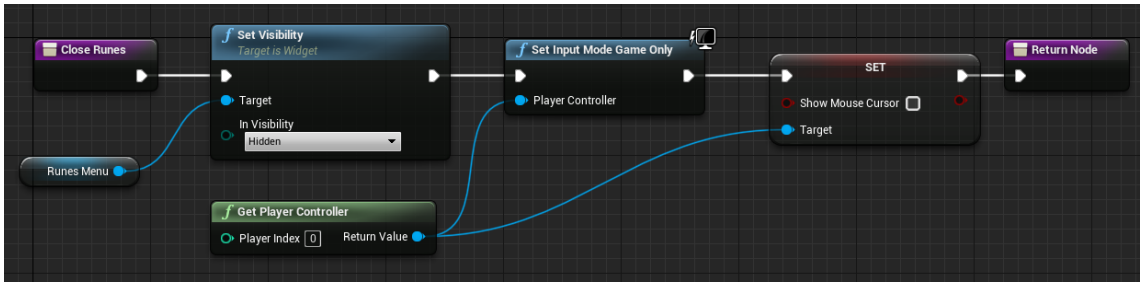

Figura 97. Implementació CloseRunes

6.1.7.1.2 Navegar per l'inventari

Tot i que pel cas del ratolí és ben senzill, ja que només s'ha de moure i fer clic on es desitja, per tal que el jugador es pugui moure per l'inventari amb el controlador cal implementar una lògica. Per aquest cas, hem optat per fer que es navegui utilitzant les fletxes ubicades a la part esquerra de la majoria de models.

Així doncs, l'inventari té dos *Event*, un per quan el jugador vol moure amunt o avall i un similar per dreta i esquerra. Com un *Widget* no pot rebre directament una acció, és el *ThirdPersonCharacter* qui rep aquests *Event*. En aquest cas s'ha utilitzat el fil de *Released* en comptes de *Pressed*, ja que el primer donava alguns problemes a l'hora de detectar. Per tant, el que es fa és rebre l'acció del jugador i, només en cas que l'inventari estigui obert, es crida l'*Event* respectiu, indicant cap a quina direcció vol anar, tal com s'aprecia

a la Figura 98. Aquestes funcions de navegació es troben explicades als Apartats 6.1.7.2.8 i 6.1.7.2.1.

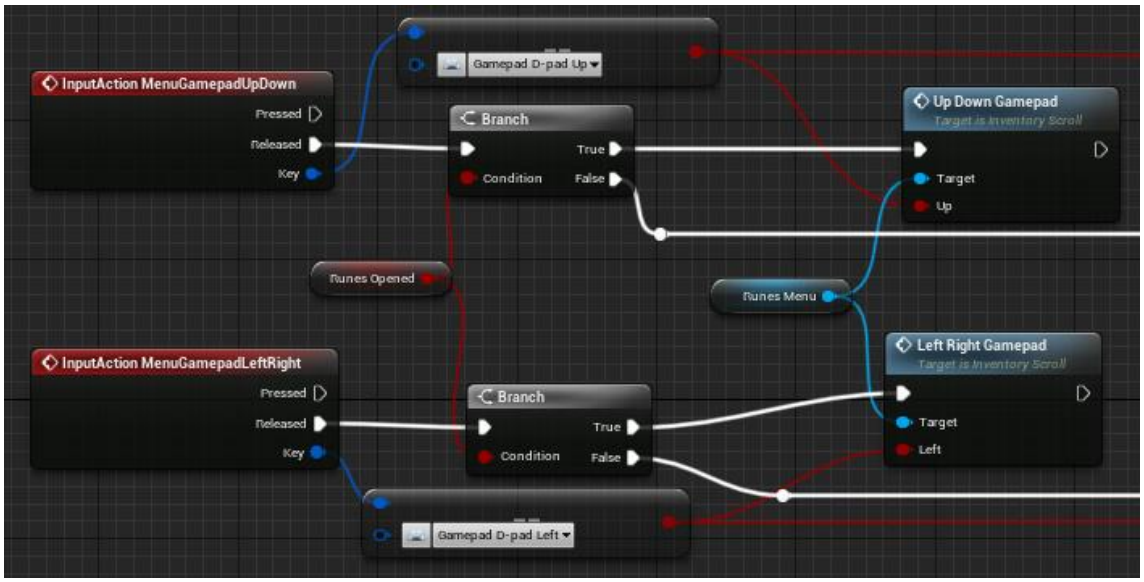

Figura 98. Navegació per l'inventari

6.1.7.1.3 AddRune

Quan el jugador vol incrustar una runa, es comprova que aquesta runa no és buida i s'itera per la llista de runes per comprovar si existeix alguna posició que tingui una runa buida (per això s'utilitza la funció *IsRuneEmpty* explicada a l'Apartat 6.1.7.1.6). Veure Figura 99.

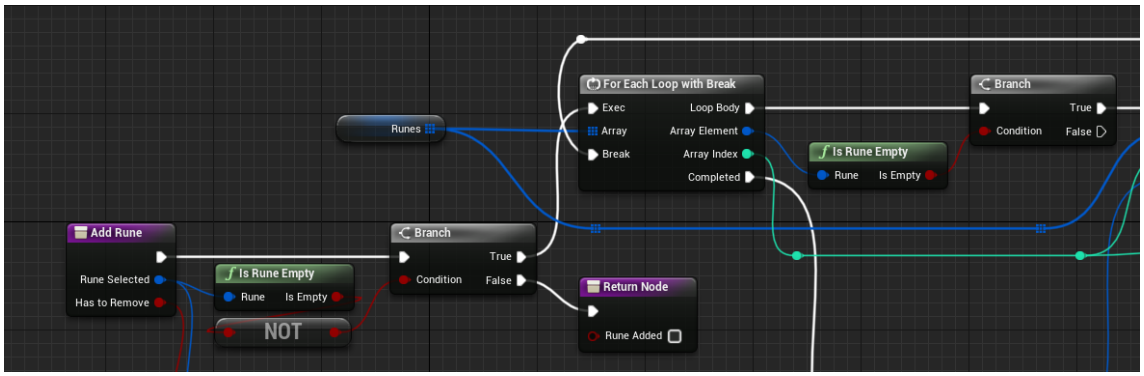

Figura 99. AddRune - Part 1

Un cop es troba el lloc per incrustar-la, es posen les dades de la runa a la posició indicada i s'afegeix la icona a l'espai corresponent (veure Apartat 6.1.7.2.10). A continuació decideix si s'elimina de l'inventari segons el valor de *hasToRemove*. Aquest serà cert quan s'afegeixi des del menú i fals quan s'estigui carregant la partida. Veure Figura 100.

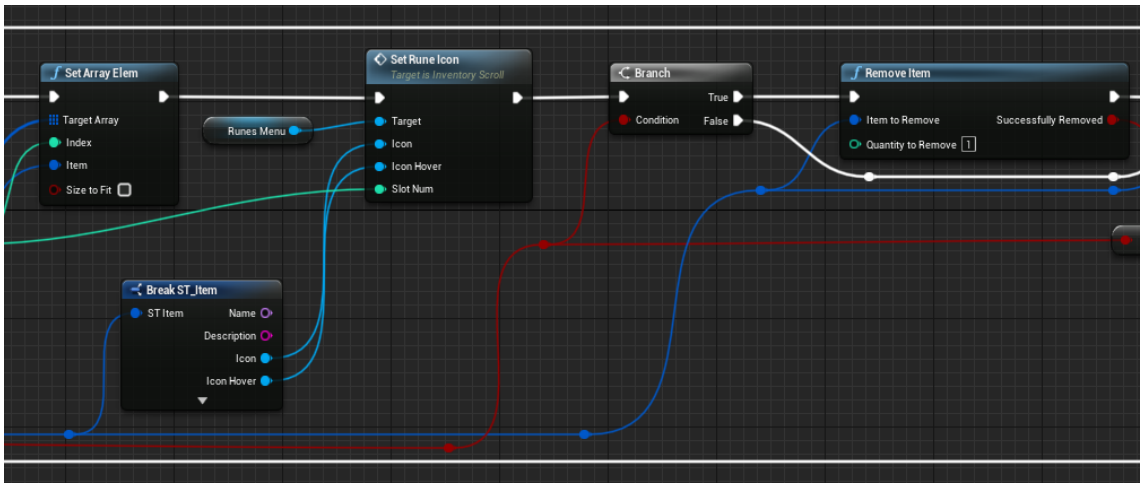

Figura 100. AddRune - Part 2

Finalment, suma les estadístiques de la runa al jugador amb la funció *ChangeRuneStats* (veure Apartat 6.1.7.1.5) i executa el *break* del bucle, per retornar si s'ha afegit correctament. Veure Figura 101.

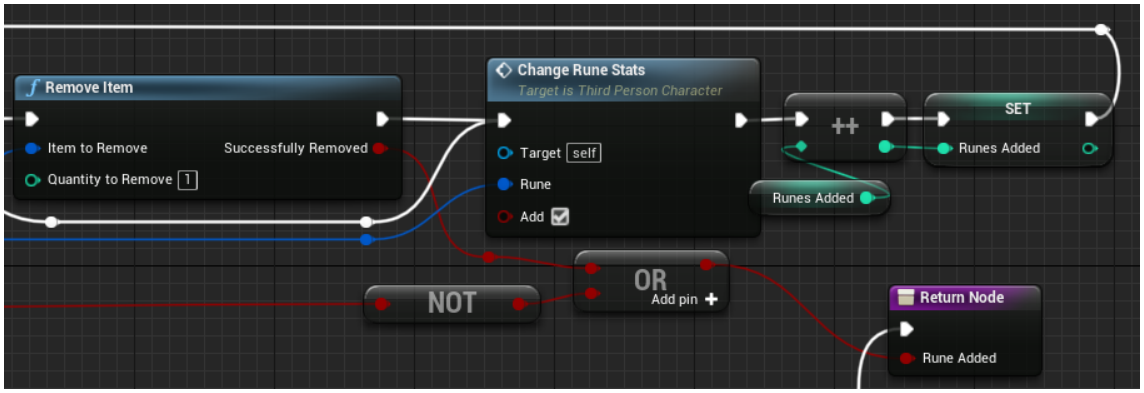

Figura 101. AddRune - Part 3

6.1.7.1.4 SubtractRune

De mateixa manera que es poden afegir runes, quan es fa clic sobre una ja afegida es crida la funció per extreure-la. El que fa aquesta funció és, obtenint l'índex com a paràmetre, obté la runa de la llista, la inserta a l'inventari, resta les seves característiques i assigna una runa buida a la posició corresponent, tal com es veu a la Figura 102.

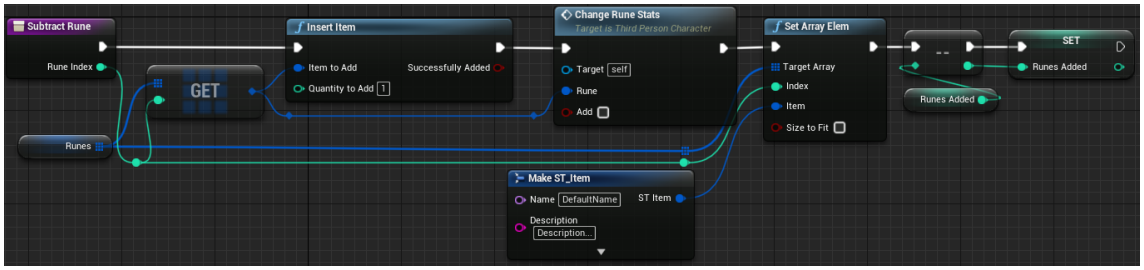

Figura 102. SubtractRune

6.1.7.1.5 ChangeRuneStats

Aquesta funció rep com a paràmetre les dades de la runa i el booleà que indica si aquestes dades s'han de restar o sumar. Per això inicialment s'assigna a unes variables locals els valors a afegir. Veure Figura 103.

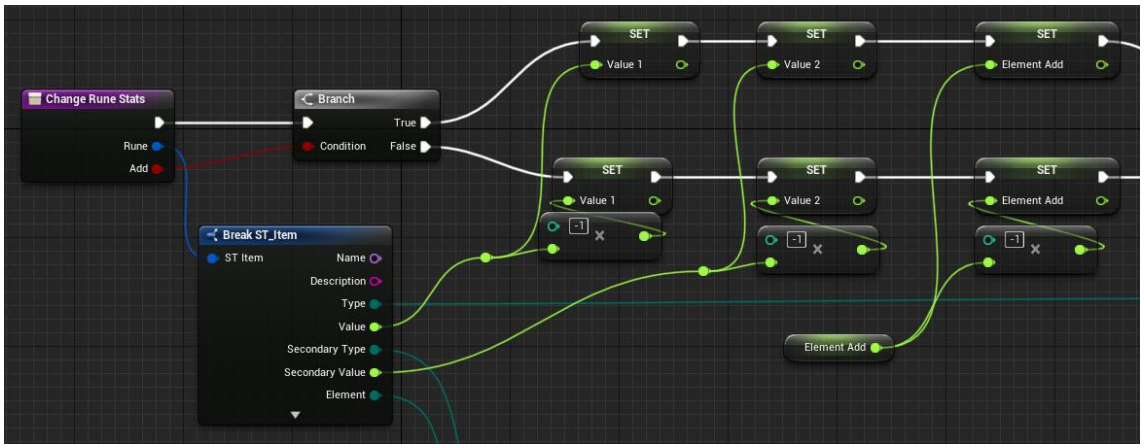

Figura 103. ChangeRuneStats - Part 1

A continuació, tria segons el tipus de la estadística (atac, vida o velocitat) per afegir el valor extra tal com es veu a la Figura 104 (ídem pel tipus secundari). La funció *SetNewMaxSpeed* rep el valor i el suma a la velocitat del jugador, obtinguda del component *Character Movement*.

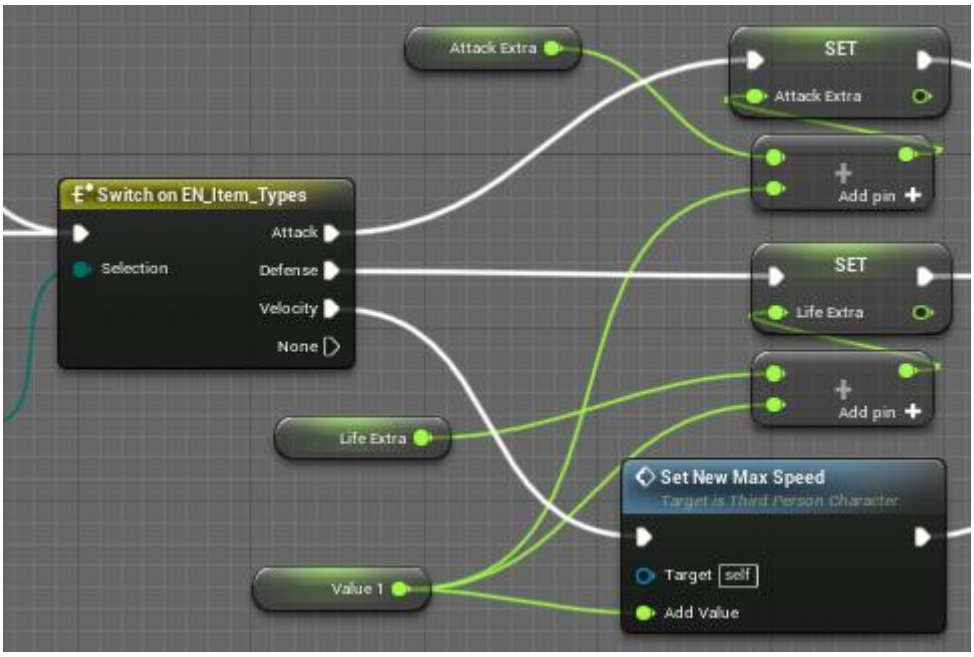

Figura 104. ChangeRuneStats - Part 2

Com s'aprecia a la Figura 105, pels danys elementals es fa el mateix, però afegint a la variable respectiva.

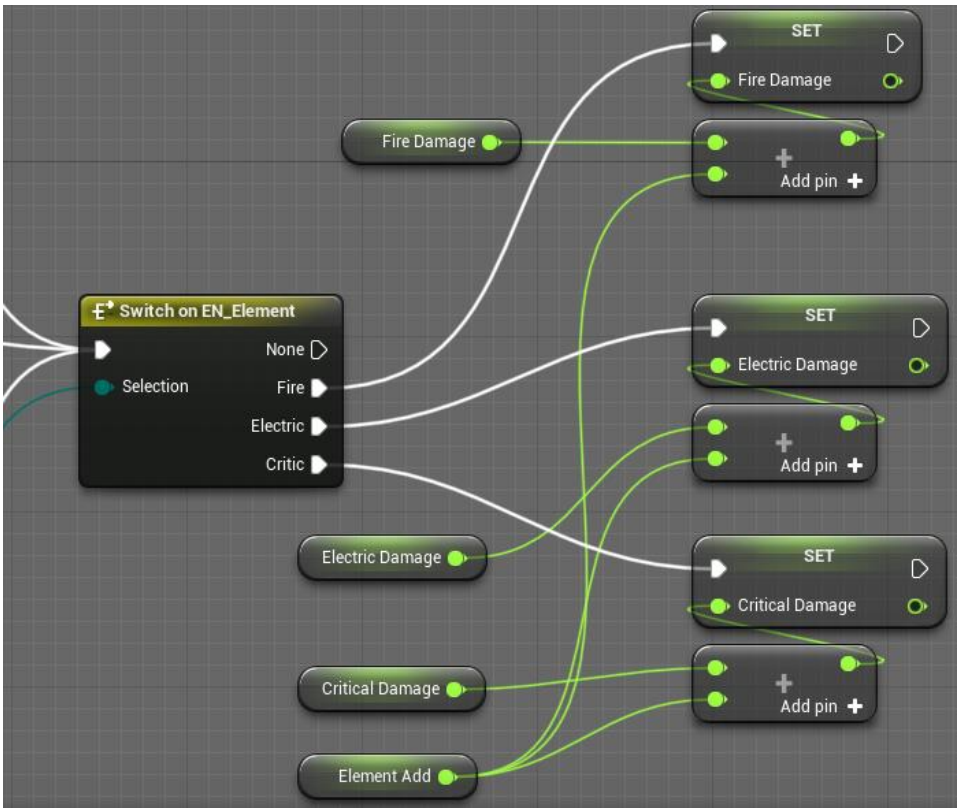

Figura 105. ChangeRuneStats - Part 3

6.1.7.1.6 IsRuneEmpty

Rebent una runa per paràmetre, aquesta funció comprova si té els valors per defecte per determinar si és buida. En aquest cas es fa amb el *Name*. Veure Figura 106.

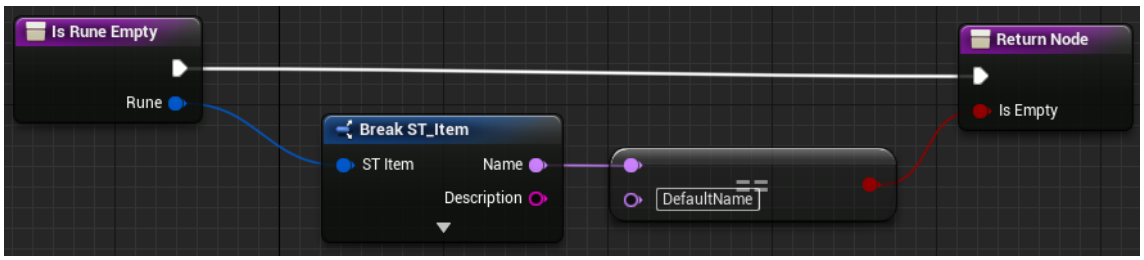

Figura 106. IsRuneEmpty

6.1.7.2 Runes a l'InventoryScroll

Com bé s'explica a l'Apartat 6.1.4, l'*InventoryScroll* és el *Blueprint* que mostra la part visual de l'inventari i gestiona les runes que té el jugador. A més, gestiona la navegació del jugador, tant per teclat i ratolí com per controlador, especialment la segona que requereix més codi. Amb les fletxes es pot pujar i baixar per la part dels *Inventory_Slots*, explicats a l'Apartat 6.1.7.3, i per la part de les runes ja incrustades. També permeten canviar entre la zona esquerra (on hi ha les runes) i la zona dreta (on hi ha els *Inventory_Slots*).

6.1.7.2.1 ChangePanelStyleHover

Aquesta funció rep com a paràmetre un índex, que ha d'estar entre 0 i 15 i un booleà per indicar si es vol posar com a *Hover* (seleccionat) o com a *Unhover* (deseleccionat). El que fa és obtenir l'*Inventory_Slot* en específic i crida a una funció pròpia anomenada

ChangeIsHover, veure Apartat 6.1.7.3.4. A més, si el booleà és cert, es fa una crida a l'*Event SetFocusToSlot,* explicat a l'Apartat 6.1.7.3.1. Per la implementació de la funció, veure Figura 107.

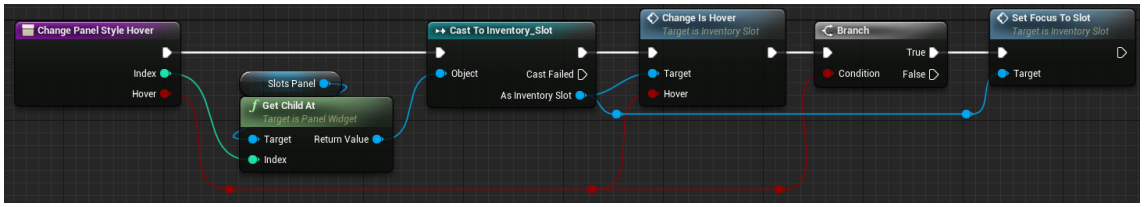

Figura 107. ChangePanelStyleHover

6.1.7.2.2 SetRuneHovered

Aquesta funció realitza dues tasques diferents, segons els paràmetres d'entrada. Rep un índex de la runa i el booleà *SetAllUnhovered*. Si aquest últim és cert, deixa totes les runes deseleccionades fent un recorregut per la llista de runes i cridant a la funció *SetHoverStyleByIndex*, veure la següent secció. Aquesta part es veu a la Figura 108.

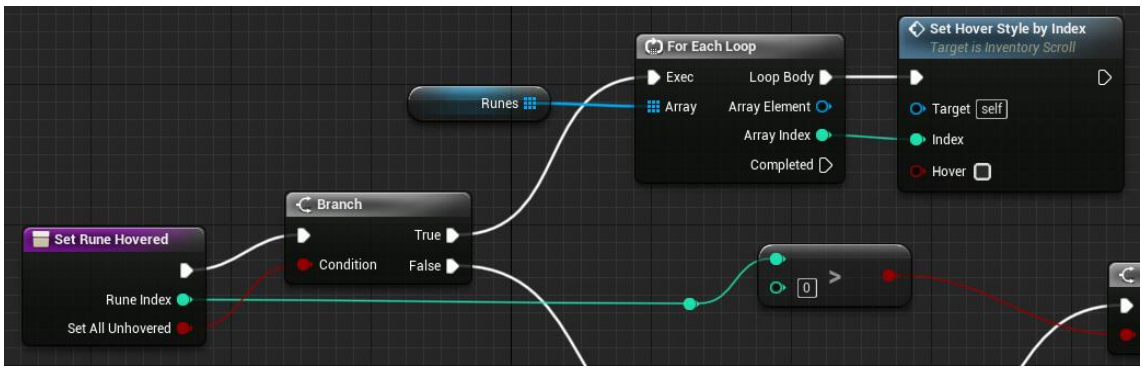

Figura 108. SetRuneHovered - Part 1

En canvi, si *SetAllUnhovered* és fals, s'executa el codi mostrat a la Figura 109. En cas que *RuneHovered* (una referència a la runa seleccionada) sigui vàlid, es selecciona la següent o l'anterior utilitzant *SelectRuneAfter* (veure Apartat 6.1.7.2.5) o *SelectRuneBefore* (veure Apartat 6.1.7.2.6), respectivament. En canvi, si no té cap valor, es fa un recorregut per la llista de runes per seleccionar el primer que estigui activat.

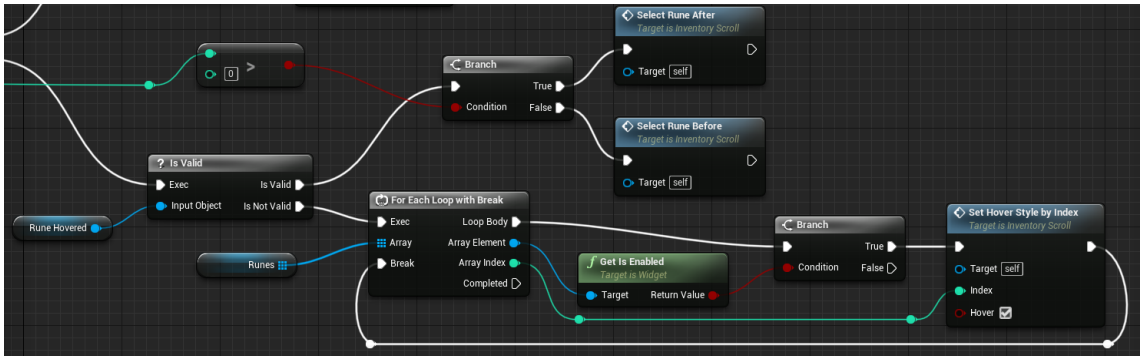

Figura 109. SetRuneHovered - Part 2

6.1.7.2.3 SetHoverStyleByIndex

Aquesta funció, donat un índex i un booleà, li dona a la runa seleccionada l'estil de seleccionada o deseleccionada, segons si *Hover* és cert o fals, respectivament. Canvia l'estil la pròpia runa amb *ChangeRuneStyle* (explicat a continuació) i de la imatge de fons utilitzant *ChangeHoverRuneHolder* (esmentat a l'Apartat 6.1.7.2.7). Veure Figura 110.

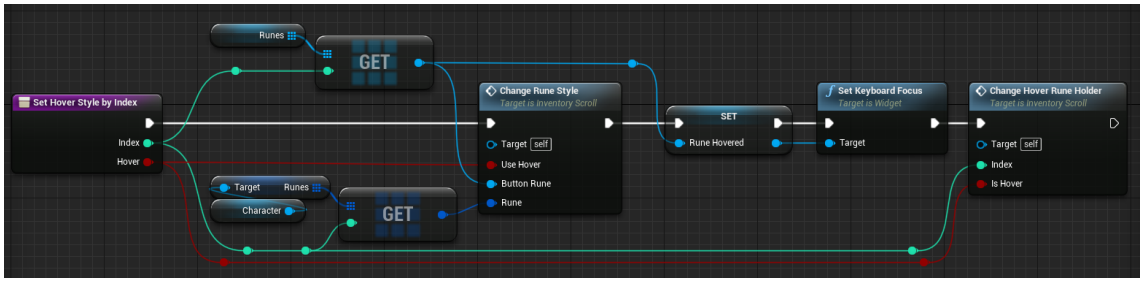

Figura 110. SetHoverStyleByIndex

6.1.7.2.4 ChangeRuneStyle

Donat un booleà que indica si s'ha de fer servir l'estil de seleccionat o deseleccionat, el botó de la runa i la informació de la pròpia runa, canvia l'estil del botó per l'estil indicat. En aquest cas es posa la imatge a l'estil *Normal*, ja que amb el controlador no s'ha trobat cap funció que permeti activar l'estil *Hovered* com si el cursor estigués situat a sobre del botó. Per això s'ha optat per modificar l'estil del botó. Veure Figura 111.

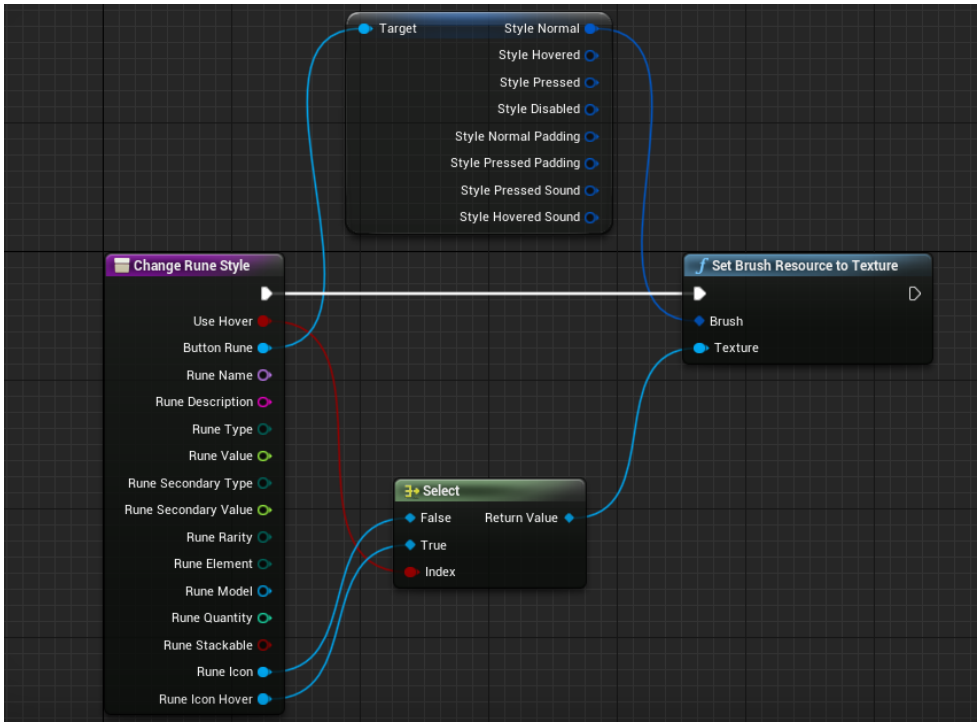

Figura 111. ChangeRuneStyle

6.1.7.2.5 SelectRuneAfter

L'objectiu d'aquesta funció és deseleccionar la runa actual i seleccionar-ne una que estigui a una casella inferior, tenint en compte que pot haver-hi forats o no haver-hi cap altra per sota. Inicialment es recorre la llista de runes per trobar la seleccionada (juntament amb el seu índex). Si el jugador només té una runa, o la seleccionada és la última o és la segona i la tercera és buida, no es fa res. Veure Figura 112.

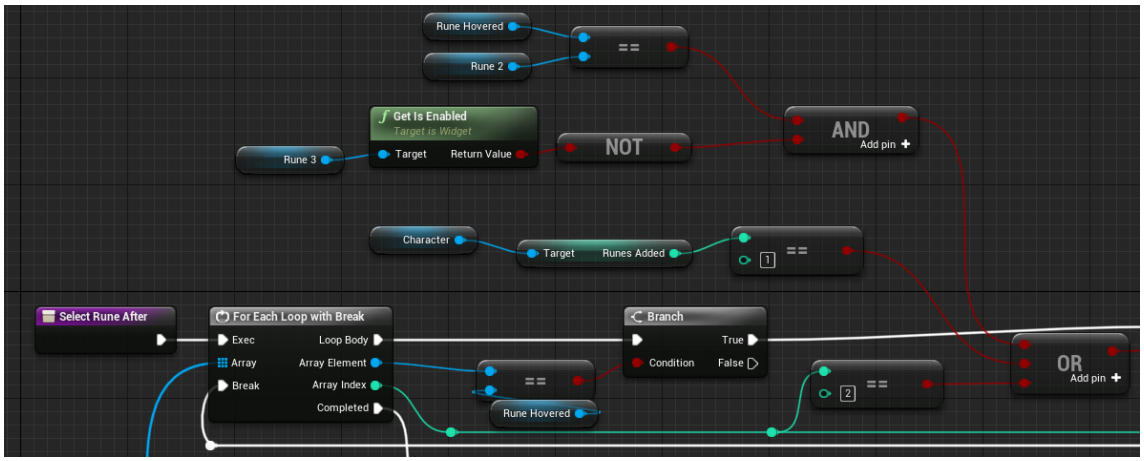

Figura 112. SelectRuneAfter - Part 1

En cas que no es compleixi cap de les condicions anteriors, es guarda l'índex a una variable local i es posa l'estil deseleccionat a la runa actual amb *SetHoverStyleByIndex*, esmentat prèviament. Veure Figura 113.

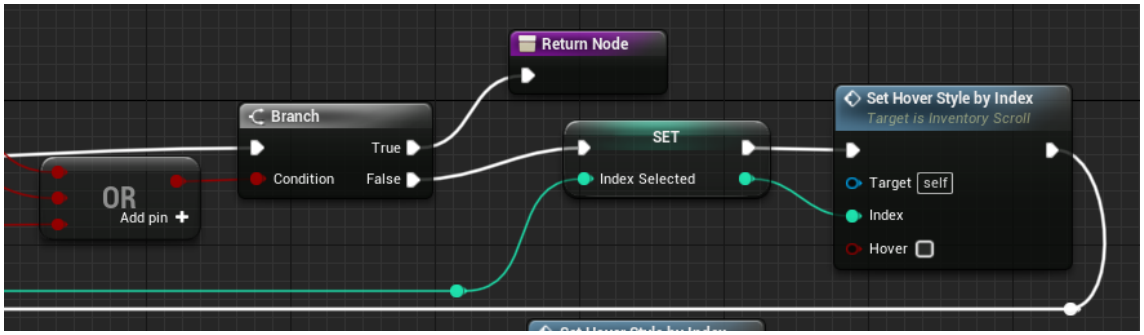

Figura 113. SelectRuneAfter - Part 2

Finalment, es fa un recorregut des del valor següent a l'*IndexSelected* fins a la última runa. La primera que es trobi es selecciona amb *SetHoverStyleByIndex*. Veure Figura 114.

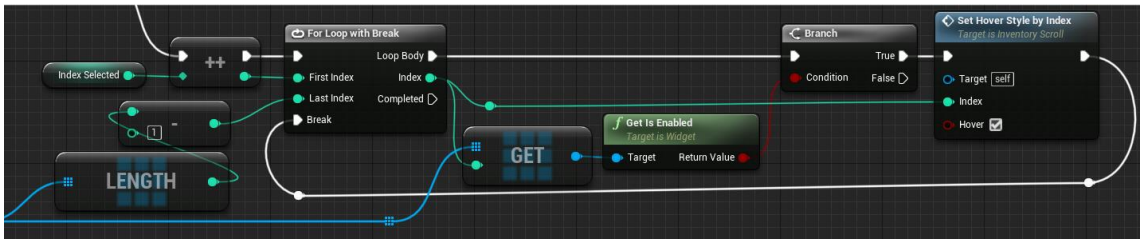

Figura 114. SelectRuneAfter - Part 3

6.1.7.2.6 SelectRuneBefore

Aquesta funció realitza una tasca similar a l'anterior, però en comptes de seleccionar una que estigui per sota, selecciona una runa que estigui per sobre. En aquest cas es comprova si l'índex de la runa seleccionada és zero o si està sobre la segona i la primera és buida, tal i com es pot apreciar a la Figura 115. El contingut de la funció després del segon *Branch* és igual al codi mostrat a la Figura 113.

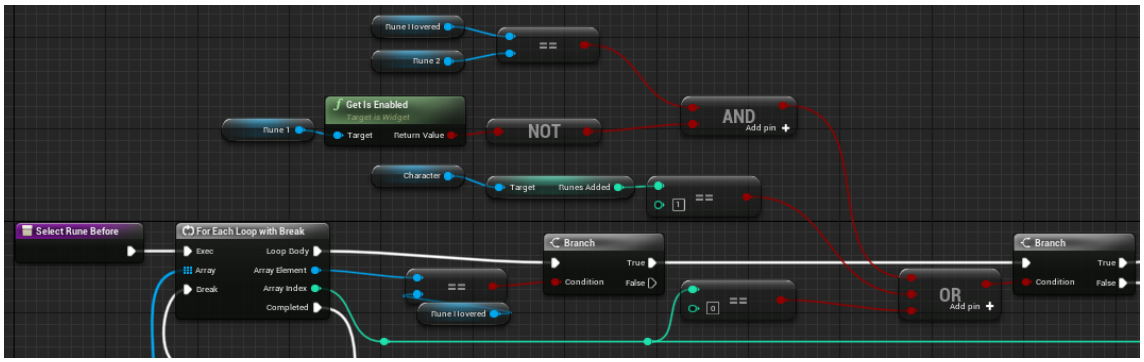

Figura 115. SelectRuneBefore - Part 1

Pel que fa al bucle per seleccionar la runa anterior, es fa una mica diferent ja que s'utilitza un *Reverse for Each Loop*, perquè no s'ha trobat un node que contingui el *Break* i faci un *Reverse For Loop*, així que en aquest cas es comprova que l'índex sigui inferior a *IndexSelected*, no s'hagi acabat i la runa actual estigui activada. Veure Figura 116.

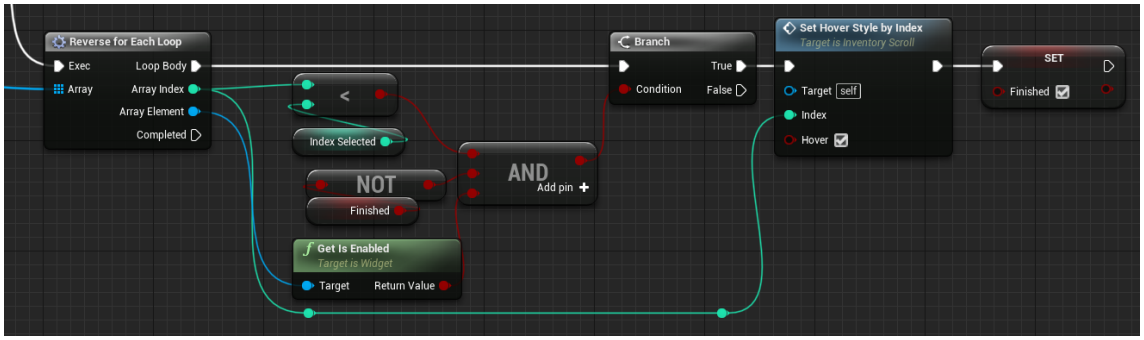

Figura 116. SelectRuneBefore - Part 2

6.1.7.2.7 ChangeHoverRuneHolder

En aquest cas es rep com a entrada l'índex de la runa i un booleà per saber si s'ha d'utilitzar la imatge normal o la imatge de l'estil *Hover*. Tenint en compte els paràmetres esmentats es selecciona la casella de la runa indicada i se li assigna el nou estil. Veure Figura 117.

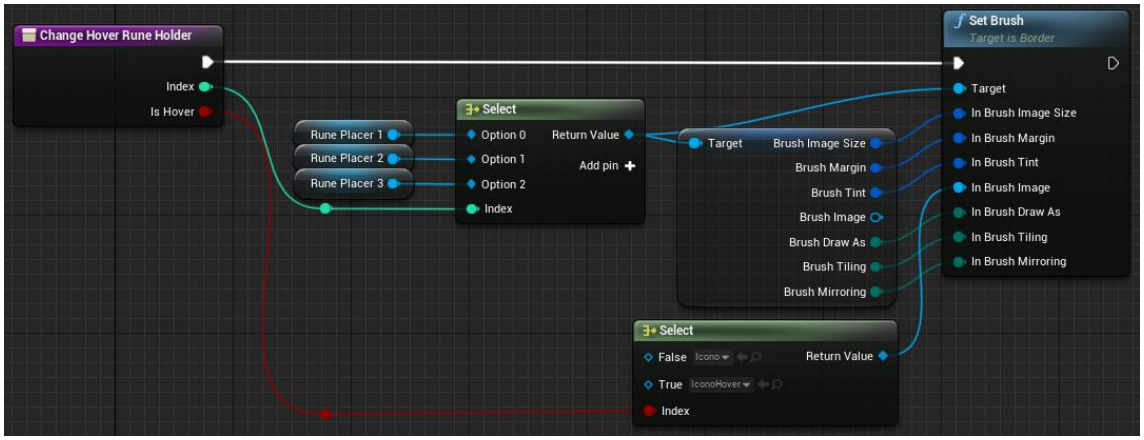

Figura 117. ChangeHoverRuneHolder

6.1.7.2.8 Event UpDownGamepad

Quan el jugador clica la fletxa d'amunt o d'avall, al *ThirdPersonCharacter* s'activa un *Event*, esmentat a l'Apartat 6.1.7.1.2. Si l'inventari està obert es crida a l'*Event UpDownGamepad*, enviant cert, si es desitja anar cap amunt i fals altrament.

El que fa aquest *Event* és donar valor a *AddValue* (-1 si *Up* és cert, -1 si és fals). En cas que el jugador estigui sobre els *InventorySlots*, es desmarca l'actual i es marca el següent utilitzant *ChangePanelStyleHover*, explicat a l'Apartat 6.1.7.2.1. Si està sobre les runes afegides, crida a la funció *SetRuneHovered* que gestiona el canvi d'estils per marcar la runa actual, com s'explica a l'Apartat 6.1.7.2.2. Veure la Figura 118.

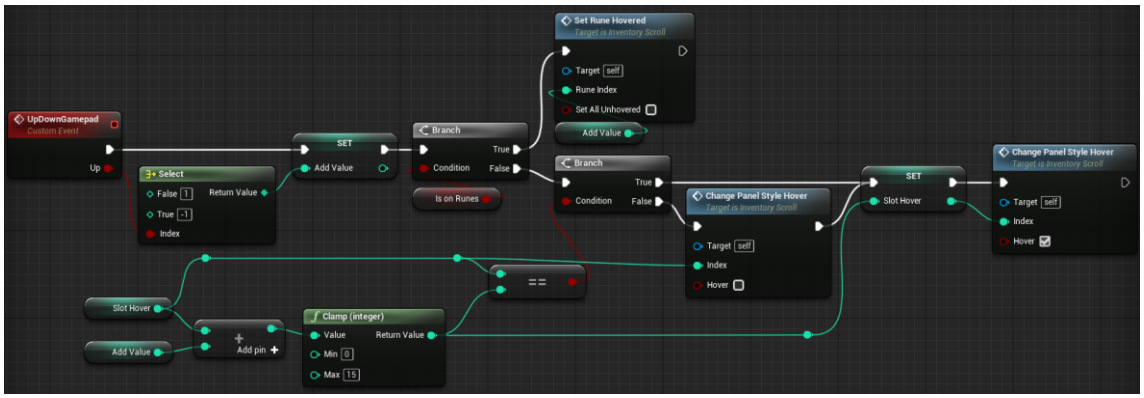

Figura 118. UpDownGamepad

6.1.7.2.9 Event LeftRightGamepad

Similar a l'*Event UpDownGamepad*, esmentat a l'Apartat 6.1.7.2.8, *InventoryScroll* té un *Event* per capturar quan el jugador clica les flextes esquerra i dreta, que s'utilitzen per canviar entre la zona de runes seleccionades i la de runes guardades a l'inventari.

Si es clica al botó esquerre, es comprova si no té runes o ja està situat sobre les runes. Si això és cert no fa res. En cas contrari, es deselecciona l'*InventorySlot* seleccionat amb *ChangePanelStyleHover* (explicat a l'Apartat 6.1.7.2.1), tal com es veu a la Figura 119. A continuació es selecciona la primera runa amb *SetRuneHovered* (esmentat a l'Apartat 6.1.7.2.2). Veure Figura 120.

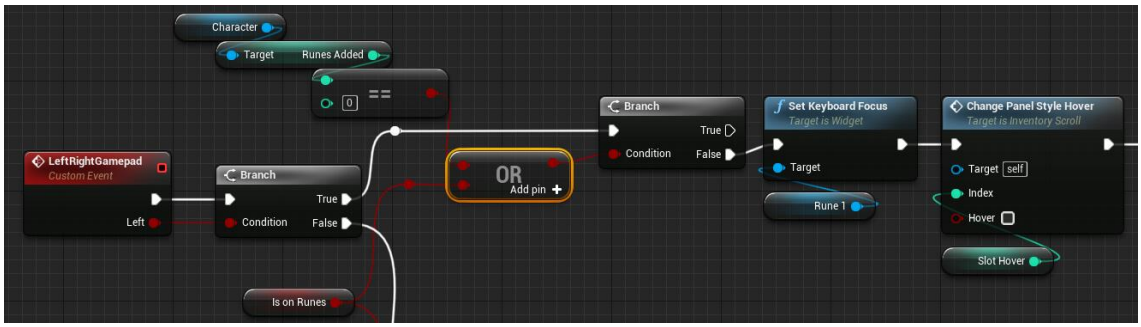

Figura 119. LeftRightGamepad - Part 1

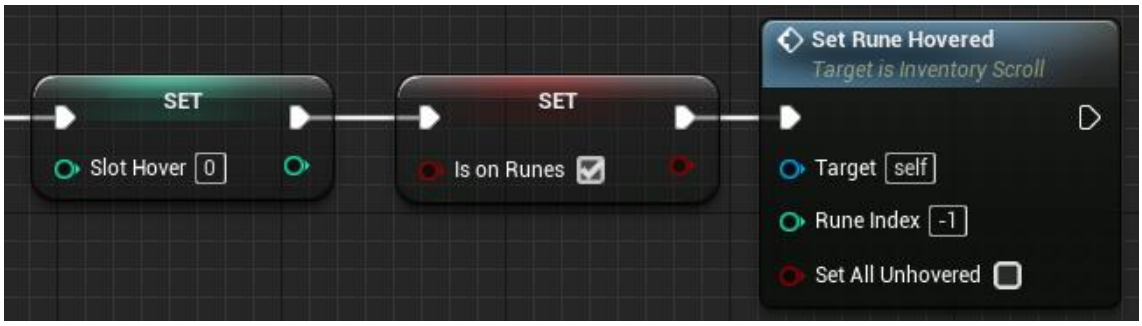

Figura 120. LeftRightGamepad - Part 2

D'altra banda, si l'usuari està sobre les runes i clica la fletxa dreta, es deseleccionen totes les runes amb *SetRuneHovered* (veure Apartat 6.1.7.2.2) i es selecciona el primer *InventorySlot* amb la funció *SetFocusToSlot* (esmentat a l'Apartat 6.1.7.3.1) i *ChangePanelStyleHover* (explicat a l'Apartat 6.1.7.2.1). Veure Figura 121.

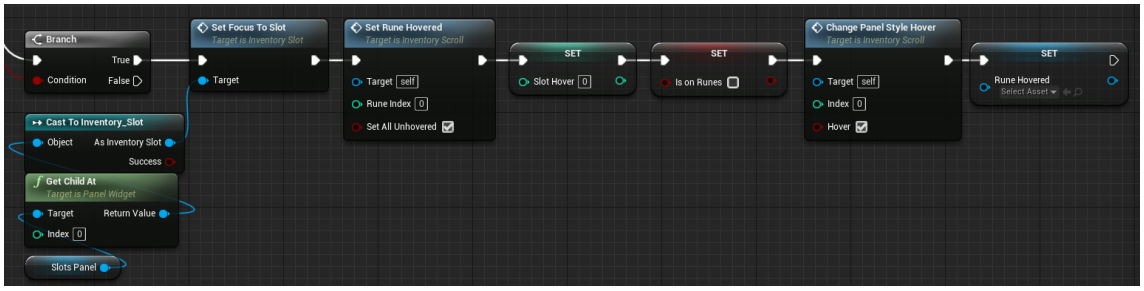

Figura 121. LeftRightGamepad - Part 3

6.1.7.2.10 SetRuneIcon

Aquesta funció rep com a paràmetre l'índex de la posició de la runa (en aquest cas entre zero i dos, ambdós inclosos) i els dos estils que s'han d'afegir: *Icon* pel normal i *IconHover* quan el jugador es situa sobre el botó. Així doncs, aquesta funció canvia l'estil del botó corresponent a la nova runa introduïda, i activa el botó. Veure Figura 122.

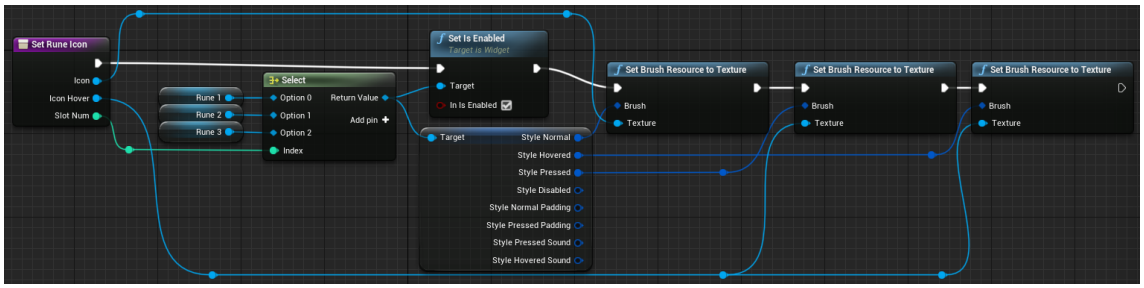

Figura 122. SetRuneIcon

6.1.7.2.11 OnClicked a les runes

Un cop el jugador ha introduït una runa, pot decidir extreure-la per posar-n'hi una altra. Per poder fer-ho només ha de fer clic sobre el botó corresponent de la runa. En aquest cas es crida a l'*Event* específic, que crida a la funció *SubtractRuneSelected*, explicat a la següent secció, amb els paràmetres corresponents. A continuació, selecciona el primer *InventorySlot* cridant a l'*Event LeftRightGamepad*, esmentat a l'Apartat 6.1.7.2.9. Veure Figura 123.

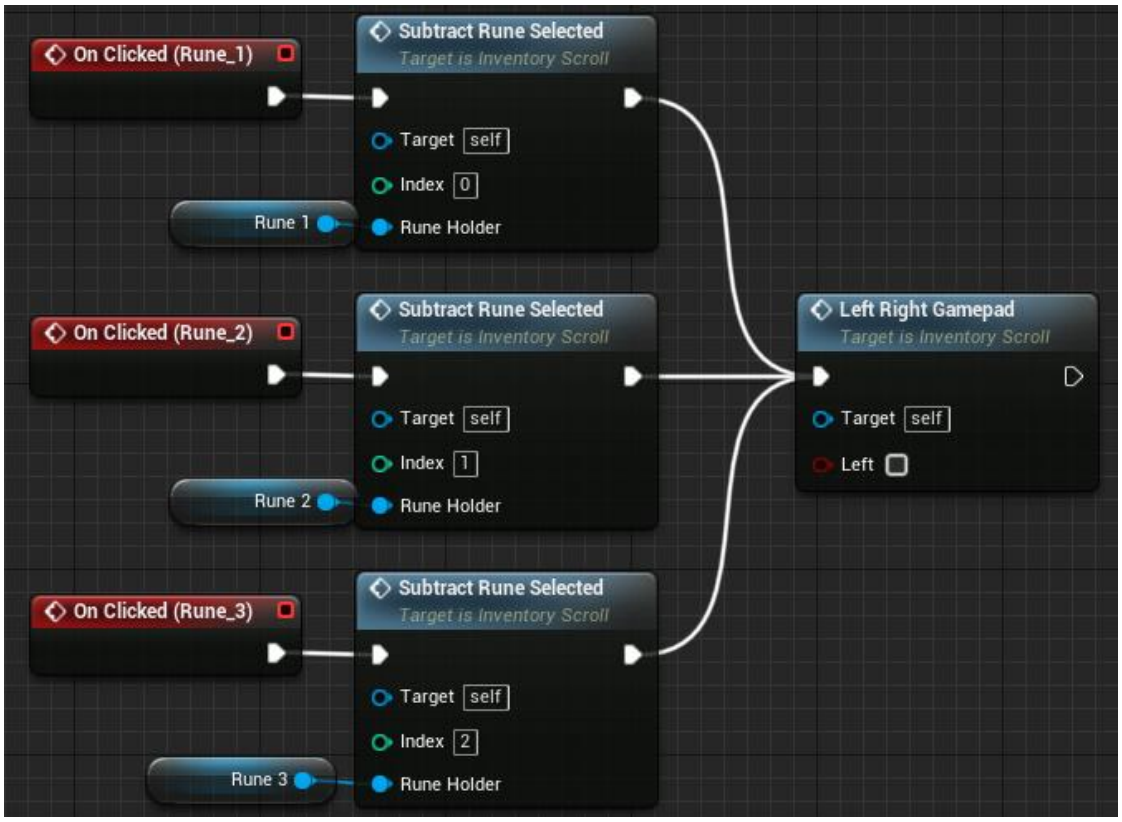

Figura 123. Events OnClicked als botons de les runes

6.1.7.2.12 SubtractRuneSelected

Donat un índex i la referència a un botó, extreu la runa utilitzant la funció *SubtractRune* del *ThirdPersonCharacter*, explicada a l'Apartat 6.1.7.1.4, i desactiva el botó. Veure Figura 124.

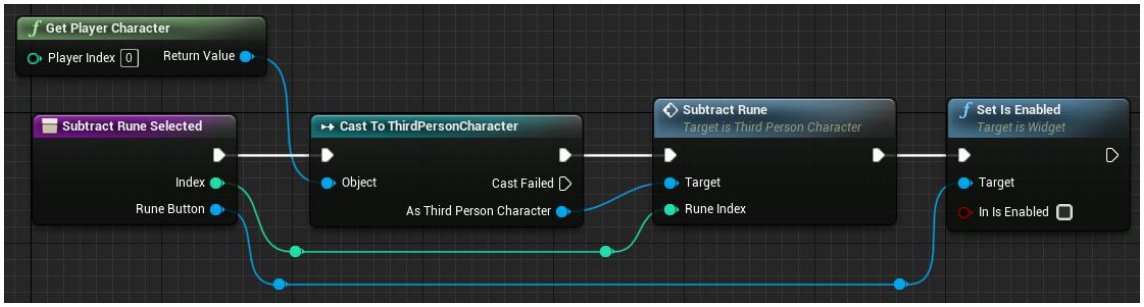

Figura 124. SubtractRuneSelected

6.1.7.2.13 OnHovered i OnUnhovered a les runes

Aquests *Events* s'han realitzat per la navegació amb teclat i ratolí. Quan el jugador situa el punter sobre qualsevol dels tres botons es crida la funció *ChangeHoverRuneHolder*, esmentada a l'Apartat 6.1.7.2.7, indicant el valor de l'índex i posant *IsHover* a cert per *OnHovered* i fals per *OnUnhovered*. Veure Figura 125.

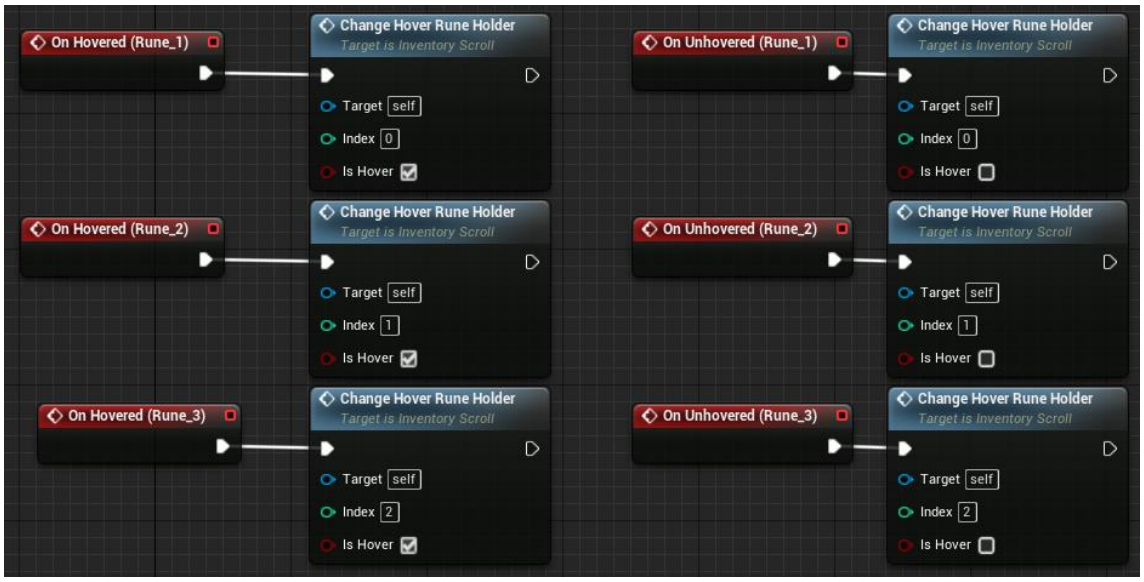

Figura 125. Events OnHovered i OnUnhovered als botons de les runes

6.1.7.3 Inventory Slot

Aquest *Widget* s'utilitza per cada compartiment de l'inventari. La funció és guardar les runes que el jugador recull i incrustar-les quan es fa clic a sobre.

6.1.7.3.1 SetFocusToSlot

Quan el jugador utilitza el controlador, s'ha de posar el focus sobre el botó per tal que es cliqui quan clica el botó indicat. Per això s'utilitza aquest *Event* que crida a la funció *SetKeyboardFocus*. Veure Figura 126.

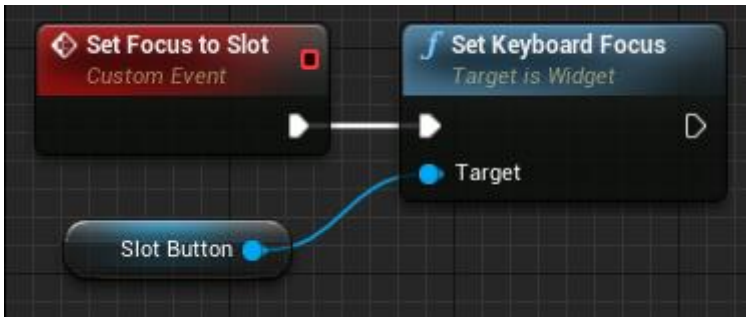

Figura 126. SetFocusToSlot

6.1.7.3.2 OnClicked (SlotButton)

Quan el jugador clica a qualsevol dels *InventorySlot*, es crida a la funció *AddRune* del *ThirdPersonCharacter*, explicada a l'Apartat 6.1.7.1.3, enviant la informació de la runa com a paràmetre i indicant que s'ha d'eliminar de l'inventari. Veure Figura 127.

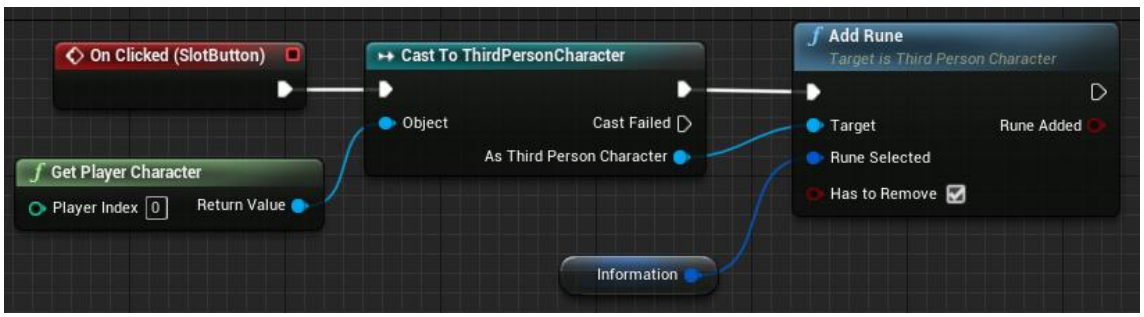

Figura 127. Event OnClicked al botó de l'"InventorySlot"

6.1.7.3.3 OnHovered (SlotButton)

Si l'usuari col·loca el ratolí sobre qualsevol *InventorySlot* s'ha de canviar l'estil del fons i de la caixa de la runa. Per això s'utilitzen els *Events OnHovered* i *OnUnhovered*, que criden a la funció *ChangeIsHover*, esmentada a la següent secció. Veure Figura 128.

Figura 128. Events OnHovered i OnUnhovered al botó de l'"InventorySlot"

6.1.7.3.4 ChangeIsHover

Aquesta funció rep com a paràmetre un booleà per determinar si s'ha de posar l'estil seleccionat o deseleccionat. Així doncs, tria les imatges corresponents i actualitza el fons de la informació de la runa i el de la icona. Veure Figura 129.

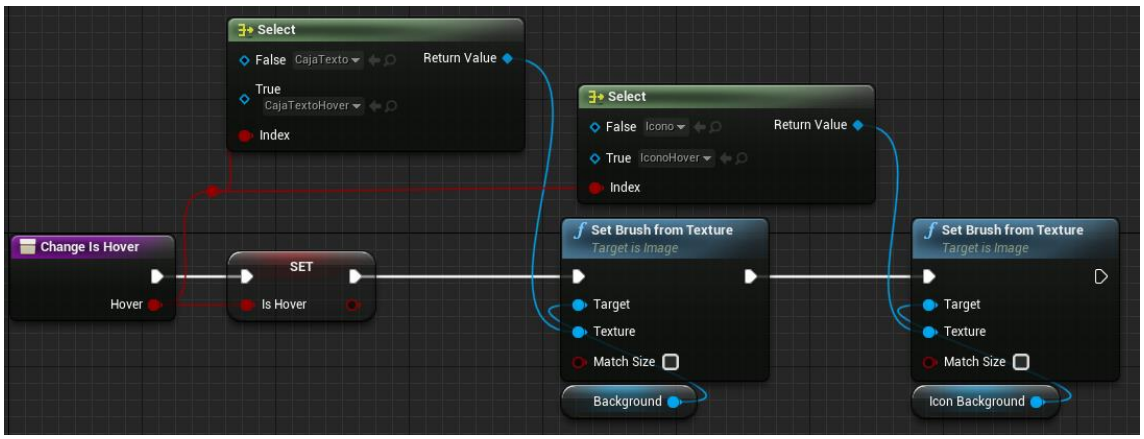

Figura 129. ChangeIsHover

6.2 Cofre

El cofre és un dels elements importants en el joc. És on es poden trobar les diferents runes a aconseguir. Aquest és genèric, i pot funcionar de dues maneres diferents, pot tenir una runa determinada, la qual donarà a l'obrir-lo, o pot ser una d'aleatòria.

6.2.1 Model

El cofre té 2 models: un de tancat, i un altre d'obert. I simplement canvia d'un a l'altre quan s'obre. Veure Figura 130 i Figura 131.

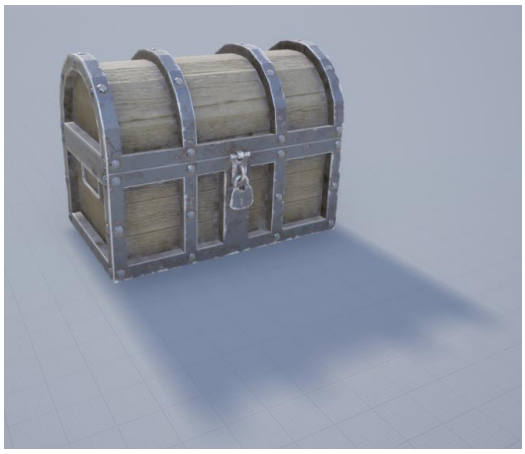

Figura 130. Cofre tancat

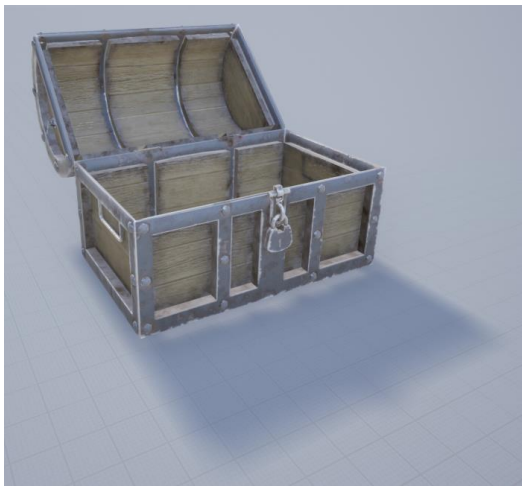

Figura 131. Cofre obert

6.2.2 Pickup_Nameplate

Aquest és un component que s'haurà d'afegir al cofre. És simplement un petit *Widget* que indica al jugador amb quin botó pot interactuar amb el cofre per obrir-lo.

Figura 132. Pickup_Nameplate

6.2.3 ChestOpenedWidget

Aquest és un *Widget* amb un text que indica quina runa s'ha aconseguit. Apareix en mig segon, es mostra 1.5 segons, i desapareix en 1 segon. Veure Figura 133.

Figura 133. ChestOpenedWidget

6.2.4 Chest

Chest és el nom del Blueprint que implementa tots els components i lògica necessària per al funcionament del cofre dins el món.

6.2.4.1 Estructura

L'estructura del cofre és molt senzilla. Té l'element pare, que no és més que el centre de l'objecte, el model del cofre que s'ha vist abans, el *NamePlate*, que s'ha explicat en l'anterior apartat, i una esfera que detecta si estàs a prop del cofre. Veure Figura 134.

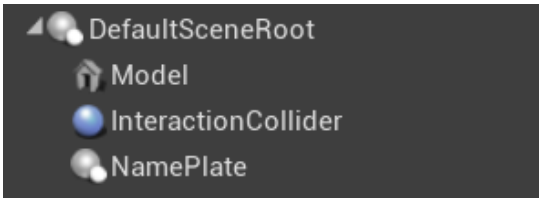

Figura 134. Estructura del Chest

6.2.4.2 Atributs

Com a atributs del cofre hi ha l'ítem en concret, que és una runa (pot tenir una d'assignada o estar buit), les runes comuns, les runes èpiques, si el cofre està obert o no, i el nom de la runa a mostrar per pantalla. Veure Figura 135.

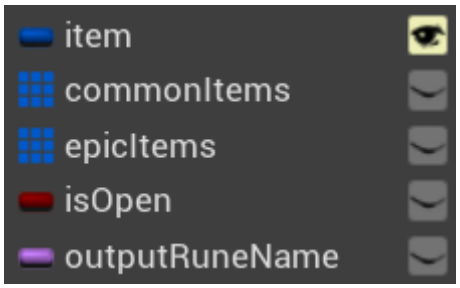

Figura 135. Atributs del Chest

6.2.4.3 Lògica

En aquest apartat s'expliquen les diverses funcions i funcionalitats que té el Cofre per actuar dins el joc com ha de fer-ho.

6.2.4.3.1 OnComponentBeginOverlap (InteractionCollider)

Aquest és un *Event* que s'executa quan algun objecte intersecciona amb el component *InteractionCollider* explicat abans.

El què fa és, primer de tot, comprovar si l'objecte que ha interseccionat és del tipus *ThirdPersonCharacter*, la classe del personatge que utilitza el jugador (aquesta s'explicarà en un apartat posterior). Si ho és, activa la visualització del *NamePlate* (Apartat 6.2.2) i la possibilitat de que el jugador interaccioni amb el cofre. En resum, fa que el jugador pugui obrir el cofre. Veure Figura 136.

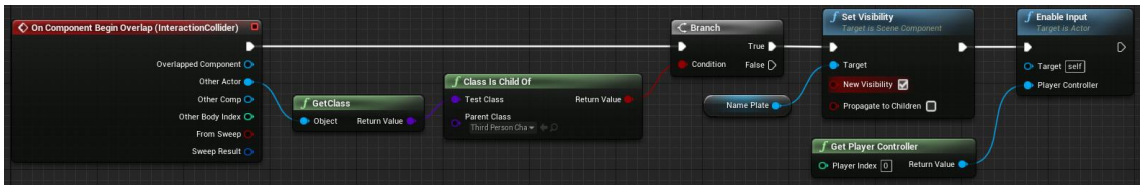

Figura 136. OnComponentBeginOverlap (InteractionCollider)

6.2.4.3.2 OnComponentEndOverlap (InteractionCollider)

Aquest *Event* funciona de la mateixa manera que l'anterior, però s'activa quan un objecte deixa de interseccionar. El que fa és, primer, comprovar que es tracti del jugador i, després, desactivar la visibilitat del *NamePlate* i la possibilitat d'interaccionar. Veure Figura 137.

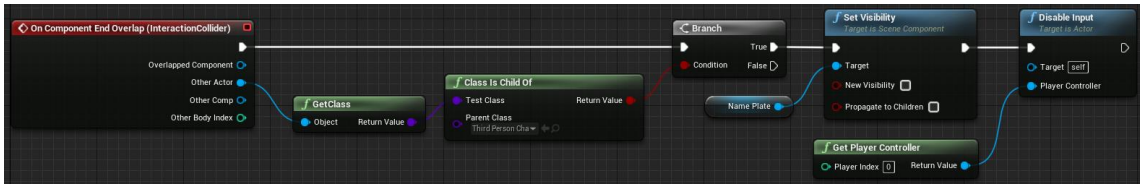

Figura 137. OnComponentEndOverlap (InteractionCollider)

6.2.4.3.3 InputAction Interact

Aquest és un *Event* que s'activa al prémer el botó d'interactuar. El que fa primer és comprovar si existeix l'ítem que li haguem assignat al cofre a la taula *DT_Items* (Apartat 6.1). Si li hem assignat algun objecte, el trobarà, si no, no. Si troba l'objecte, fa la inserció directament (apartat de la lògica de l'inventari), i guarda el nom de la runa. Si no el troba, donarà un d'aleatori d'entre els comuns i els èpics. Primer de tot, es fan dues branques per saber si serà comú (amb un 70% de possibilitats) o èpic (30 % de possibilitats). Veure Figura 138.

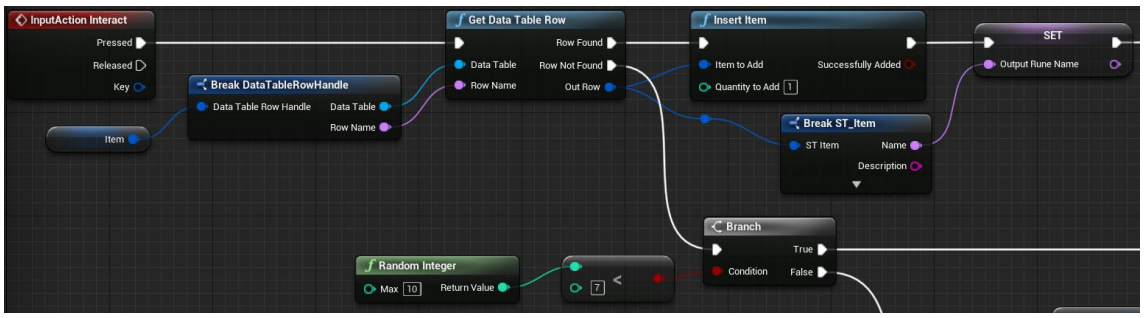

Figura 138. InputAction Interact - Part 1

Tots dos camins seran iguals: s'agafarà una runa aleatòria d'aquell grup, s'inserirà, i guardarà el nom de la runa escollida. L'única diferencia que en un s'agafarà una de comú o una d'èpica. Veure Figura 139.

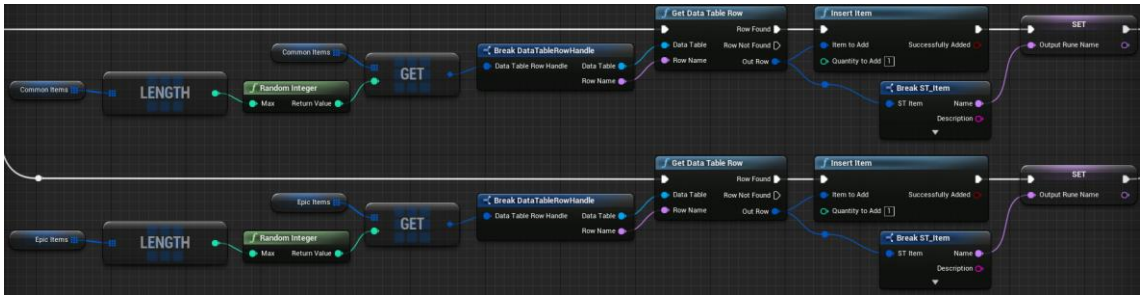

Figura 139. InputAction Interact - Part 2

Al final de tot, tots 3 camins s'uneixen per mostrar el nom de la runa aconseguida amb el *ChestOpenedWidget* i cridar l'*Event SetOpened*, que s'explicarà a continuació. Veure Figura 140.

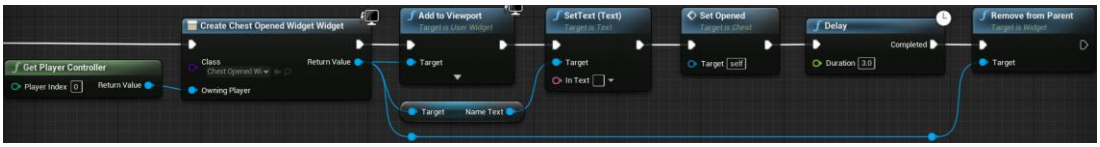

Figura 140. InputAction Interact - Part 3

6.2.4.3.4 SetOpened

Aquest *Event* serveix per desactivar la possibilitat de tornar a obrir el cofre: el guarda com a obert, li elimina la visibilitat del *NamePlate*, treu la possibilitat d'interactuar, desactiva els *Events OnComponentBeginOverlap* i *OnComponentEndOverlap*, i li posa el model del cofre obert. Veure Figura 141.

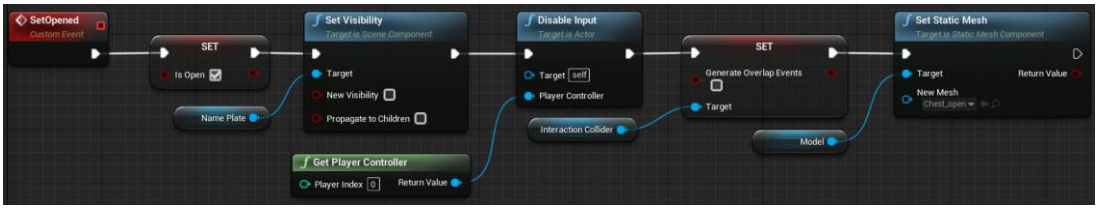

Figura 141. SetOpened

6.2.5 ChestClosed

Blueprint que hereta directament de *Chest*. Aquest és, bàsicament, un *Chest* sense la possibilitat d'obrir-lo directament. Es necessita que alguna cosa activi la possibilitat d'obrir-lo. S'utilitza en els cofres que requereixen d'un diàleg (l'apartat de diàlegs explica aquest tema en profunditat). No té res d'especial més que un parell de canvis en el comportament.

6.2.5.1 Event BeginPlay

Event que s'activa a l'iniciar l'aplicació. Desactiva la possibilitat del jugador d'interactuar amb el cofre. Veure Figura 142.

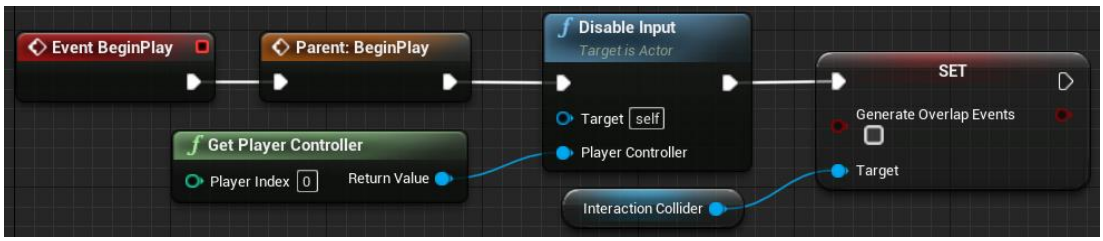

Figura 142. Event BeginPlay

6.2.5.2 Event SetCanBeOpened

Es tracta d'un *Event* que cal cridar des de fora. Reactiva la possibilitat al jugador d'interactuar amb el cofre, sempre que no estigui ja obert. Veure Figura 143.

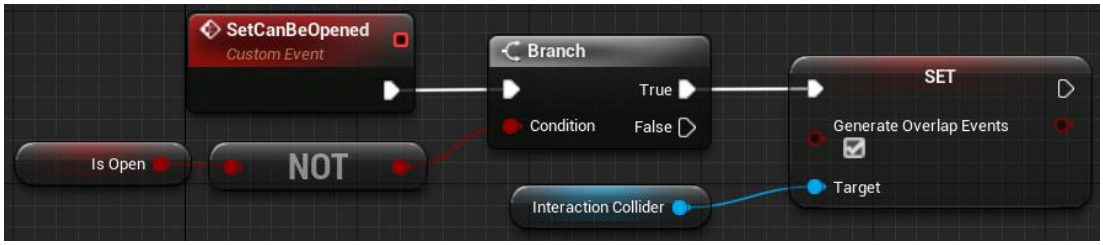

Figura 143. Event SetCanBeOpened

6.3 Moviment del Jugador

El sistema de moviment es basa en tot allò que el jugador pot fer per desplaçar-se. S'inclouen aquí el córrer i saltar, a més dels moviments especials, com esquivar o el doble salt. Aquesta és una de les mecàniques més importants del joc, i en la què el jugador haurà de dedicar més temps per millorar, ja que és increïblement important tant en la part d'exploració com la de combat. També és una de les parts més complexes, ja que tots els tipus de moviments es relacionen entre ells i amb la resta de mecàniques.

6.3.1 CharacterMovement

El *CharacterMovement* és un component d'UE4 que està acoblat als Blueprint de tipus *Character*, i que els atorga la possibilitat de moure's (caminar, saltar, ...) amb una configuració complexa (acceleracions, fricció, ...).

Aquest component porta valors per defecte ja assignats, però es poden canviar per a obtenir una configuració i tipus de moviment desitjat. Justament això és una de les primeres coses que es va fer, ja que el moviment per defecte del personatge no s'hi assemblava gens al requerit per les característiques del joc. Al descarregar el personatge del *Paragon*, alguna configuració ja venia donada, però aquestes només eren per fer que el personatge funcionés de forma normal. Aquí es comentaran només les rellevants que canvien per complet el moviment de la configuració inicial.

Els canvis que s'han fet, han estat pensats per a generar un moviment ràpid i molt responsiu a les indicacions del jugador.

6.3.1.1 General Settings

Aquest apartat fa referència a la configuració dels atributs generals del personatge. Primer es va canviar l'escala de la gravetat, de 1 a 3. D'aquesta manera, el personatge torna al terra molt més ràpid després de saltar. A més, es va canviar l'acceleració del personatge a un valor enorme, per fer que el personatge arribés a la seva velocitat de moviment màxima a l'instant. D'altra manera, el personatge trigava un temps notable en arribar a la velocitat desitjada. Per a entendre el canvi, el valor inicial era 2048, passant a ser 1000000. Veure Figura 144.

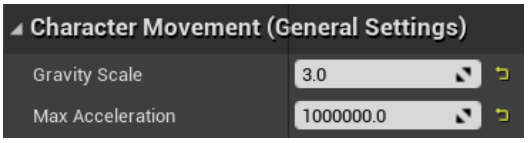

Figura 144. CharacterMovement - General Settings

6.3.1.2 Walking

Aquesta és la configuració del personatge quan es mou tocant el terra. Aquí es van canviar tres atributs. El primer és el grau d'inclinació pel què el personatge pot caminar. S'ha assignat 45 graus, tot i que el valor per defecte era molt similar. El següent és la fricció que fa el terra sobre el personatge. Bàsicament determina el temps que triga el personatge en frenar per complet quan es deixa de fer que es mogui. Vindria a ser el contrari a l'acceleració de l'apartat anterior. Aquí els valors són molt inferiors. L'original era 8, i 30 el desitjat. L'últim canvi és el de la velocitat màxima del personatge, que ha passat de 600 a 1000. Veure Figura 145.

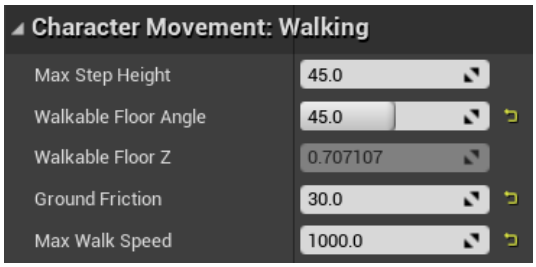

Figura 145. CharacterMovement - Walking

6.3.1.3 *Jumping/Falling*

Aquesta és la configuració del personatge en tot el què es tracta quan no toca el terra. El primer canvi és la velocitat inicial en el salt. Aquest valor va molt de la mà amb el de la gravetat mencionat anteriorment. Passa de 420 a 1250. El segon canvi seria l'equivalent a la fricció en el terra, però a l'aire. Bàsicament, fa que el personatge s'aturi en el punt en què està al moment en què es deixen de prémer les tecles de moviment. Originalment, aquest valor era 0, fent que el personatge funcioni amb inèrcies, però aquí no interessava, així que s'ha posat el valor de 40000, eliminant per complet la inèrcia en el jugador. El tercer canvi és el control que el jugador té sobre el personatge quan aquest no toca el terra. És a dir, la possibilitat de fer que el personatge canviï de direcció a l'aire. Passa de 0.05 a 0.2. Veure Figura 146.

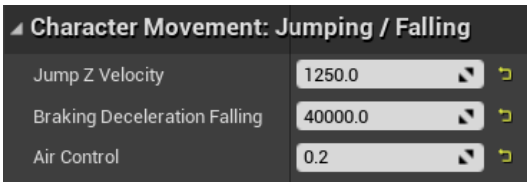

Figura 146. CharacterMovement - Jumping/Falling

6.3.2 Moviment

Aquest apartat és per comentar quines condicions hi ha per a que el personatge respongui al moviment que se li hagi indicat. Primer de tot, cal que l'animació d'inici de nivell hagi acabat. Aquesta és simplement això, una animació que s'executa a l'iniciar la partida. A més, per eliminar el *Drift* dels *joysticks* dels comandaments (error comú, en què s'envia la senyal de què s'està canviat el *joystick* de la seva posició en repòs, quan no és així), fem que el moviment del *joystick* sigui de mínim un 10%. Les altres tres condicions són: que el personatge no estigui interactuant (és a dir, en un diàleg), que no estigui fent un *Dash*, i que no estigui atacant. Totes tres situacions s'aniran explicant en apartats posteriors. Veure Figura 147.

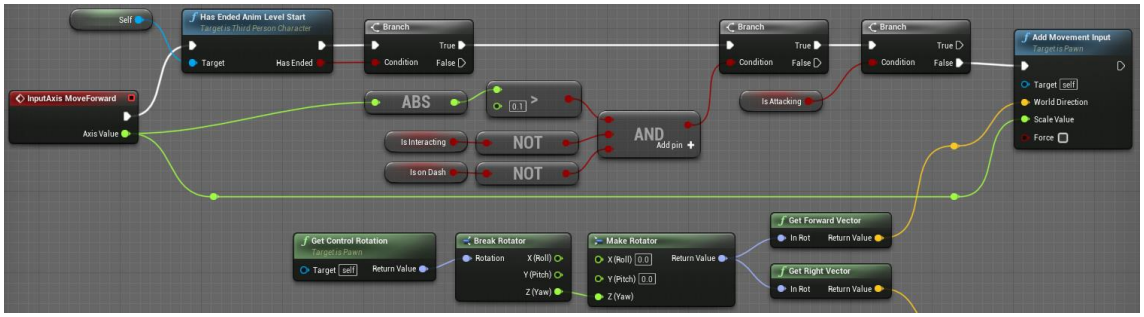

Figura 147. Moviment - Endavant/Endarrere

6.3.3 Salt

El salt del jugador està fet de forma que es pugui fer un doble salt, és a dir, tornar a saltar quan ja està a l'aire. Aquesta és una mecànica molt utilitzada en molts tipus de joc, ja que dona un punt extra a la mobilitat del jugador.

El codi és bastant senzill. Quan es prem el botó de salt, el primer que fa és comprovar que no estigui interactuant. Després, crida el bloc anomenat *Do N*. Aquest és un *Blueprint* propi d'UE4, que permet executar el codi següent les N vegades indicades. A més, es pot reiniciar el comptador de vegades amb el *Reset*. Veure Figura 148.

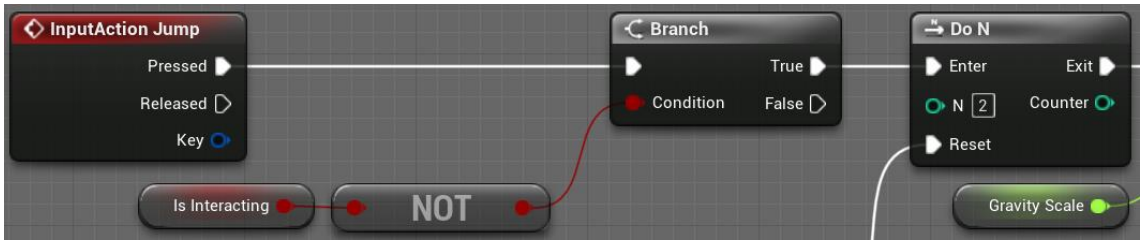

Figura 148. Salt - Part 1

En cas que es pugui executar, reinicia la gravetat al valor indicat abans, fa que les variables *doneFirstAirAttack* i *doneFirstAirDash* estiguin a fals, es reinicia la fricció a l'aire, es crida la funció *StopAttackingAndDash* i, per fi, s'impulsa al jugador cap amunt amb la funció *LaunchCharacter*.

Els valors de la gravetat i la fricció a l'aire es reinicien perquè en el codi que s'explica més endavant, aquests canvien per necessitat. Pel què fa a les variables dites, s'utilitzen en un altre lloc i cal reiniciar-les. La funció *StopAttackingAndDash* s'explica a l'Apartat 6.3.5, i la funció *LaunchCharacter* és una funció interna que genera un impuls en el personatge. Veure Figura 149.

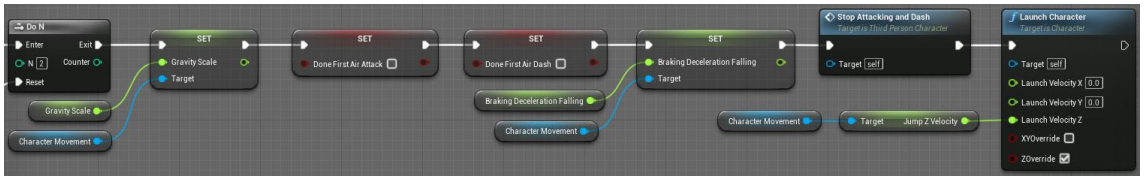

Figura 149. Salt - Part 2

Per reiniciar el comptador del *Do N*, s'utilitza l'*Event OnLanded*, que es crida quan el personatge toca el terra després d'haver estat a l'aire. Aquest reinicia la fricció a l'aire i la variable *doneFirstAirAttack*, igual que abans. Al final, s'uneix al *Reset* del *Do N*. Veure Figura 150.

Figura 150. Salt - Part 3

6.3.4 Dash

El *Dash* és un impuls que fa el personatge cap a la direcció a la que està mirant per tal d'esquivar un atac o arribar a algun lloc. Aquest té un comportament molt lligat al salt, per tal de generar un sistema de moviment fluït i complex. Aquest impuls es pot fer de forma il·limitada quan el personatge es troba al terra, però a l'aire no funciona així. Quan el personatge està a l'aire, el *Dash* només es pot fer després de cada salt.

El *Dash* s'activa al prémer el botó configurat. Per a generar l'impuls desitjat hi ha diverses condicions, les quals són: que no estigui interactuant, que no estigui executant un *Dash*, que estigui movent-se pel terra, o que encara no s'hagi fet el *Dash* a l'aire (el corresponent a l'últim salt). Si tot va bé, es crida la funció *StopAttackingAndDash*, i s'assigna cert a la variable *isOnDash*, que indica que s'està fent un *Dash*. Veure Figura 151.

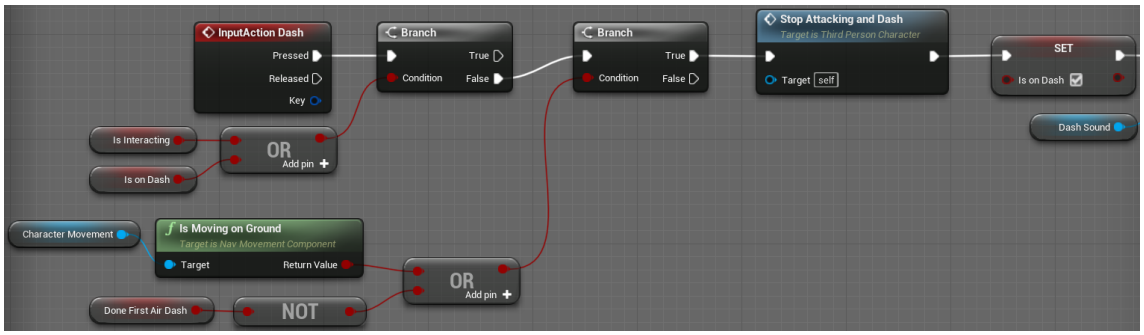

Figura 151. Dash - Part 1

Fet això, s'activa el so del *Dash*, i la animació corresponent. A més, segons si el personatge està al terra o no, es creen dues branques. Veure Figura 152.

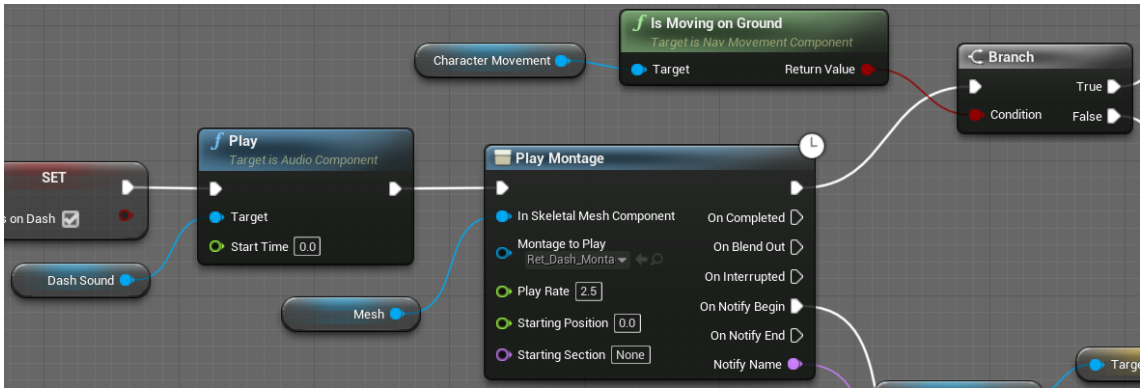

Figura 152. Dash - Part 2

En cas que el jugador estigui al terra, es posa l'escala de la gravetat a 50, per fer que el personatge estigui tota l'estona enganxat al terra. En cas contrari, s'atura la velocitat vertical i es posa la gravetat a 0 per a que el personatge es mantingui a aquella altura. Al final, tots dos camins s'uneixen. Veure Figura 153.

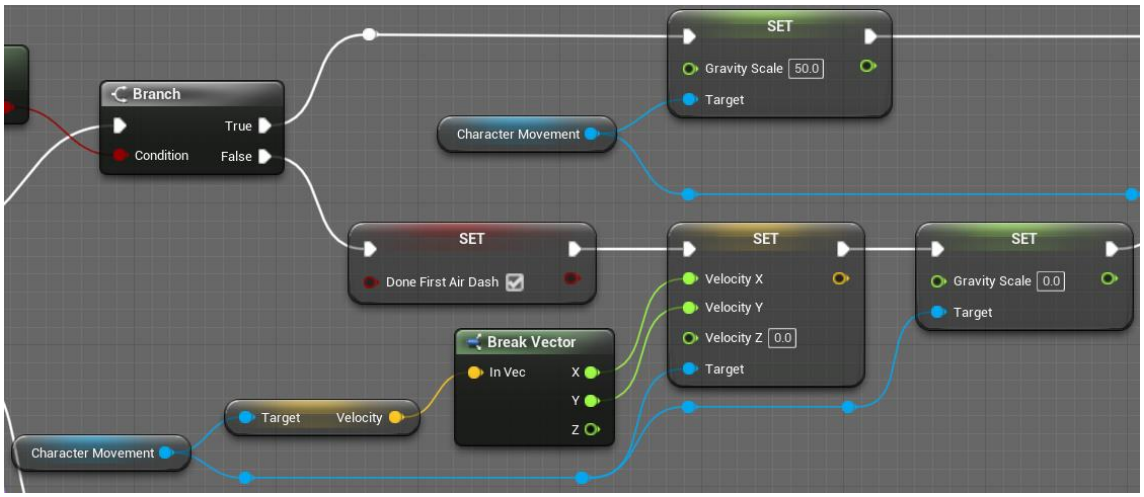

Figura 153. Dash - Part 3

Per acabar, es posa la fricció del terra a 5, es reinicia la fricció de l'aire, i es genera un impuls cap a on mira el jugador amb la funció *LaunchCharacterTo* (Apartat 6.3.6). Veure Figura 154.

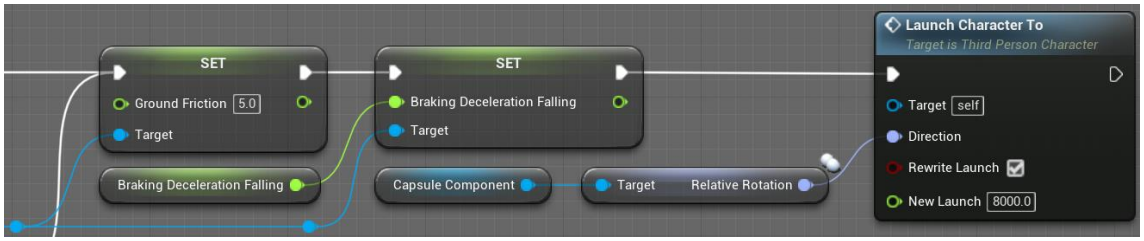

Figura 154. Dash - Part 4

A més d'aquesta execució, l'animació té una notificació al final que activa un altre fil. La notificació és *ResetGravity*, i serveix per reiniciar el valor de la gravetat, el de la fricció, i posar *isOnDash* a fals. Veure Figura 155.

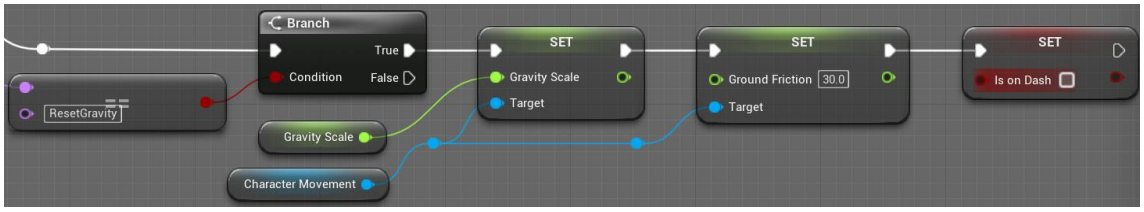

Figura 155. Dash - Part 5

6.3.5 StopAttackingAndDash

Aquesta és una funció que atura tant el *Dash* com els atacs, per a poder executar una altra cosa. Per exemple, com ja s'ha mencionat a l'Apartat 6.3.3, s'utilitza aquesta funció per aturar el *Dash* i executar un salt. Això dona molta flexibilitat al moviment, podent cancel·lar execucions cada vegada que ho necessitem.

La funció *StopAttackingAndDash* té dues branques, segons si s'està atacant, o no. En cas que sigui que no, i que s'estigui executant un *Dash*, atura l'animació, i posa la variable *isOnDash* a fals. Veure Figura 156.

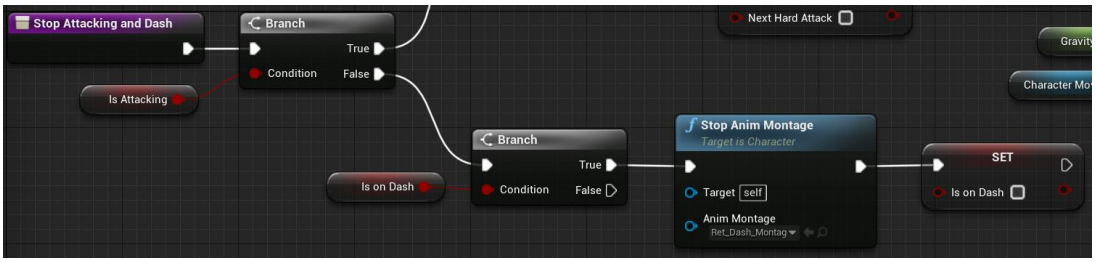

Figura 156. StopAttackingAndDash - Part 1

En cas que sí s'estigui atacant, es posen totes les variables d'atac a fals, es desactiva l'animació d'atac, i es reinicia la gravetat. Veure Figura 157.

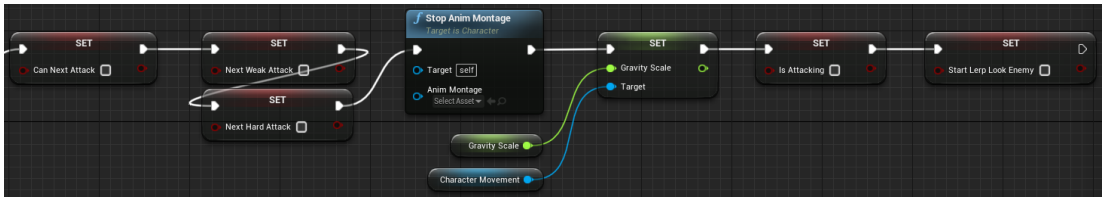

Figura 157. StopAttackingAndDash - Part 2

6.3.6 LaunchCharacterTo

Aquesta funció genera un impuls sobre el personatge en la direcció indicada. La força de l'impuls és una de configurada per defecte o la entrada, si és que el *rewriteLaunch* està a cert. Veure Figura 158.

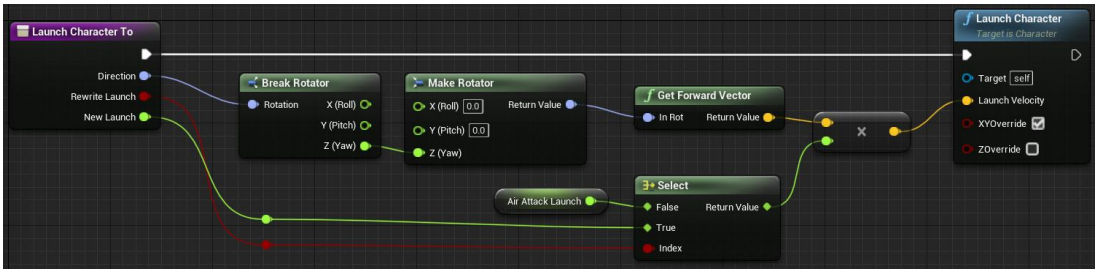

Figura 158. LaunchCharacterTo

6.3.7 Set Deceleration = 0 At Colliding Landscape

En aquesta part, es fa que el personatge tingui una fricció de 0 quan està caient mentre toca alguna part del món, com per exemple, una muntanya. Això es fa perquè, en cas contrari, el personatge aniria caient molt poc a poc, i això no era correcte. Per fer-ho, cada vegada que el personatge xoca, es comprova que estigui caient, que no estigui atacant, que no estigui fent un *Dash*, i que ha tocat l'objecte que forma el terra. Si tot es compleix, es posa la fricció a 0. Veure Figura 159.

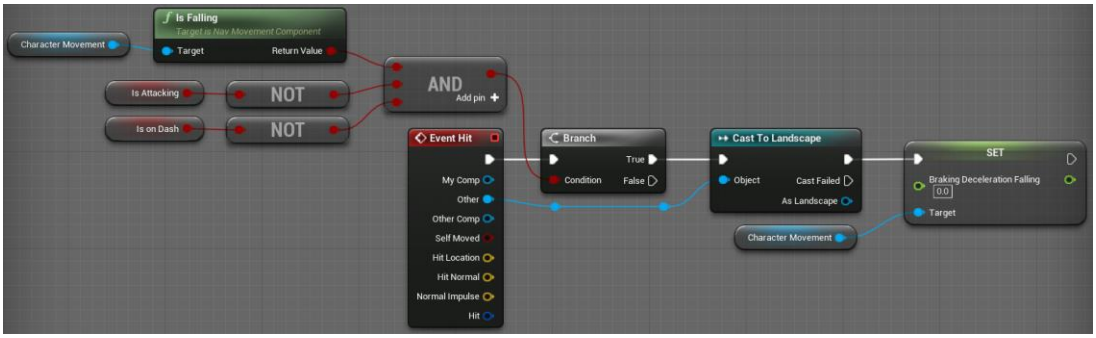

Figura 159. Set Deceleration = 0 At Colliding Landscape

6.4 Estadístiques del Jugador

En aquest cas el personatge té quatre estadístiques bàsiques: vida, nivell, atac i velocitat. També existeix una cinquena estadística que només serveix per quantificar el progrés del nivell actual, que és l'experiència. Cal destacar que només la vida es pot augmentar amb el nivell, ja que l'atac i la velocitat es milloren exclusivament amb les runes.

6.4.1 Vida

Aquesta estadística quantifica el mal que es pot rebre abans de morir. Per a aquesta implementació es parteix de 300 de vida base. A més, amb cada nivell augmenta 60, fins a un màxim de 600 extra. Finalment, es pot obtenir 300 punts de vida extra si el jugador utilitza tres runes èpiques de doble vida, veure Apartat 6.1.2.2. Existeixen tres funcions per modificar-la, explicades als següents apartats. Aquestes permeten augmentar la vida base, curar o rebre mal.

6.4.1.1 AddBaseLife

Aquesta funció permet augmentar la vida base. Es crida quan es puja de nivell i rep com a paràmetre l'extra de vida, en aquest cas són 60. Veure Figura 160.

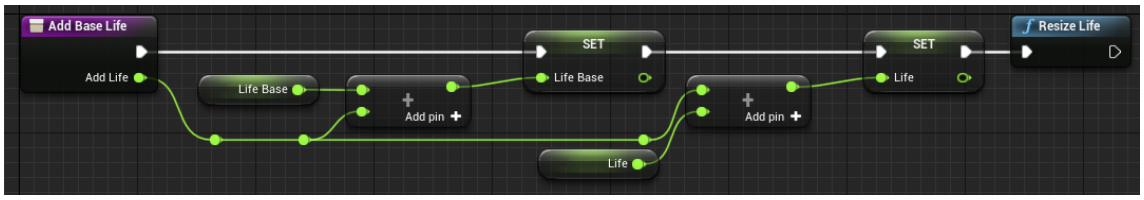

Figura 160. AddBaseLife

6.4.1.2 AddLife

En aquest cas es rep una quantitat de vida i s'afegeix a la vida del jugador per curar-lo. Si es supera el màxim, s'agafa la suma de la vida base i la vida extra. Veure Figura 161.

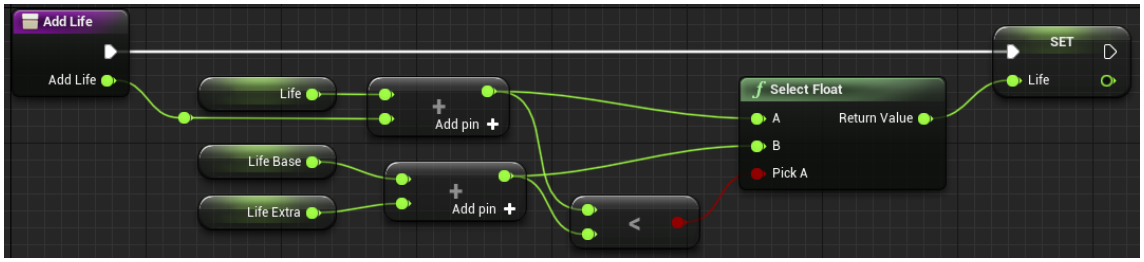

Figura 161. AddLife

6.4.1.3 ReceiveDamage

Quan un enemic ataca al jugador, es crida a aquesta funció que resta l'atac de l'enemic a la vida, tenint en compte que el mínim ha de ser zero. Figura 162.

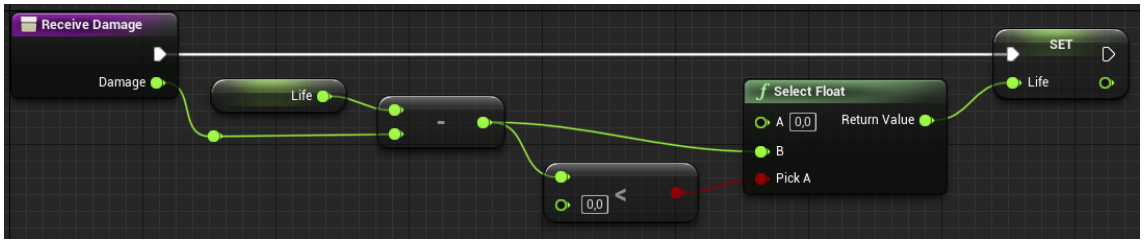

Figura 162. ReceiveDamage – Part 1

Si la vida del jugador quan rep un atac és igual o inferior a 0, es mostra el missatge per pantalla de que ha mort (Figura 47) abans de carregar el menú d'inici. Figura 163.

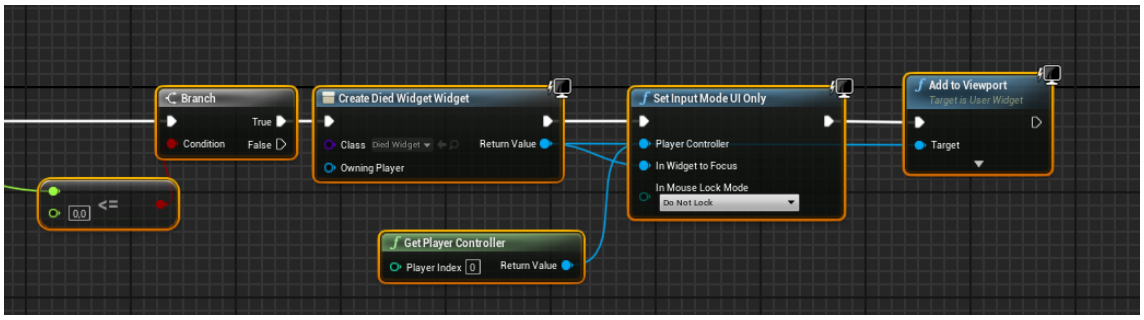

Figura 163. ReceiveDamage – Part 2

6.4.2 Nivell

Aquesta estadística permet augmentar la vida base del jugador. Cal destacar que s'ha creat un sistema progressiu on cada cop es requereix més experiència per pujar de nivell.

6.4.2.1 AddExperience

Cada cop que un enemic mor, es crida a la funció *AddExperience*, passant com a paràmetre l'experiència. Es suma a l'actual i, en cas que es superi el valor d'*ExperienceNextLevel*, es puja de nivell amb *LevelUp*, explicat a l'Apartat 6.4.2.3. Veure Figura 164.

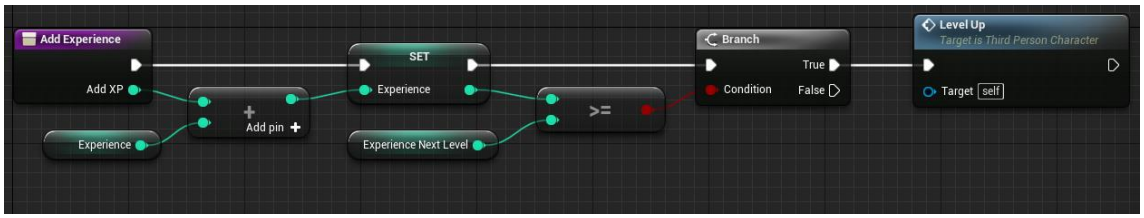

Figura 164. AddExperience

6.4.2.2 CalculateExperienceForLevel

Cada cop que es puja de nivell, cal recalcular l'experiència que caldrà per pujar al següent. Per fer-ho, es fa utilitzant el valor de l'actual. En aquest cas s'ha optat per fer que aquest valor augmenti d'una forma no lineal. Per aquest motiu es multiplica el nivell per 10 i se li suma el valor al quadrat. Veure Figura 165.

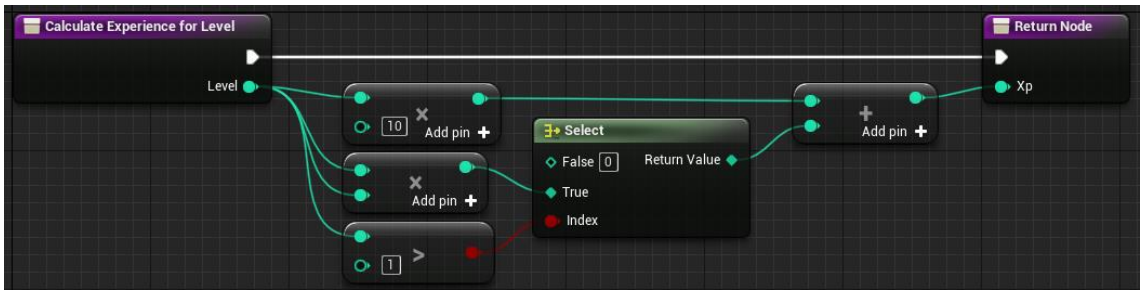

Figura 165. CalculateExperienceForLevel

6.4.2.3 LevelUp

La tasca que realitza aquesta funció és augmentar el nivell, recalcular l'experiència necessària pel nou nivell amb *CalculateExperienceForLevel*, explicat a la secció anterior, i afegir vida al jugador amb *AddBaseLife*, esmentat a l'Apartat 6.4.1.1. Veure Figura 166.

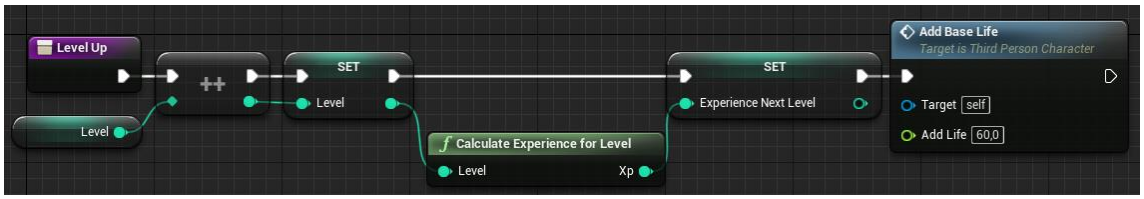

Figura 166. LevelUp

6.4.3 Atac

L'atac és el que determina quants punts de vida es resten a l'enemic. S'inicia amb 100 punts d'atac, que poden arribar a un màxim de 400 si s'utilitzen les tres runes èpiques de doble atac, veure Apartat 6.1.2.2. En aquest cas existeix una única funció que permet rebre el mal per part del jugador. Pel prototip desenvolupat s'ha optat per sumar totes les variables d'atac que conté el personatge. Veure Figura 167.

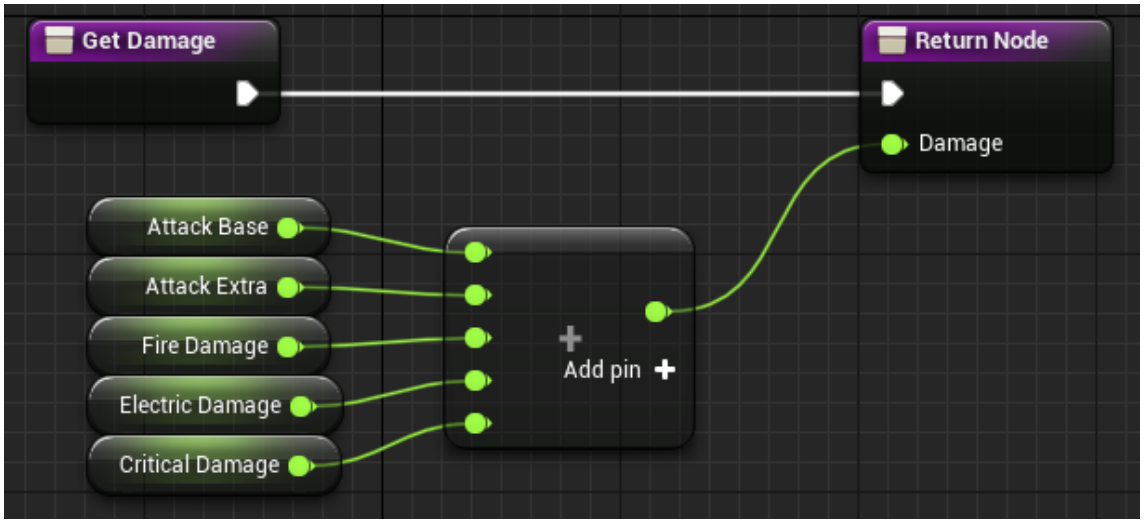

Figura 167. GetDamage

6.4.4 Velocitat

La velocitat també es pot augmentar només amb les runes. Parteix d'un mínim de 1000 punts i arriba a un màxim de 1562.5, amb les tres runes èpiques de doble velocitat (veure Apartat 6.1.2.2). Igual que amb la secció anterior, només existeix una única funció que es la que permet alterar la velocitat de moviment del jugador. Per això es suma el valor entrat al *MaxSpeed*, una variable interna del *CharacterMovement*. Cal destacar que el valor d'entrada pot ser tant negatiu com positiu, per restar o augmentar vida, respectivament. Veure Figura 168.

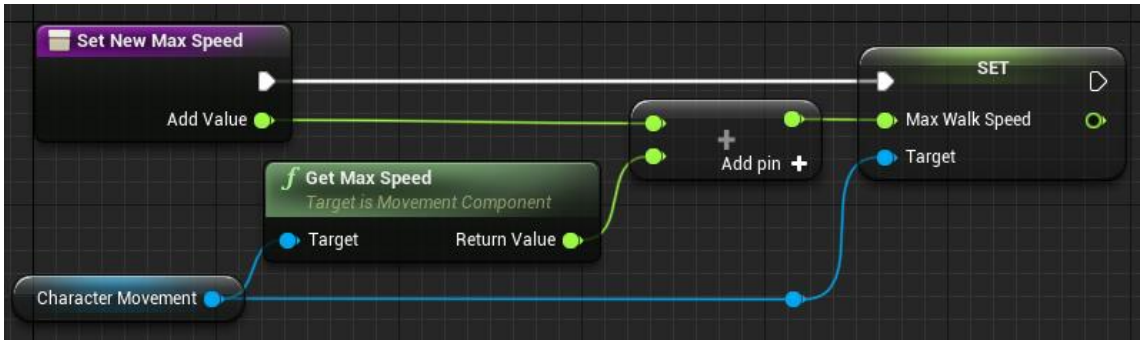

Figura 168. SetNewMaxSpeed

6.5 HUD

El HUD és el component que mostra la vida, el nivell, i l'experiència actual del jugador.

6.5.1 LifeExperienceHUD

Aquest és un *Widget* que conté la informació dita. L'apartat visual es pot veure a la Figura 169.

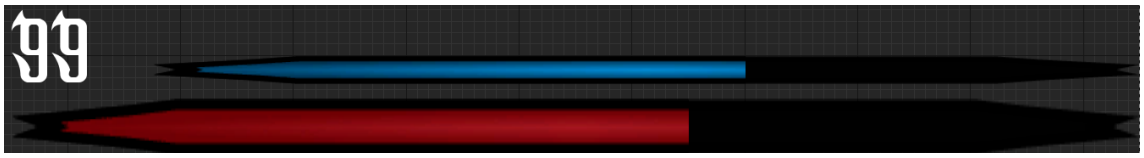

Figura 169. LifeExperienceHUD

Tant les barres com el número tenen una funció que determinen el percentatge o el valor a mostrar. El nivell s'obté amb la funció *GetText*, que obté el nivell del *ThirdPersonCharacter* i el converteix a text. Veure Figura 170.

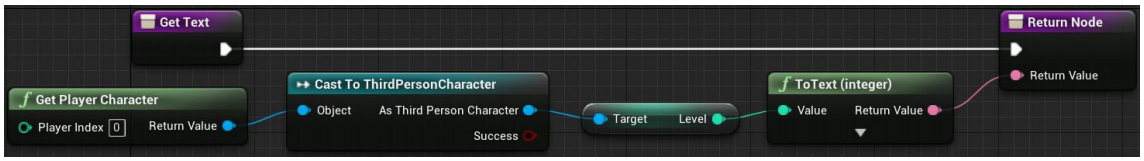

Figura 170. GetText

El percentatge de la barra de vida s'obté amb la funció *GetLifePercent*, que calcula el percentatge que forma la vida actual sobre la màxima. Veure Figura 171

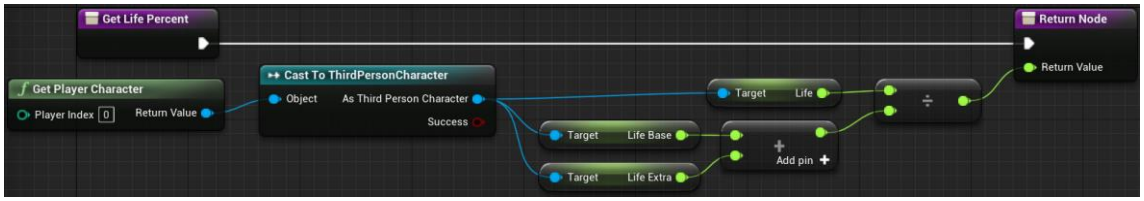

Figura 171. GetLifePercent

El percentatge de la barra d'experiència es calcula a la funció *GetExperiencePercent*. Aquest s'obté al dividir l'experiència obtinguda des de l'últim nivell per la necessària per passar de nivell. S'utilitza la funció *CalculateExperienceForLevel* explicada a l'Apartat 6.4.2.2 per saber tant l'experiència del nivell anterior, com la del posterior. Veure Figura 172.

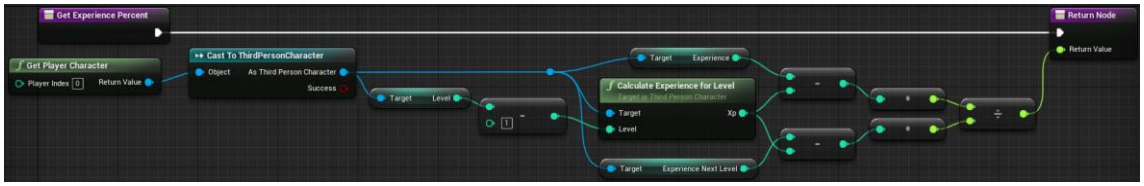

Figura 172. GetExperiencePercent

6.5.2 BPFL_LifeExperience

Aquesta és la llibreria que conté les funcions necessàries per al *LifeExperienceHUD*. Només n'hi ha dues: una que inicialitza el *Widget*, i una altra que li canvia la mida a la barra de vida.

6.5.2.1 *InitializeLifeExperience*

Aquesta funció s'encarrega de crear el *Widget*, assignar-li al *ThirdPersonCharacter*, i afegir-ho a la pantalla. Veure les figures Figura 173 i Figura 174. Aquesta es crida des del *BeginPlay* del *ThirdPersonCharacter*.

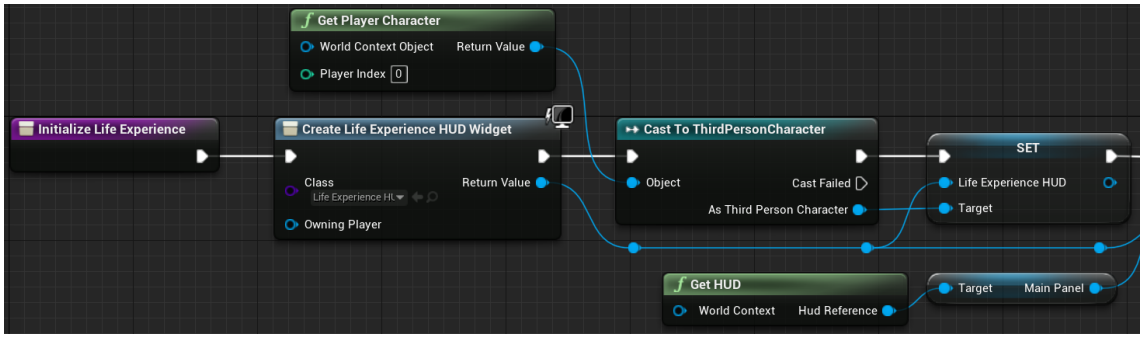

Figura 173. InitializeLifeExperience - Part 1

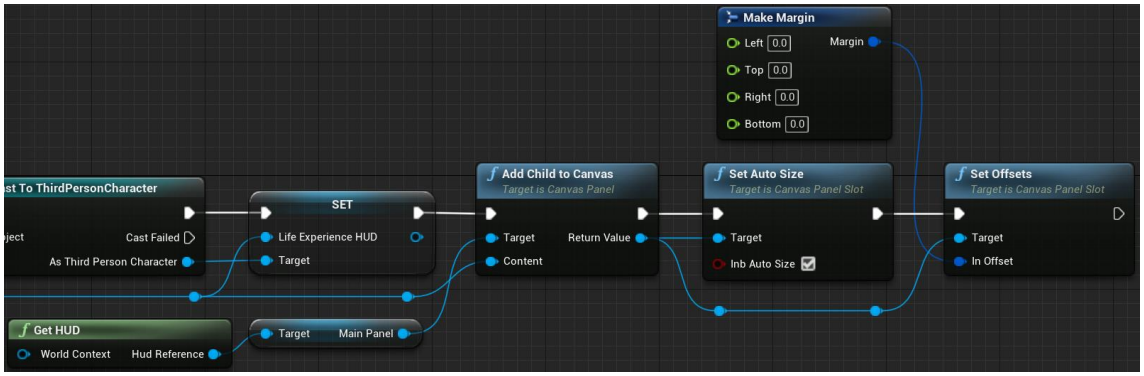

Figura 174. InitializeLifeExperience - Part 2

6.5.2.2 *ResizeLife*

Aquesta funció s'encarrega de posar la llargada de la barra de vida segons la vida màxima del jugador. Veure Figura 175.

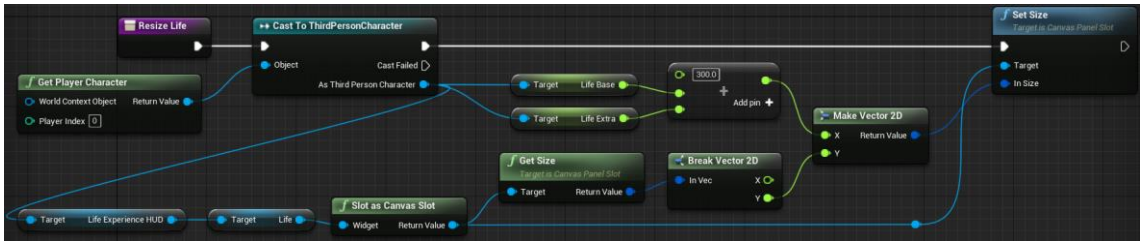

Figura 175. ResizeLife

6.6 Sistema de Combat

El sistema de combat es basa en combinació d'atacs forts i febles, podent fer la combinació que més interessi en cada combat. Per fer-ho, hi ha una primera execució d'atac quan es prem un dels botons d'atac. A partir d'aquí, es poden fer seqüencies d'atacs si es segueixen prement els botons d'atac abans que s'acabi l'atac anterior. Les seqüencies poden començar per un atac fort o un de feble, però el codi és exactament el mateix, així que només s'explicarà la seqüència que comença amb l'atac feble.

6.6.1 CombosCollapsed

Per començar a atacar, és essencial que no s'estigui interactuant ni fent un *Dash*. Si encara no s'està atacant, comença la seqüència. Primer s'executa la funció *StopAttackingAndDash* i, després, es fan dues branques segons si s'està al terra, o no. Veure Figura 176.

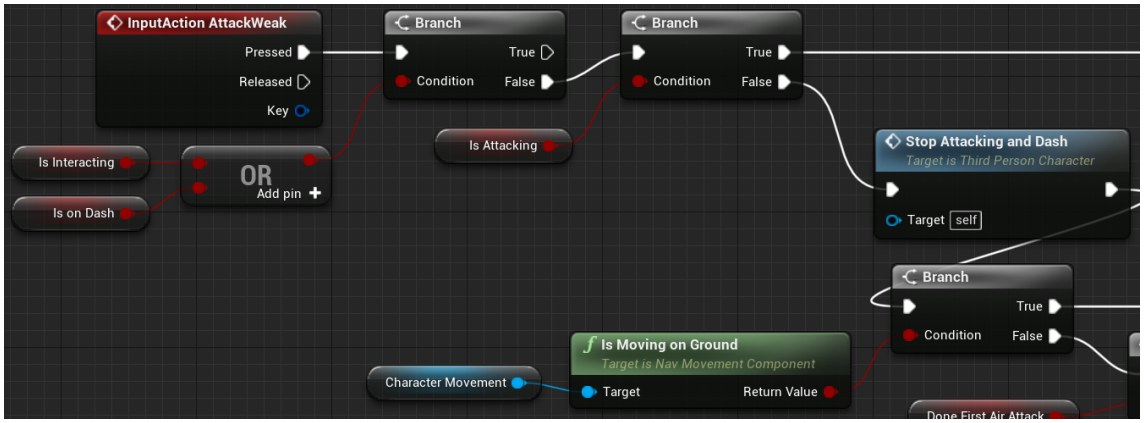

Figura 176. CombosCollapsed - Part 1

En cas que s'estigui a l'aire, es posa a cert la variable *doneFirstAirAttck*, que indica si s'ha iniciat una seqüència d'atac a l'aire. Això es fa perquè només es pot fer una seqüència per cada salt. També es posen la gravetat i la velocitat del personatge a 0. Totes dues branques s'uneixen després d'això. Veure Figura 177.

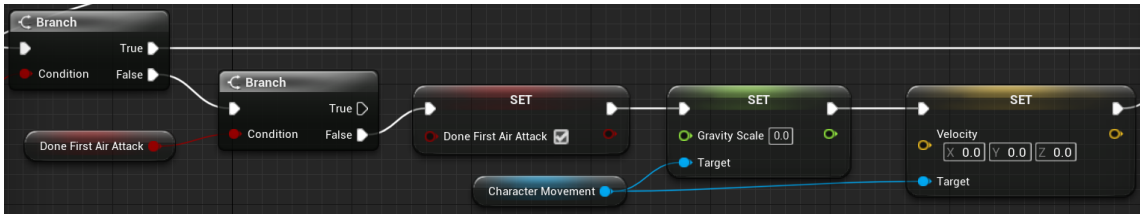

Figura 177. CombosCollapsed - Part 2

A l'unir-se, es posa a cert la variable *isAttacking*, que indica si està en una seqüència d'atac, i a fals les variables *nextWeakAttack* i *nextHardAttack*, que serveixen per a seguir una seqüència d'atac per un dels dos camins possibles. Just després s'executa el què seria l'atac i tota la lògica que requereix. Com es pot veure a la Figura 178, hi ha dos camins disponibles, gestionats dins aquesta funció per les variables *nextWeakAttack* i *nextHardAttack*.

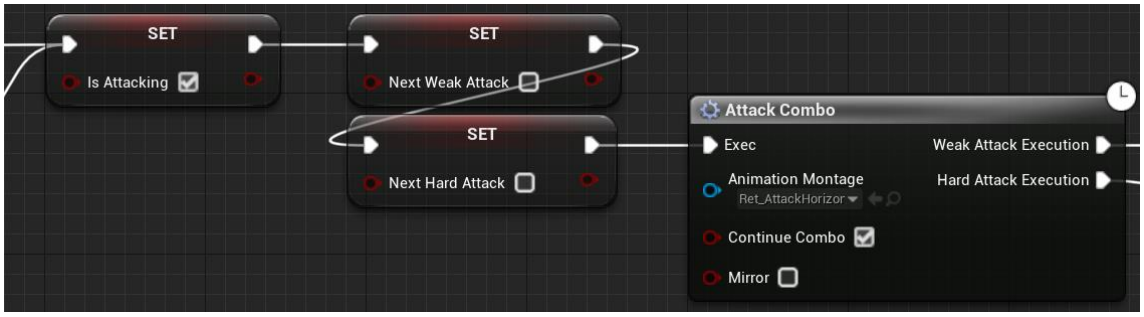

Figura 178. CombosCollapsed - Part 3

Al final d'una seqüència d'atac, sempre es restaura el valor de la gravetat i es posa *isAttacking* a fals. Veure Figura 179.

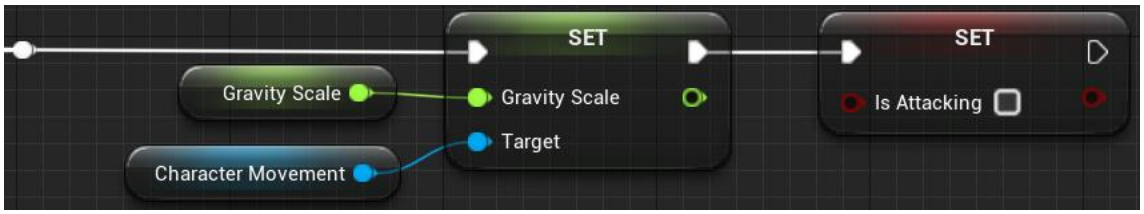

Figura 179. CombosCollapsed - Part 4

En el cas en què s'estigués fent un atac a l'hora de prémer un dels botons d'atac, es mira si es pot executar l'ordre d'encadenar el següent atac en la seqüència (ho determina la variable *canNextAttack*). Si és així, es comprova que no s'hagués enviat ja l'ordre, i si no és així, s'assigna cert al camí de la seqüència que es seguirà (*nextWeakAttack* o *nextHardAttack*, segons el botó pres). Veure Figura 180.

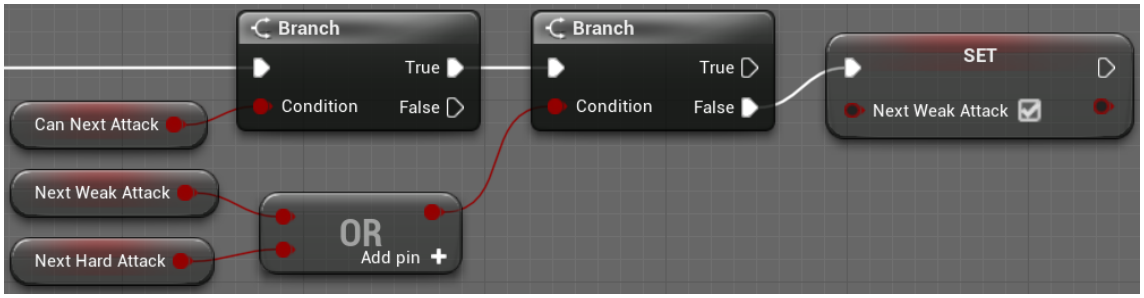

Figura 180. CombosCollapsed - Part 5

6.6.2 AttackCombo

Aquesta és la funció que executa tota la lògica d'un atac. Té diversos paràmetres d'entrada: l'animació, el *continueCombo* (serveix per determinar si es pot seguir la seqüència o no, independentment de si es vol) i el *mirror*. El primer que es fa és utilitzar aquest paràmetre. Serveix per girar el personatge per fer que ataqui amb l'espasa esquerra o la dreta. Després, es creen dues branques segons si es pot seguir la seqüència o no. Veure Figura 181.

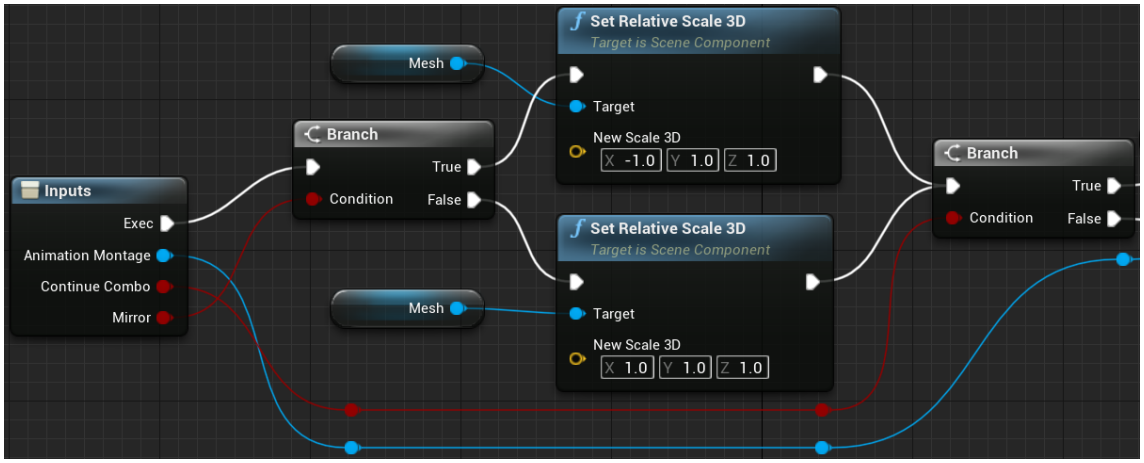

Figura 181. AttackCombo - Part 1

Si no es pot seguir la seqüència, s'executa l'animació d'entrada i, a l'acabar l'execució, es reinicien la gravetat i es marca que ja no s'està atacant. En cas contrari, s'executa

l'animació entrada i es comproven les variables *nextWeakAttack* i *nextHardAttack*, per saber si l'execució ha de seguir per alguna de les dues branques, o ha d'acabar. Si *nextWeakAttack* és cert, l'execució segueix per la branca del següent cop feble, si és fals i *nextHardAttack* és cert, l'execució segueix per la branca del següent cop fort. En cas que tots dos siguin fals, reinicia la gravetat i es marca que ja no s'està atacant. Sigui com sigui, just després d'inicial l'animació, s'uneixen les branques. Tot aquest procés es pot veure a la Figura 182.

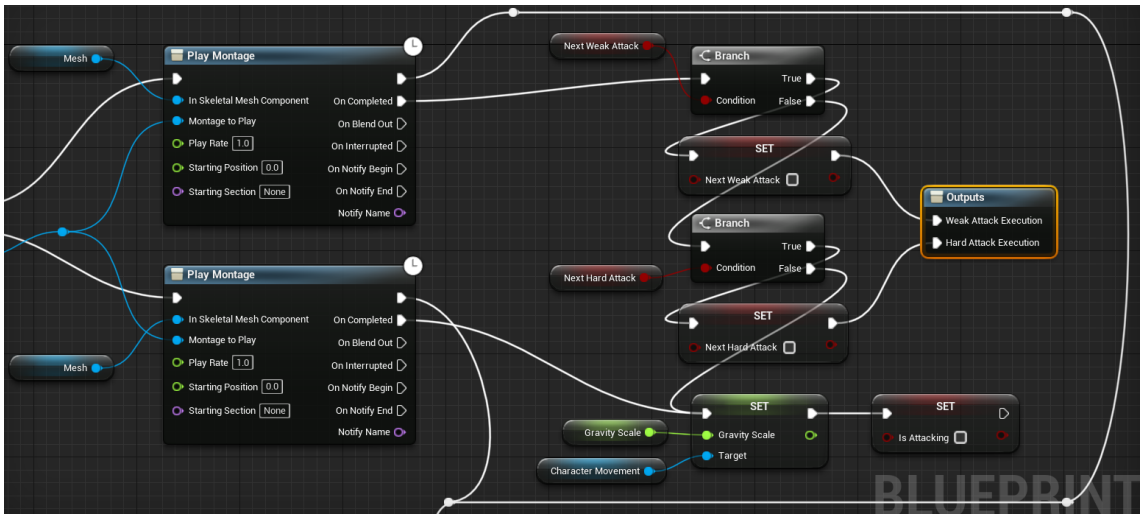

Figura 182. AttackCombo - Part 2

Per acabar, es reinicia la fricció a l'aire i s'executa la funció *AttackComboLaunchAndStartLookEnemy*. Veure Figura 183.

Figura 183. AttackCombo - Part 3

6.6.3 AttackComboLaunchAndStartLookEnemy

Aquesta funció té 2 objectius. El primer és iniciar un procés per a que el jugador es giri cap a l'enemic de forma progressiva. Aquest procés s'explica a l'Apartat 6.6.5. L'altre objectiu és impulsar el jugador cap a l'enemic quan està a l'aire. Aquestes funcions s'apliquen segons si el personatge està al terra o no, tal i com es veu a la Figura 184.

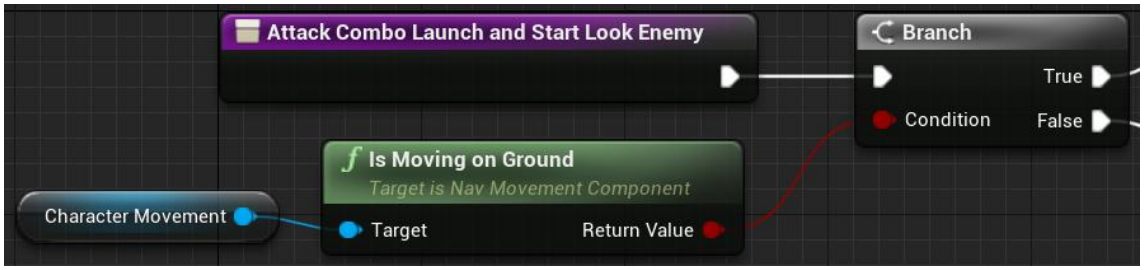

Figura 184. AttackComboLaunchAndStartLookEnemy - Part 1

Si el jugador està al terra, es comprova que hi hagi un *currentEnemy* vàlid (enemic a qui atacar) i que hi hagi valors al *collidingEnemies* o hi hagi un enemic marcat. Si es compleix, es posa *startLerpLookEnemy* a cert i *alphaLerpLookEnemy* a 0. Veure Figura 185.

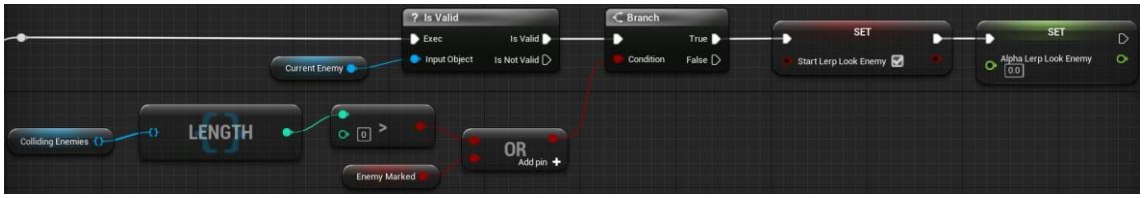

Figura 185. AttackComboLaunchAndStartLookEnemy - Part 2

En cas contrari, es comprova si hi ha un *currentEnemy* vàlid i, si no hi és, s'impulsa el jugador cap endavant amb la funció *LaunchCharacterTo*, com es veu a la Figura 186.

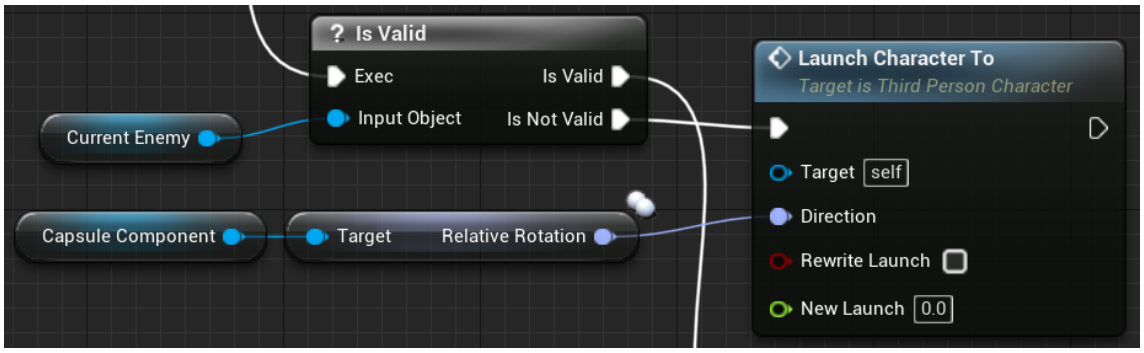

Figura 186. AttackComboLaunchAndStartLookEnemy - Part 3

En cas que sí sigui vàlid, es calcula la distància entre el jugador i l'enemic. Si és menor al doble de la mínima distància configurada, no s'executarà l'impuls. En cas que sigui major, es mirarà si és menor a 300. Si ho és, es reescriurà el valor de l'impuls del *LauchCharacterTo* amb el valor de la divisió de la distància entre 300 i multiplicat per el valor d'impuls configurat. Veure Figura 187. D'aquesta forma, l'impuls serà menor com més a prop de l'enemic s'estigui, i deixarà d'impulsar si ja s'està molt a prop. La direcció d'impuls és cap a l'enemic, i es calcula a la Figura 188.

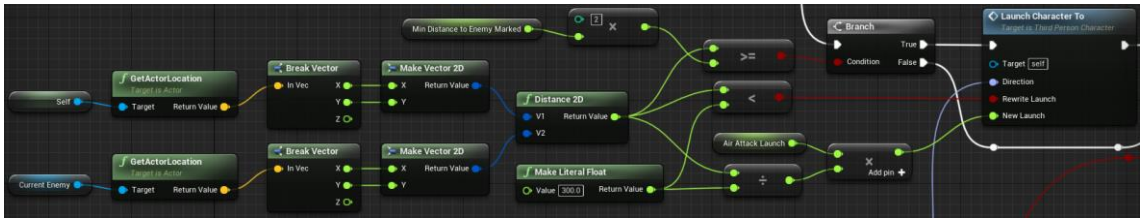

Figura 187. AttackComboLaunchAndStartLookEnemy - Part 4

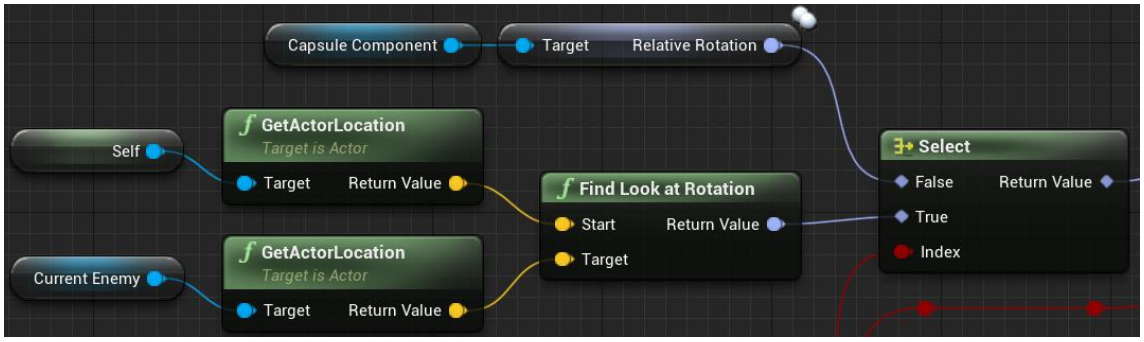

Figura 188. AttackComboLaunchAndStartLookEnemy - Part 5

Es faci o no l'impuls, si es compleix la mateixa condició que a la Figura 185, es posa *startLerpLookEnemy* a cert i *alphaLerpLookEnemy* a 0.

6.6.4 SetGravityOnAttacking

Aquesta és una funció que s'executa a cada Tick del rellotge del *ThirdPersonCharacter*. El primer que fa és comprovar si s'està atacant. Si s'està fent, es posen la gravetat i la velocitat vertical a 0. Veure Figura 189.

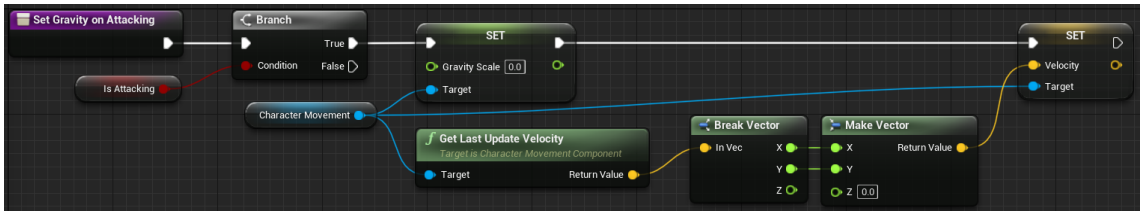

Figura 189. SetGravityOnAttacking

6.6.5 CharacterLookAtEnemy

Aquesta és una funció que s'executa a cada *Tick* del rellotge del *ThirdPersonCharacter*. El què fa és que el personatge es giri per mirar a l'enemic al què està atacant de forma suau.

Per fer-ho, primer de tot es comprova que *startLerpLookEnemy* sigui cert. Aquesta variable indica exactament si aquest procés s'ha d'executar o no, i com s'ha vist en aquest mateix apartat, l'activen els atacs. Just després, es comprova que hi hagi un enemic objectiu vàlid amb el *currentEnemy*, i que hi hagi algun enemic a la llista *collidingEnemies* o hi hagi un enemic marcat amb l'*enemyMarked*. Si tot això es compleix, es calcula la rotació què s'assignarà al personatge. La rotació dependrà del *alphaLerpLookAtEnemy*, que indica en quin percentatge ha d'estar el personatge mirant a l'enemic respecte la posició inicial. Després d'assignar la rotació, es canvia l'*alpha* afegint el triple del temps que ha passat des de l'últim *Tick*. Veure les figures: Figura 190 i Figura 191.

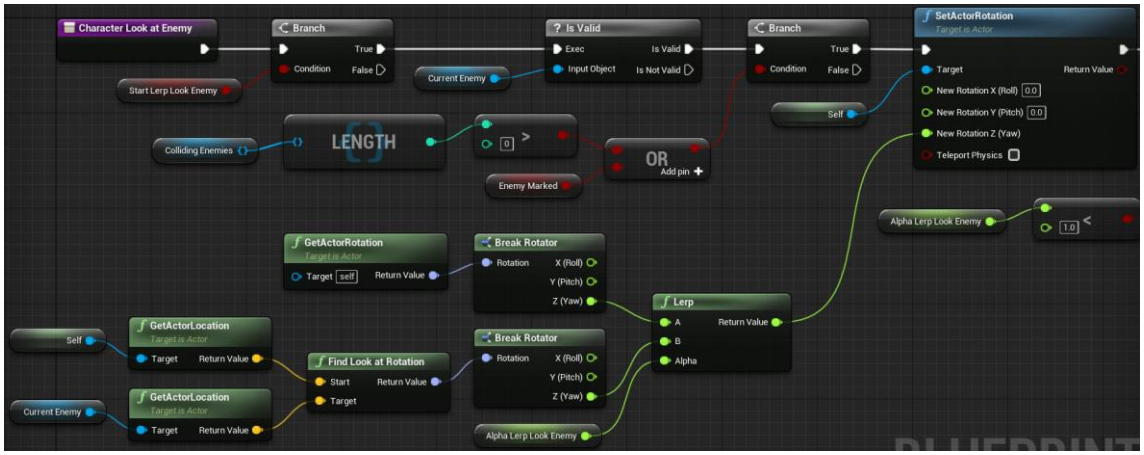

Figura 190. CharacterLookAtEnemy - Part 1

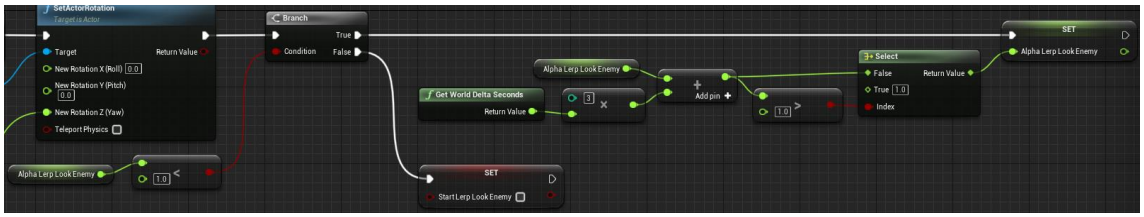

Figura 191. CharacterLookAtEnemy - Part 2

6.6.6 NextAttack_AnimNotify

Aquest Blueprint és una notificació que s'assigna a les animacions i que executa els seus mètodes quan l'animació en concret està activa. Aquesta s'utilitza per indicar en quins moments de l'animació d'atac es pot enviar l'ordre de seguir la seqüència, ja que serveix per canviar el valor de la variable *canNextAttack*. Aquesta variable s'encarrega de permetre seguir amb la seqüència d'atacs, o no.

Per assignar la notificació, simplement s'ha d'entrar a un *AnimationMontage*, i dir de quin moment a quin moment de l'animació ha d'estar activa. A la Figura 192 es pot veure que la notificació està activa des del *frame* 25 fins al 45.

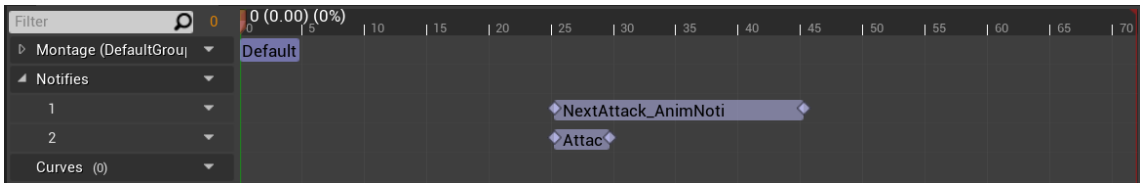

Figura 192. NextAttack_AnimNotify

6.6.6.1 RecievedNotifyBegin

Aquesta funció s'executa a l'iniciar la notificació. Simplement posa el valor de *canNextAttack* a cert. Veure Figura 193.

Figura 193. NextAttack_AnimNotify - RecievedNotifyBegin

6.6.6.2 RecievedNotifyEnd

Aquesta funció s'executa a l'acabar la notificació. Simplement posa el valor de *canNextAttack* a fals. Veure Figura 194.

Figura 194. NextAttack_AnimNotify - RecievedNotifyEnd

6.6.7 AttackParent_AnimNotify

Aquesta és una altra notificació que s'assigna a les animacions d'atac. Té l'objectiu de colpejar els enemics quan es fa un atac. El què fa és dibuixar uns *colliders* a cada *Tick* del rellotge per saber si l'espada a tocat un enemic. Si el toca, envia a l'enemic que l'ha colpejat. Aquesta notificació té dos fills, un per cada espasa.

6.6.7.1 RecievedNotifyBegin

Aquesta funció s'executa a l'iniciar la notificació. El què fa és deixar buida la llista d'enemics colpejats que té el *ThirdPersonCharacter*. Això es fa a cada atac, per tal d'evitar que es pugui enviar a l'enemic que s'ha colpejat més d'una vegada per cada cop. A més, executa el so de l'espasa. Veure Figura 195.

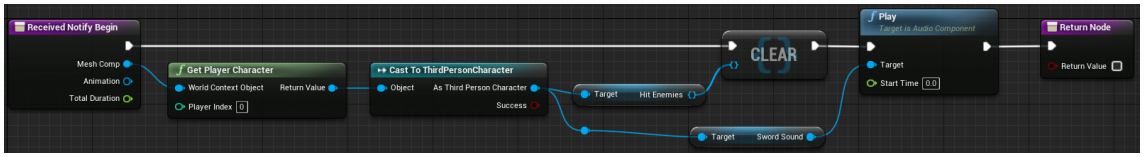

Figura 195. AttackParent_AnimNotify - RecievedNotifyBegin

6.6.7.2 MakeColliderTrace

Aquesta funció dibuixa els *colliders* que indicaran si s'ha colpejat a l'enemic. Aquests es dibuixaran des d'una posició inicial fins una de final, indicades pels ossos del cos del personatge indicats, i amb la mida indicada. Veure Figura 196.

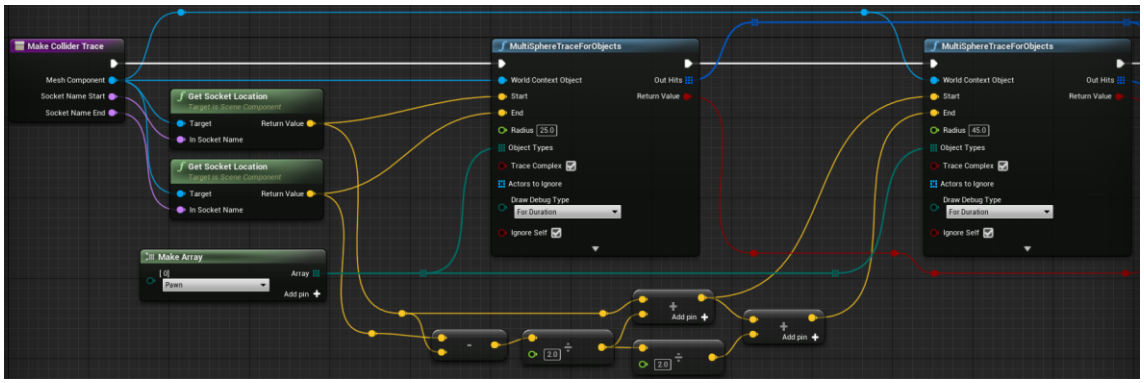

Figura 196. AttackParent_AnimNotify - MakeColliderTrace - Part 1

Després de dibuixar els *colliders*, aquests indicaran quins elements del món han intersecat. Per cadascun d'aquests, es cridarà la funció *MakeHitEnemies*, com es pot veure a la Figura 197.

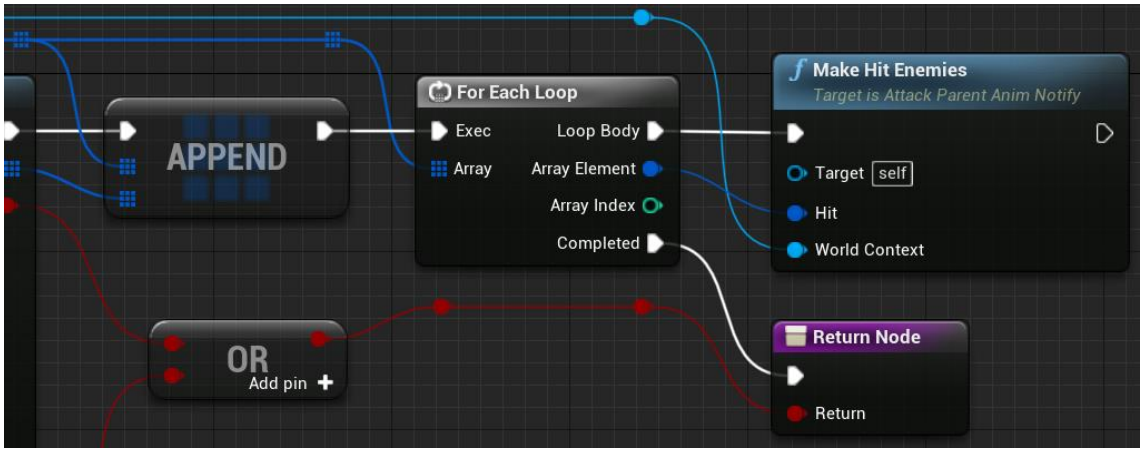

Figura 197. AttackParent_AnimNotify - MakeColliderTrace - Part 2

6.6.7.3 MakeHitEnemies

Aquesta funció rep cadascun dels objectes que s'han colpejat a l'apartat anterior, un per un. El primer que es fa és comprovar que l'objecte rebut sigui un enemic. Si ho és, es comprova que no estigui ja a la llista de colpejats. Si encara no existeix, s'afegeix i s'indica a l'enemic que ha estat colpejat, tal i com es veu a la Figura 198.

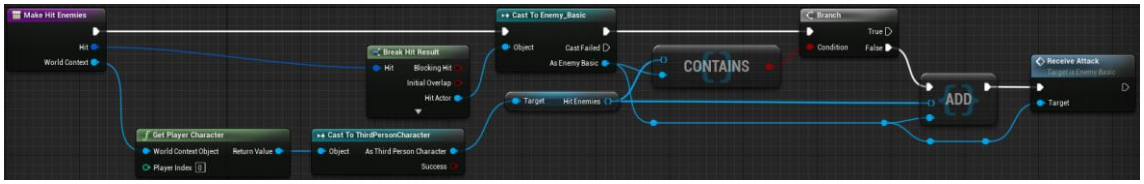

Figura 198. AttackParent_AnimNotify - MakeHitEnemies

6.6.8 AttackLeftWeapon_AnimNotify i AttackRightWeapon_AnimNotify

Aquestes dues són notificacions que hereten de *AttackParent_AnimNotify*. Serveixen per indicar quins són els ossos del personatge on s'hauran de dibuixar els *colliders*, i cridar la funció que ho farà (Apartat 6.6.7.2). Aquestes s'assignen als *AnimationMotange*, tal i com s'ha explicat a l'Apartat 6.6.6.

Només tenen una funció, que es crida cada vegada que el rellotge fa *Tick*. Aquesta crida la funció *MakeColliderTrace* indicant els ossos configurats a l'objecte en concret.

6.7 Animacions del Personatge

Aquest apartat serveix per explicar quin ha estat el procés per crear les diverses animacions del personatge.

Les animacions utilitzades no s'han creat des de zero. Aquestes s'han extret d'internet, i s'han hagut de convertir al personatge que s'utilitza. Per fer la conversió, es necessita d'una funcionalitat d'UE4 anomenada *BoneMapping*.

6.7.1 BoneMapping

Un *BoneMapping* és una conversió dels ossos d'un esquelet cap a un altre. Aquest element és molt necessari per convertir les animacions descarregades, ja que venen fetes sobre un esquelet diferent. El funcionament és senzill: s'ha d'anar mirant poc a poc cadascun dels ossos de l'esquelet, i indicar a quin os fa referencia en un altre esquelet. Per fer-ho, existeixen unes plantilles a UE4. Una d'elles és la d'*Humanoid*, que ha estat la utilitzada.

El què s'ha fet ha estat fer un *BoneMapping* del personatge que utilitzem cap a la plantilla de *Humanoid*, i un altre *BoneMapping* de l'esquelet de les animacions, també cap a *Humanoid*. D'aquesta manera, amb tots dos fets, es poden fer les conversions de les animacions de tots dos esquelets cap a l'altre.

6.7.2 AnimationMontage

Aquest és un *Blueprint* per defecte d'UE4. Aquest permet, a partir d'una animació base, afegir-hi funcionalitats. Aquest ha estat un altre element important en la gestió d'animacions, ja que s'ha treballat amb aquests durant tot el projecte.

La funció utilitzada ha estat, principalment, l'assignació de les notificacions explicades. Tal com s'ha explicat a l'Apartat 6.6.6, als *AnimationMontage* se'ls hi poden assignar notificacions, són aquestes les que afegeixen funcionalitat.

Totes les animacions de combat de la protagonista porten també assignades notificacions extres per als dos sistemes de partícules implementats, els del rastre que deixa l'espasa (*Trail*) i els que surten com petites espurnes.

6.8 Sistema de partícules

Al joc hi han dos sistemes de partícules a destacar: el *Trail* o rastre que deixa l'espasa en el seu moviment, i unes espurnes que surten en diferents direccions, ambdues extretes d'internet.

Aquests sistemes van associats cadascun a un sistema de partícules diferent (Figura 199) i també van associats a uns *Sockets* associats a uns punts de l'esquelet, per poder saber des de quin punt han de sortir les partícules com es pot veure a la Figura 200 i Figura 201.

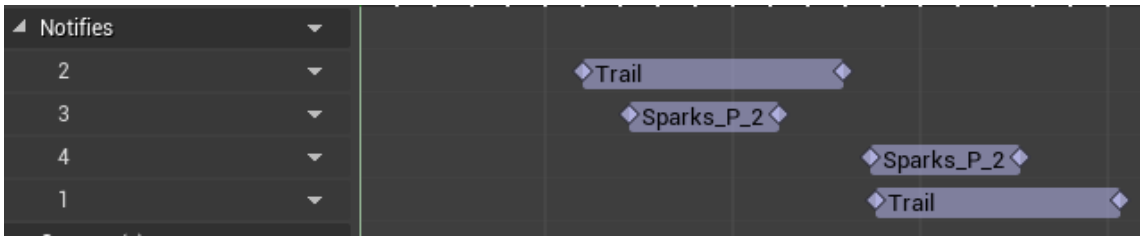

Figura 199. Notificacions dels rastres de cada espasa i de les xispes

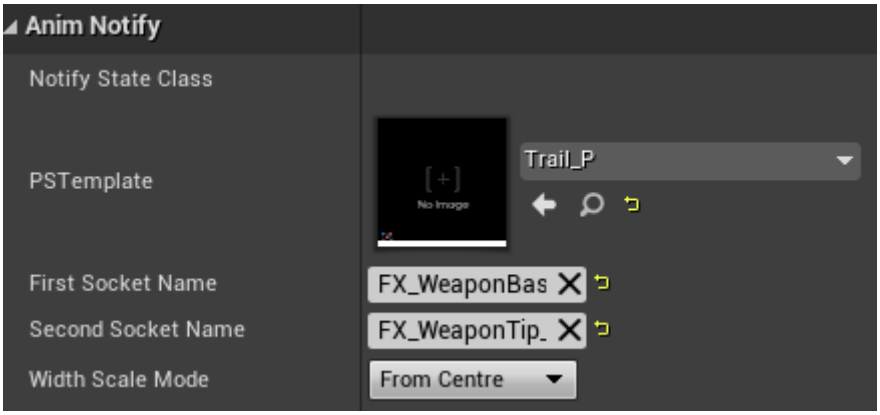

Figura 200. Detalls sobre la notificació Trail, amb el sistema de partícules associat, que s'encarrega des de quin punt inicial a quin punt final deixa rastre l'espasa.

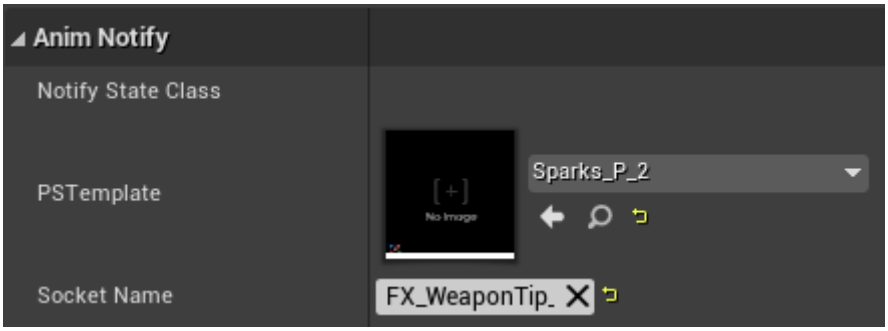

Figura 201. Detalls sobre la notificació Sparks, amb el sistema de partícules associat, que s'encarrega des de quin punt de l'espasa es llencen espurnes.

6.9 Enemics

Els enemics són tots aquells NPC's contra qui ha de lluitar el jugador. Per crear-los, s'ha fet primer una base inicial d'on podessin heretar tots els altres, de forma que es puguin crear altres funcionalitats respecte els enemics i s'apliquessin a tots ells per igual, sense cap tipus de problema. La base creada amb aquest propòsit és l'*EnemyBasic*.

En aquest cas, existeixen dos tipus d'enemics diferents: els *Minions* i el *Boss* final. Els primers són enemics més bàsics que permetran al jugador aprendre els moviments del combat i guanyar experiència. D'altra banda està el *Boss* final, que és l'enemic que s'ha de vèncer per tal de finalitzar la partida. Si bé es pot lluitar amb ell i guanyar-lo des del principi, s'han escollit les estadístiques per tal que això sigui molt complicat i, en canvi, si el jugador opta per obtenir runes i pujar de nivell, serà més fàcil matar-lo. Finalment, cal destacar que aquest prototip es centra més en les mecàniques. Per aquest motiu, les estadístiques dels enemics són més baixes per tal que sigui més senzill poder testejar l'experiència al complert.

Quant a la implementació, cada enemic té una classe pròpia, on implementa alguns aspectes i afegeix nous comportaments, segons cada necessitat. Aquestes classes estan explicades als següents apartats.

6.9.1 Enemic bàsic

L'enemic bàsic és, com ja s'ha dit, el pare de qui heretaran tots els enemics, per tal per d'evitar repetir codi. Aquest està format per un Blueprint tipus *Pawn* que implementa l'*Event ReceiveAttack* comentat a l'Apartat 6.6.7.3, i una interfície i implementació per a la part de marcatge d'enemics. Aquesta segona part s'explica a l'Apartat 6.10. A més, té dues funcions que no tenen cap implementació: *PlayReceiveDamage* i *Die*. Ambdues s'utilitzen a l'*Event ReceiveAttack*, permetent que cada enemic les hereti, implementi i gestioni segon les necessitats.

6.9.1.1 Rebre atac

Aquesta funció és cridada per part del personatge, quan detecta la col·lisió d'un atac amb un enemic. Inicialment, comprova si està assignada la referència al personatge del jugador i, si no és així, l'assigna. A continuació obté els punts d'atac del jugador i els resta a la vida. Després crida a la funció *PlayReceiveDamage*, que s'implementa a la classe que hereta. Veure Figura 202.

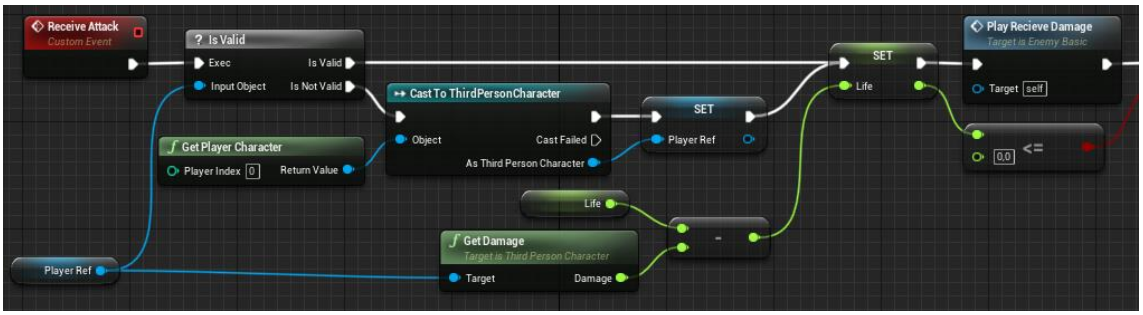

Figura 202. ReceiveAttack - Part 1

Finalment, es comprova si la vida és inferior a 0. Si és així desactiva el moviment i executa la funció Die, que també s'implementa a la classe que hereta. Veure Figura 203.

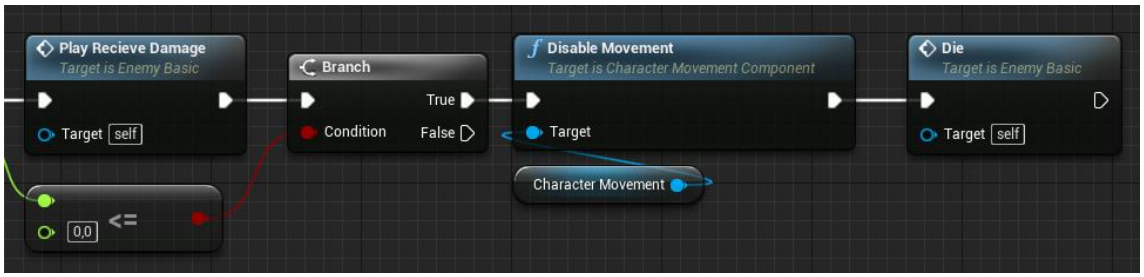

Figura 203. ReceiveAttack - Part 2

6.9.2 Minions

Aquest és l'enemic més dèbil. Està ubicat per tot el mapa inicial i permet al jugador aprendre bé les mecàniques de combat mentre puja de nivell. Es pot veure caminant de forma aleatòria pel mapa, sempre dins d'un radi. La lògica de combat es basa en caminar rodejant a l'objectiu (en aquest cas el jugador) i, en algun moment aleatori, atacar. Si perd de vista al jugador, torna a la zona inicial per seguir patrullant. Així doncs, tal com es pot apreciar a la Figura 204, té tres estats:

- *NotDetected*: caminarà de forma aleatòria dins un radi a partir de la seva posició inicial.
- *Holding*: si està massa lluny del jugador corre cap a ell. Un cop arriba, es dedica a orbitar al voltant del jugador.
- *Attacking*: s'apropa encara més al jugador i llança un atac. Quan ha finalitzat, torna a l'estat de Holding.

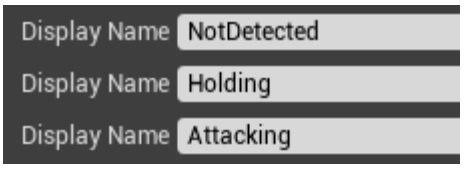

Figura 204. Valors d'AIState

Tot aquest comportament ve definit gràcies a tres parts: *Minion*, *Minion_Controller* i *Minion_BT*, explicats als següents apartats.

6.9.2.1 Minion

Aquest Blueprint és el que conté el model de l'enemic i qui hereta la classe *Enemy_Basic*, explicat a l'Apartat 6.9.1. La funció és inicialitzar i implementar alguns atributs de la classe pare i afegir alguns *Event* extra.

6.9.2.1.1 BeginPlay

Aquest *Event* es crida al principi, s'utilitza per donar valor a les variables *Life* i *Attack*, que provenen d'*Enemy_Basic* i assigna la posició inicial al *Behavior Tree* (BT) esmentat a l'Apartat 6.9.2.4. Veure Figura 205.

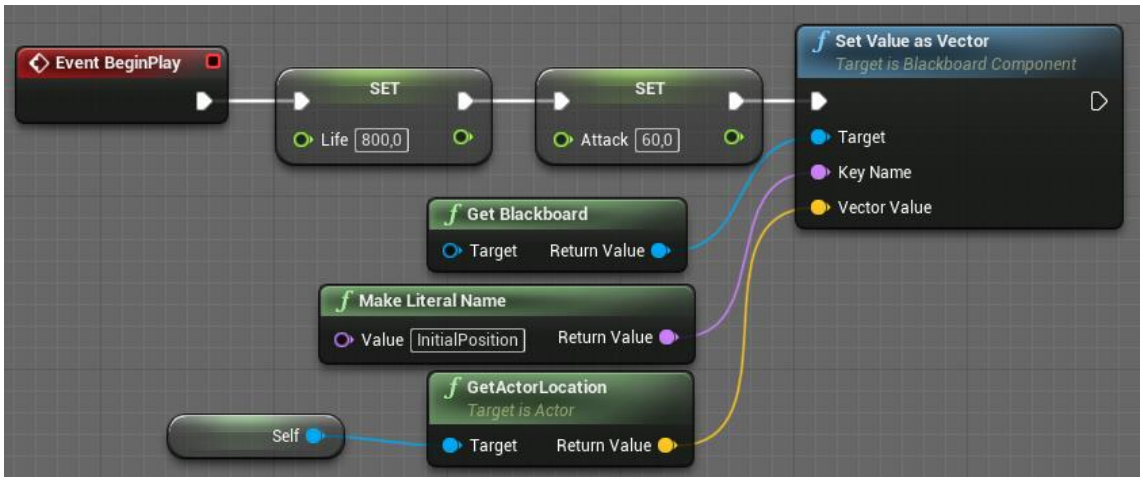

Figura 205. BeginPlay al Minion

6.9.2.1.2 Col·lisió amb un altre enemic

Aquest *Event* es va implementar a causa d'una problemàtica: en algunes ocasions, quan hi ha més d'un *Minion* volent atacar, un es podia quedar darrere l'altre, bloquejat. Per aquest motiu, si un enemic xoca amb un altre, es crida a la funció *AlertCollidingMate*, esmentada a l'Apartat 6.9.2.2.5, per a que es puguin recol·locar. Veure Figura 206.

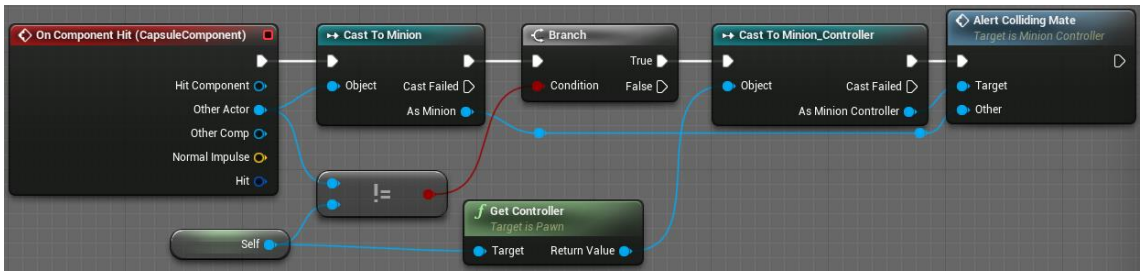

Figura 206. Event OnComponentHit

6.9.2.1.3 PlayerHit

Quan un enemic ataca, es dibuixen uns *colliders* que segueixen el camí de l'espasa. Si aquests *colliders* col·lisionen amb el jugador, es crida a la funció *ReceiveDamage*, explicada a l'Apartat 6.4.1.3. S'utilitza el node *DoOnce* per evitar aplicar el mal més d'un cop amb un sol atac. Aquest node es reinicia quan s'acaba l'animació d'atac, explicada a la següent secció. Veure Figura 207.

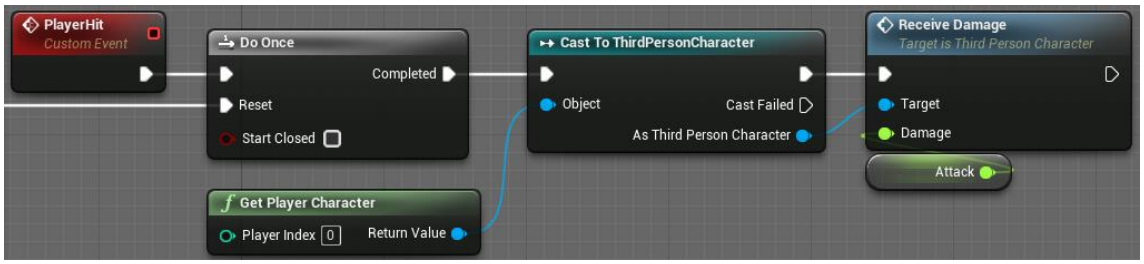

Figura 207. PlayerHit

6.9.2.1.4 PlayAttack

Aquest *Event* es crida per realitzar l'animació d'atac. Agafa una animació aleatòria de l'*AttackArr* (veure valors a la Figura 208) i s'executa. Un cop s'acaba, es reinicia el *DoOnce* explicat a la secció anterior. Veure Figura 209.

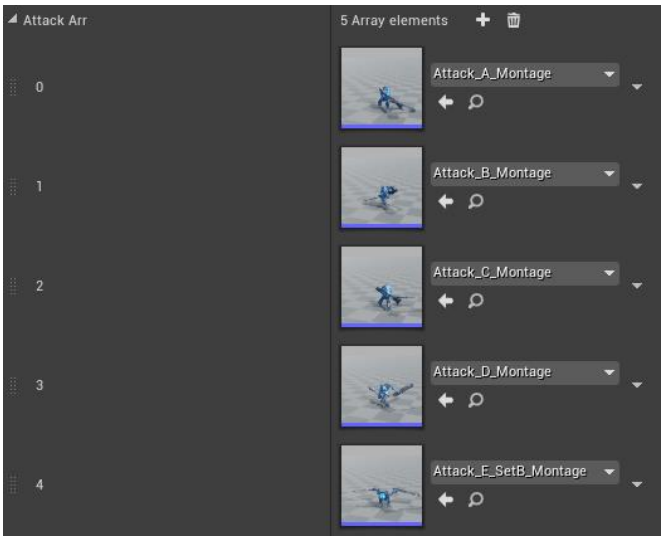

Figura 208. Valors d'AttackArr

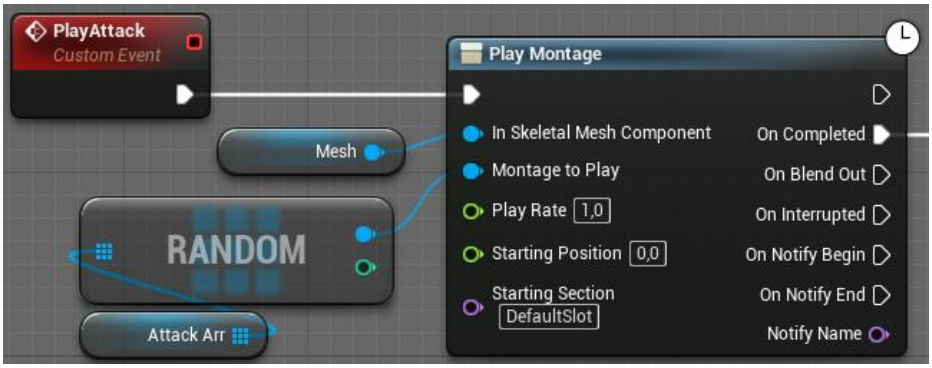

Figura 209. PlayAttack

6.9.2.1.5 PlayRecieveDamage

Aquest *Event* és una implementació de la classe pare. Executa l'animació de rebre mal cada cop que el jugador ataca a l'enemic. Veure Figura 210.

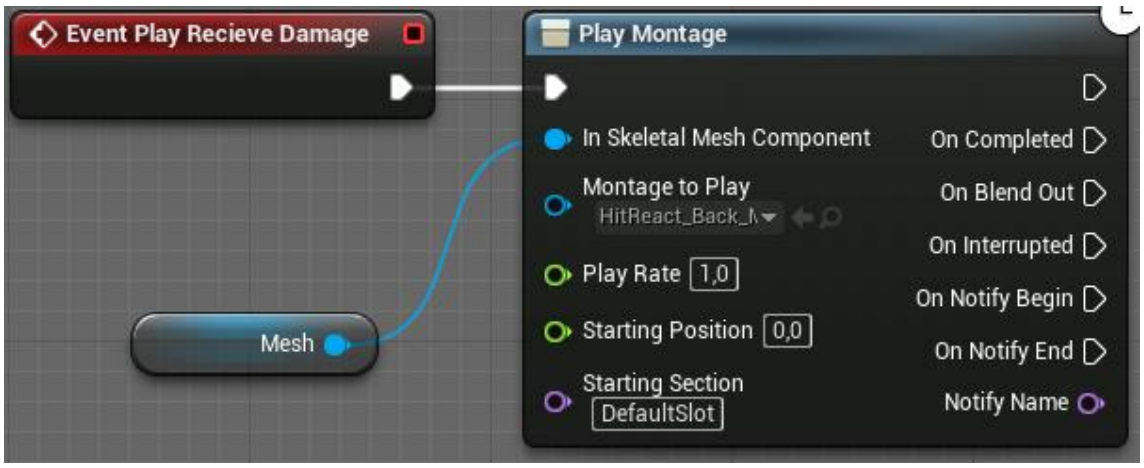

Figura 210. PlayRecieveDamage

6.9.2.1.6 Die

De mateixa manera que la secció anterior, aquest Event implementa la funció Die de la classe pare. Per començar l'execució, es fa una crida a *ChangeInCombat* del *ForestBackgroundMusicManager* amb *addMinion* a fals (Apartat 6.16.2.1.5), es desmarca si és l'objectiu, i s'elimina de la llista d'enemics del *ThirdPersonCharater*. Veure Figura 211.

Figura 211. Die - Part 1

A continuació, afegeix l'experiència pertinent al jugador amb *AddExperience*, esmentada a l'Apartat 6.4.2.1. Executa una animació aleatòria de *DeathArr* (valors a la Figura 212) i quan acaba destrueix l'actor. Veure Figura 213.

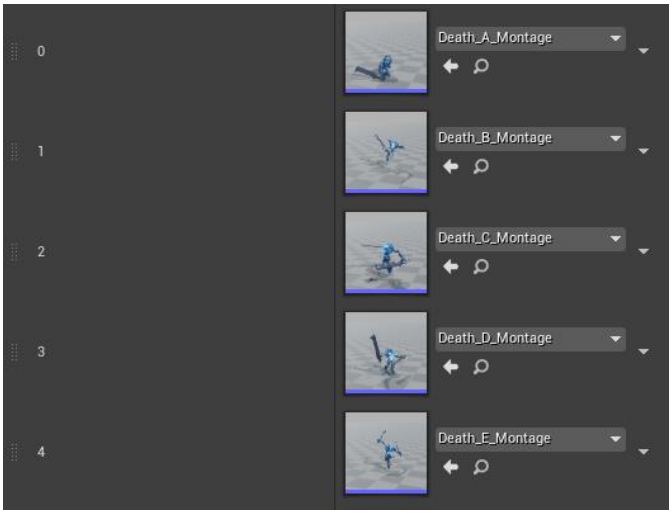

Figura 212. Valors de DeathArr

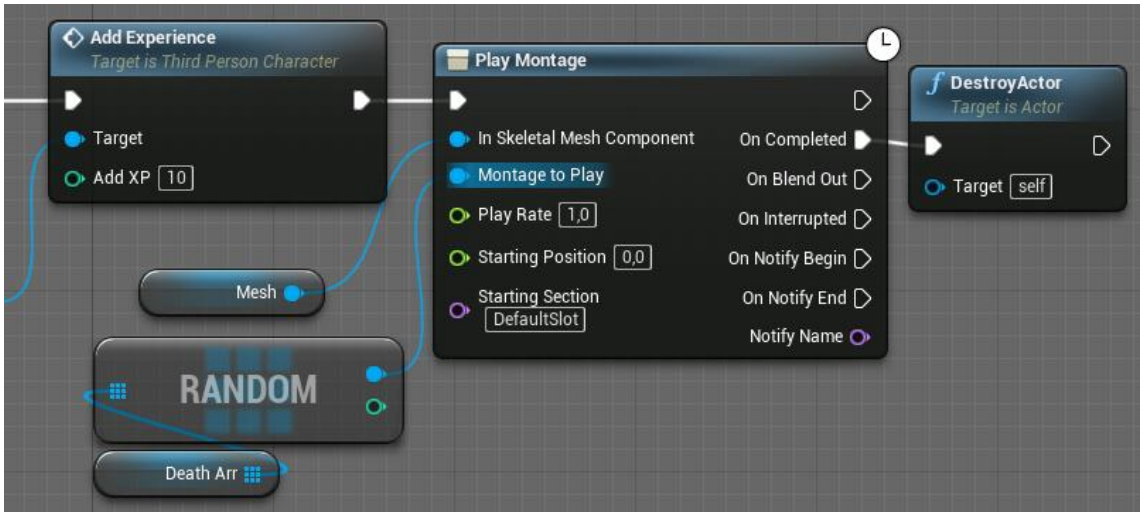

Figura 213. Die - Part 2

6.9.2.2 Minion Controller

Aquest Blueprint funciona com a cervell del *Minion*. Basant-se en el BT, és qui decideix quines accions es realitzen i els canvis d'estat de cada individu.

6.9.2.2.1 BeginPlay

Quan s'inicia posa l'estat per defecte a *NotDetected*, guarda una referència al *Minion* controlat i executa el BT del *Minion*. Veure Figura 214.

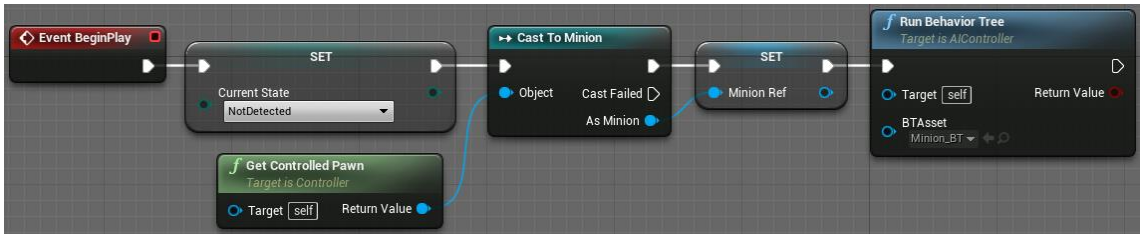

Figura 214. BeginPlay al Minion_Controller

6.9.2.2.2 OnTargetPerceptionUpdated

Aquest *Event* ve donat per un component anomenat *AIPerception*, el qual se li han donat els valors de la Figura 215 per tal que pugui detectar al personatge a través de la visió.

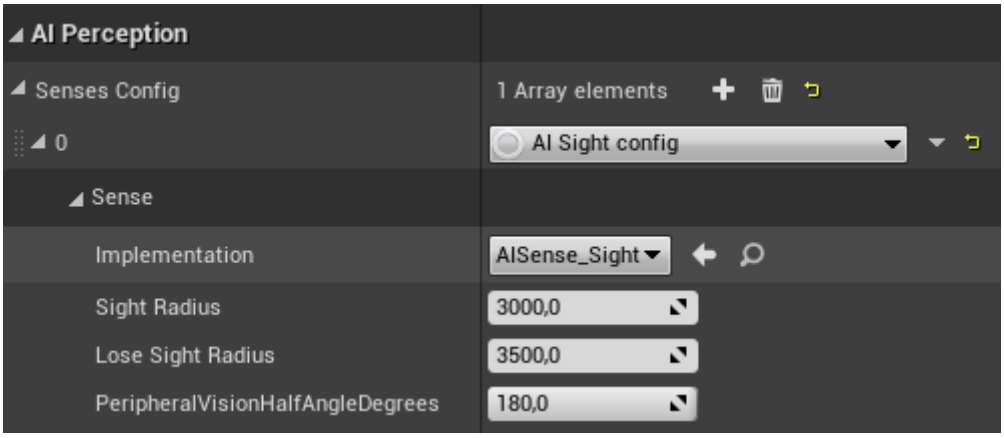

Figura 215. Valors AIPerception

Quan comença a detectar un nou actor o deixa de detectar-ne un altre, es dispara aquest *Event*. Inicialment es comprova si és un enemic i, si no és així, es gestiona l'estímul visual a través de la funció *HandleSightSense*, explicada a l'Apartat 6.9.2.2.7. Aquesta funció retorna si ha detectat una amenaça. Si no és així, es retorna a l'estat de *NotDetected*. En canvi, si ha detectat al jugador, s'activa l'*Event* d'*AIAttack*, esmentat a la següent secció. Veure Figura 216 i Figura 217.

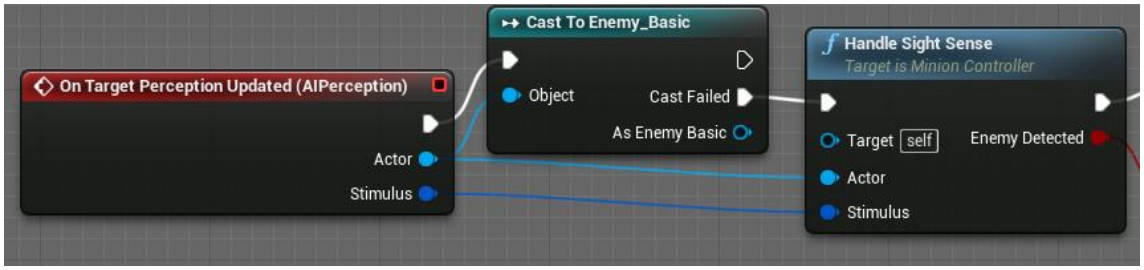

Figura 216. OnTargetPerceptionUpdated - Part 1

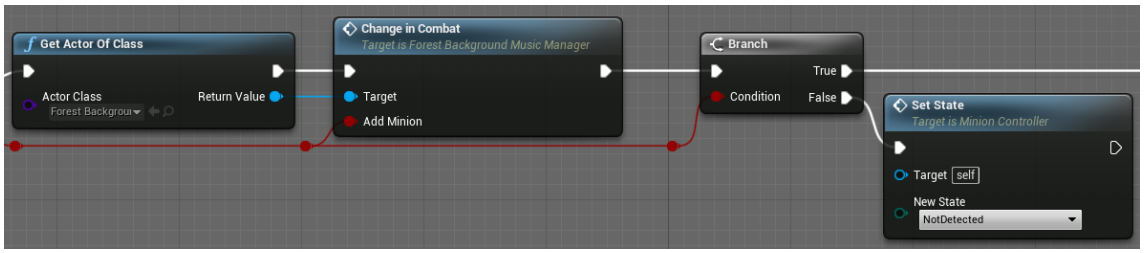

Figura 217. OnTargetPerceptionUpdated - Part 2

6.9.2.2.3 Event AIAttack

Mentre l'enemic es troba en estat de Holding, s'utilitza aquest *Event* que, amb el node *SetTimerByEvent*, es fa que es dispari de forma aleatòria en un rang d'entre un i tres segons. Cada cop que es dispara es comprova si l'estat és *NotDetected* per saber si no ha de fer res o ha d'atacar, respectivament. Veure Figura 218.

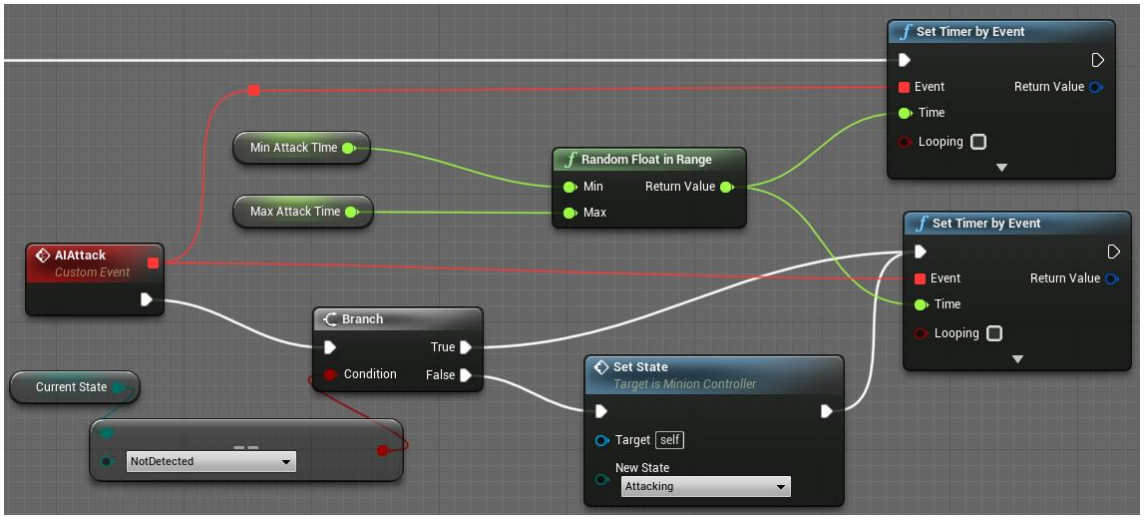

Figura 218. AIAttack

6.9.2.2.4 MinionAttack

Aquest *Event* s'ha creat per a ser executat des de la *Task Attack_Target*, esmentada a l'Apartat 6.9.2.3.1. Agafa la referència del *Minion* i executa l'*Event PlayAttack*, explicat a l'Apartat 6.9.2.1.4. Veure Figura 219.

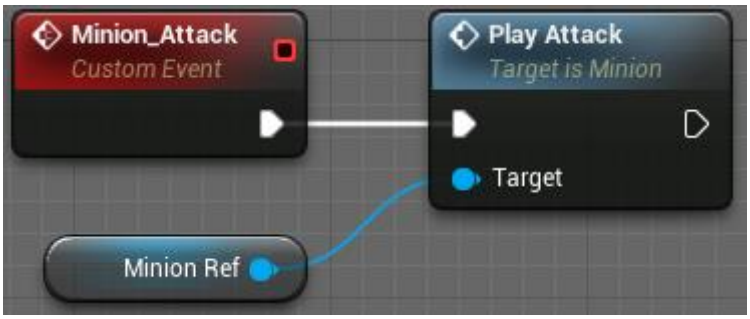

Figura 219. Minion_Attack

6.9.2.2.5 AlertCollidingMate

Un cop es rep l'alerta que dos *Minions* han col·lisionat, s'utilitza aquesta funció que canvia a l'estat que correspongui tant a l'actor que l'executa com a l'actor amb qui ha xocat. Per fer-ho, s'utilitza dos cops la funció *SetState*, esmentada a l'Apartat 6.9.2.2.6, tal com es pot apreciar a la Figura 220.

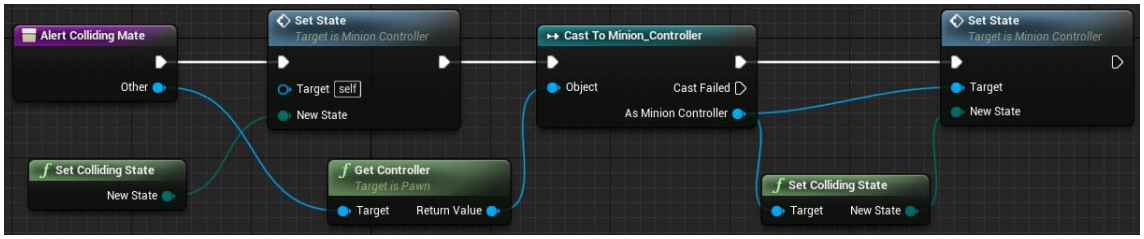

Figura 220. AllertCollidingMate

Per triar quin estat s'ha d'utilitzar s'aplica la funció *SetCollidingState*, que si estava a *NotDetected* el manté i si no el canvia a Holding. Veure Figura 221.

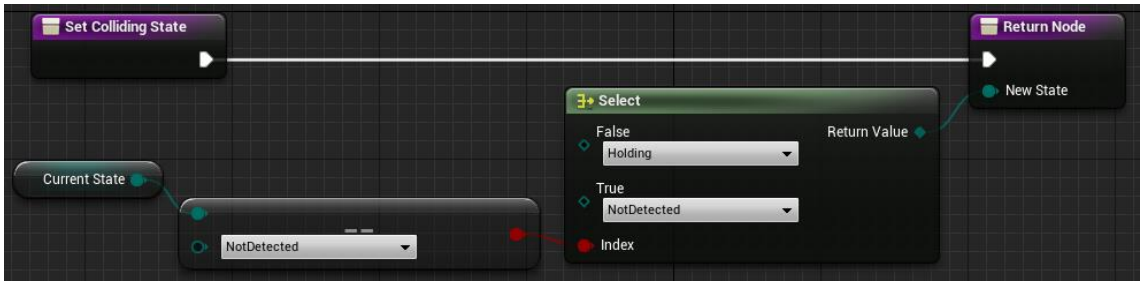

Figura 221. SetCollidingState

6.9.2.2.6 SetState

Aquesta funció rep com a paràmetre el nou estat i el guarda tant a la variable *CurrentState* com a l'*AIState*, del BT (explicat a l'Apartat 6.9.2.4). A més, modifica l'objectiu, deixant-lo buit si és *NotDetected* o el *ThirdPersonCharacter*, si és qualsevol altre estat. Veure Figura 222.

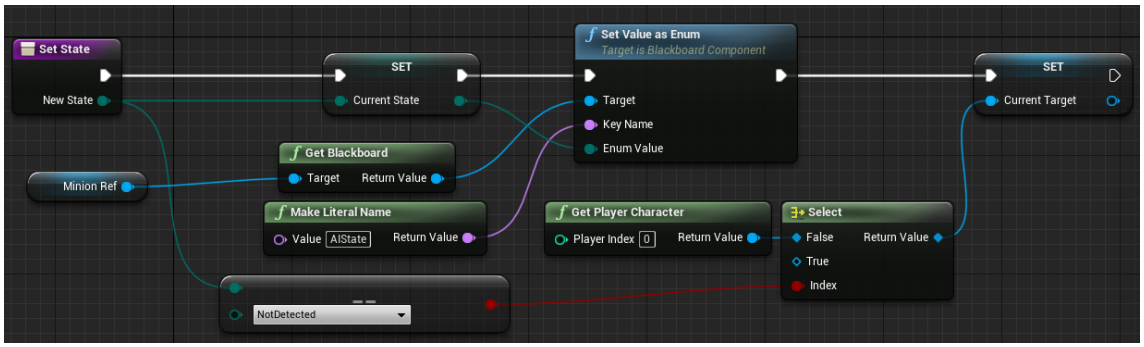

Figura 222. SetState

6.9.2.2.7 HandleSightSense

Aquesta funció rep com a paràmetre l'actor que s'ha detectat i l'estímul rebut. Si l'estímul és visual (*AISense Sight*) i l'actor és el jugador, es canvia l'estat. Altrament, es retorna que no s'ha detectat res. Veure Figura 223.

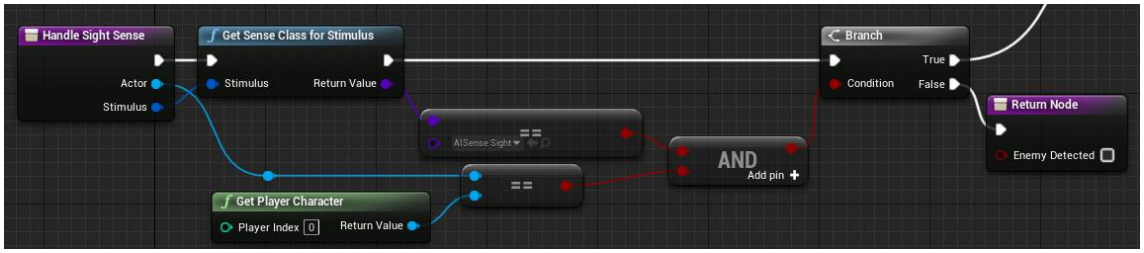

Figura 223. HandleSightSense – Part 1

Si l'estímul ve per part del jugador, es comprova si l'ha començat a veure o l'ha deixat de detectar amb *StimulusSuccesfullySensed*. Si és cert es posa l'estat de Holding, *NotDetected* altrament, utilitzant *SetState*, esmentat a la secció anterior. Veure Figura 224.

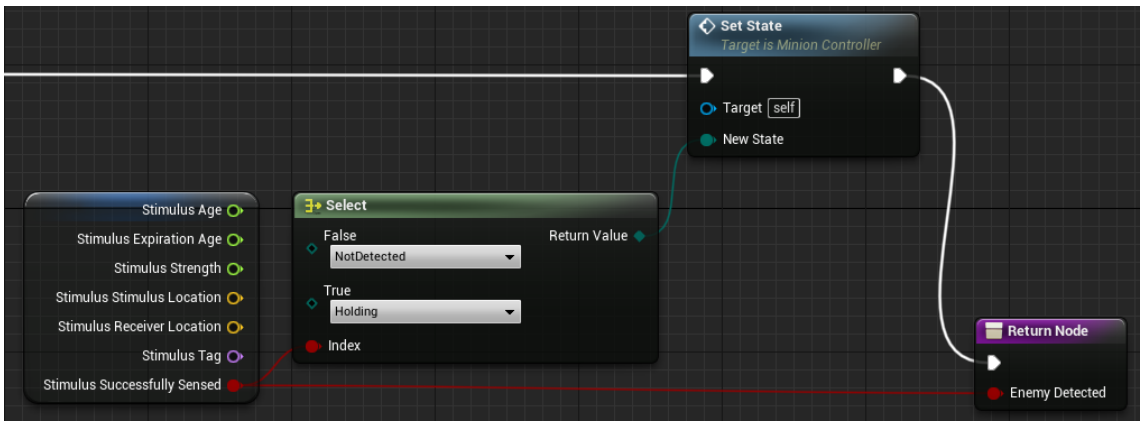

Figura 224. HandleSightSense - Part 2

6.9.2.3 Tasks

Les tasques, o *Tasks*, són funcions que s'executen des del BT (Apartat 6.9.2.4). Per iniciar-les cal utilitzar el node *EventReceiveExecuteAI* i sempre al final és important cridar el node *FinishExecute*, indicant si s'ha realitzar la tasca amb èxit o no. Les variables que inicien el nom per "BB", venen donades pel BT i s'expliquen en aquell apartat.

6.9.2.3.1 Attack_Target

Amb aquesta tasca es comprova si el *Minion* està suficientment a prop. Si és així, crida a l'*Event Minion Attack*, esmentat a l'Apartat 6.9.2.2.4. Veure Figura 225.

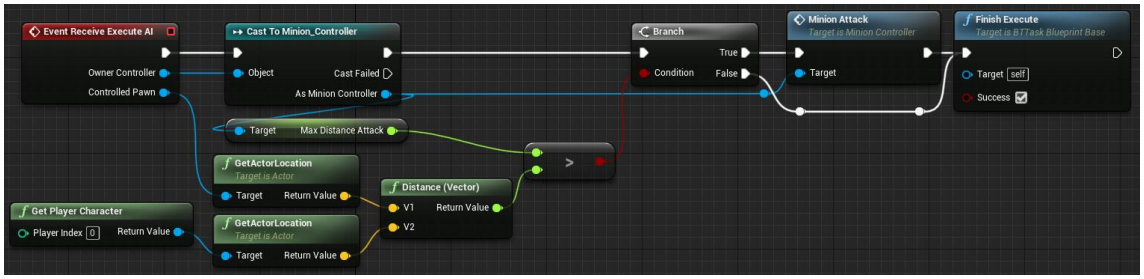

Figura 225. Attack_Target

6.9.2.3.2 ChangeSpeed_Task

Donat un *WalkSpeed* (el qual ve donat des del BT), canvia la velocitat de moviment de l'enemic controlat. Veure Figura 226.

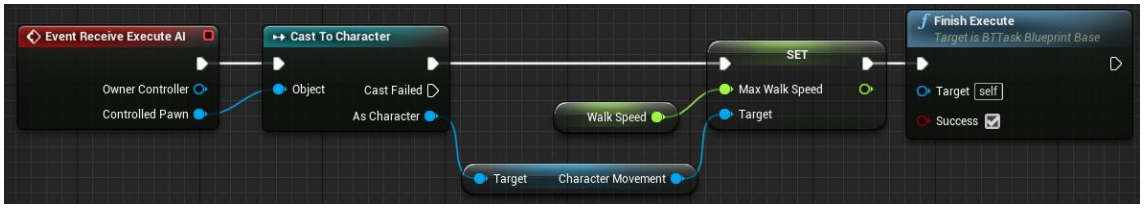

Figura 226. ChangeSpeed_Task

6.9.2.3.3 FindPlayerLocation

Si l'enemic es troba en estat de *NotDetected*, deixa com a buit el focus i posa com a objectiu la posició inicial. Si no és així, canvia la posició objectiu per la posició del jugador. Veure Figura 227.

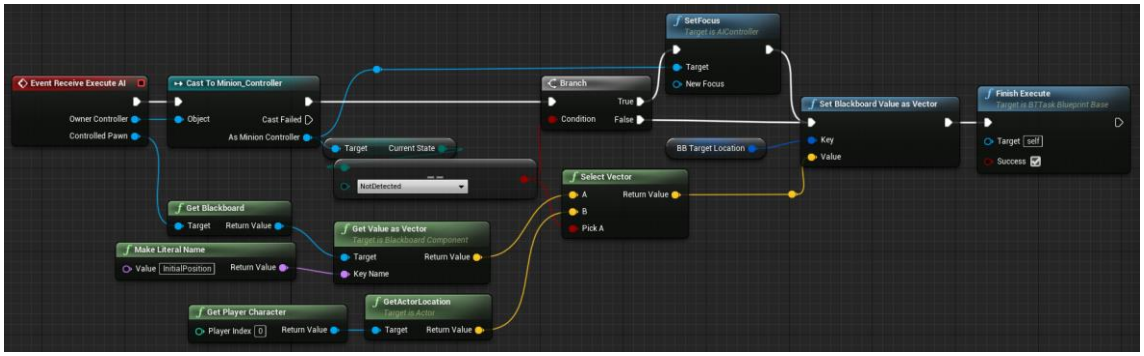

Figura 227. FindPlayerLocation

6.9.2.3.4 FindRandomPosition_Task

Donada una posició inicial, busca un punt aleatori dins la *NavMesh*, explicada a l'Apartat 4.1, amb un radi de 3000 i marca com a objectiu aquest punt. Veure Figura 228.

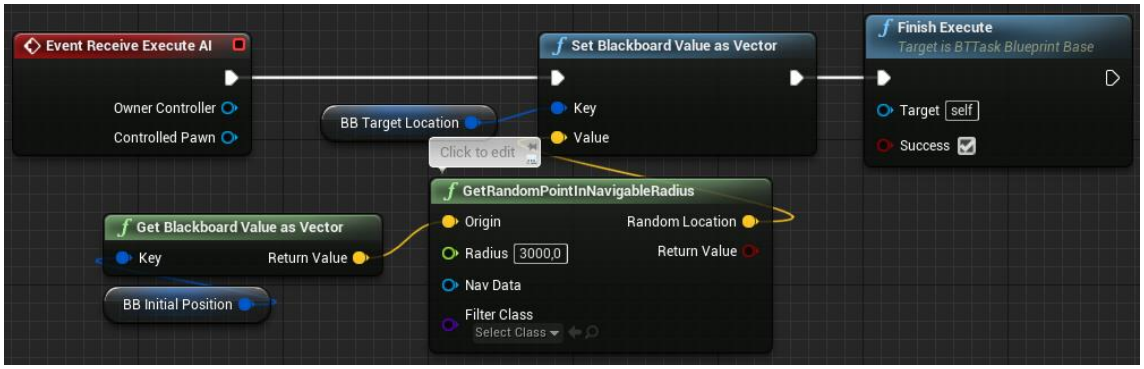

Figura 228. FindRandomPosition_Task

6.9.2.3.5 FindWanderLocation_Task

Quan s'està atacant al jugador. L'enemic camina al voltant fins que es dispara l'*Event* d'atacar. Per fer-ho, obté un punt aleatori al seu voltant amb *GetRandomPoint* (veure Figura 231) i el recalcula constantment fins que sigui superior a la distància mínima (*MinWanderingDistance*). Un cop l'ha trobat, el guarda com a nou objectiu per desplaçar-se. Veure Figura 229 i Figura 230.

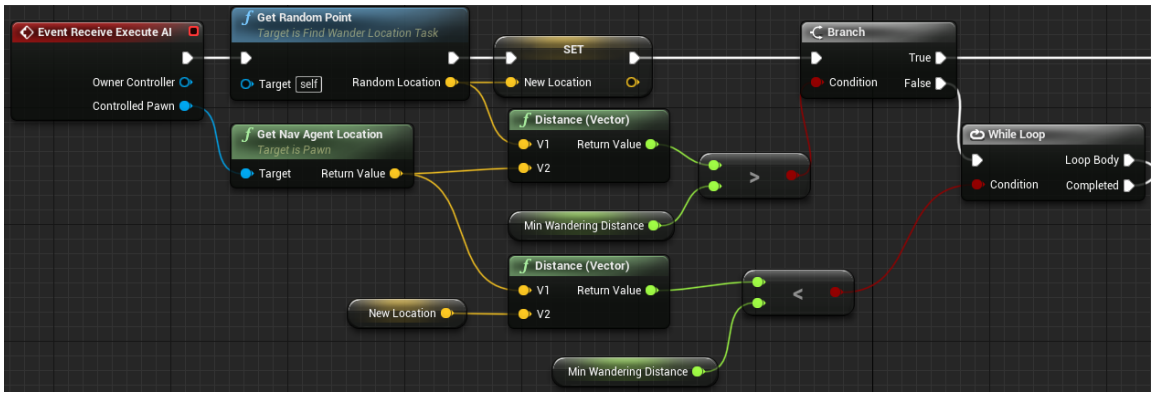

Figura 229. FindWanderLocation_Task – Part 1

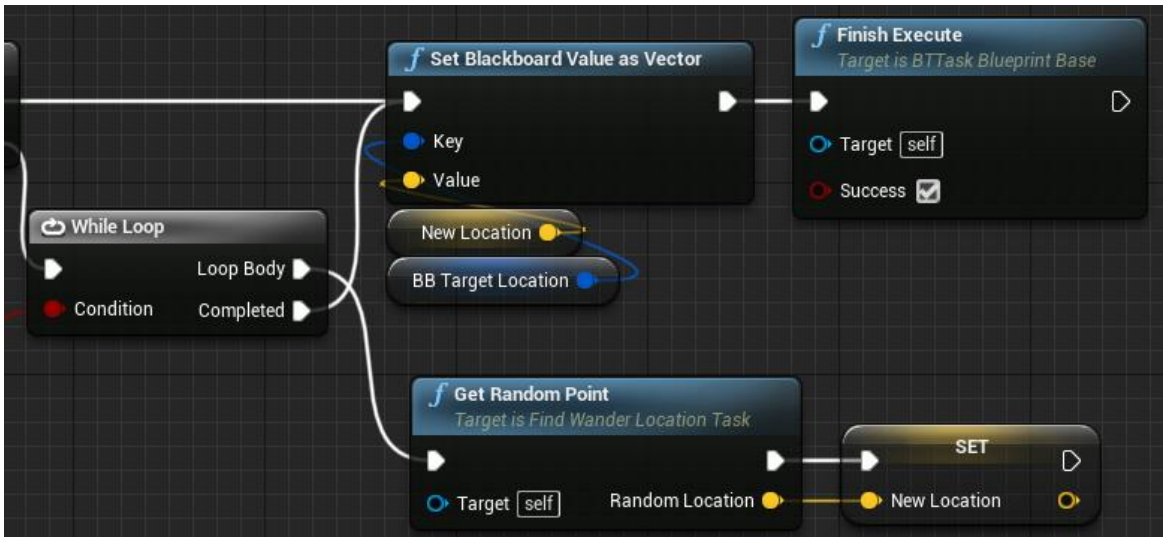

Figura 230. FindWanderLocation_Task - Part 2

La funció *GetRandomPoint* utilitza un radi i una localització donades i obté un punt aleatori dins la *NavMesh*.

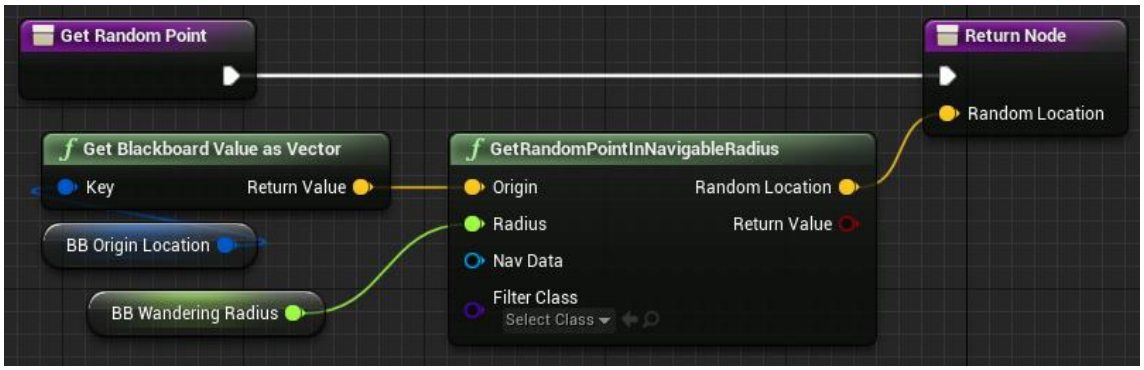

Figura 231. GetRandomPoint

6.9.2.3.6 FocusTarget

Aquesta tasca, a part del *Minion*, s'ha reaprofitat per utilitzar a l'enemic final, explicat a l'Apartat 6.9.3. Si és un *Minion*, posa el focus sobre el *CurrentTarget*, que sempre serà el *PlayerCharacter*, però s'ha fet així per si es vol canviar el disseny i tenir *Minions* aliats. En canvi, per l'enemic final sempre es posa el jugador com a objectiu. Veure Figura 232.

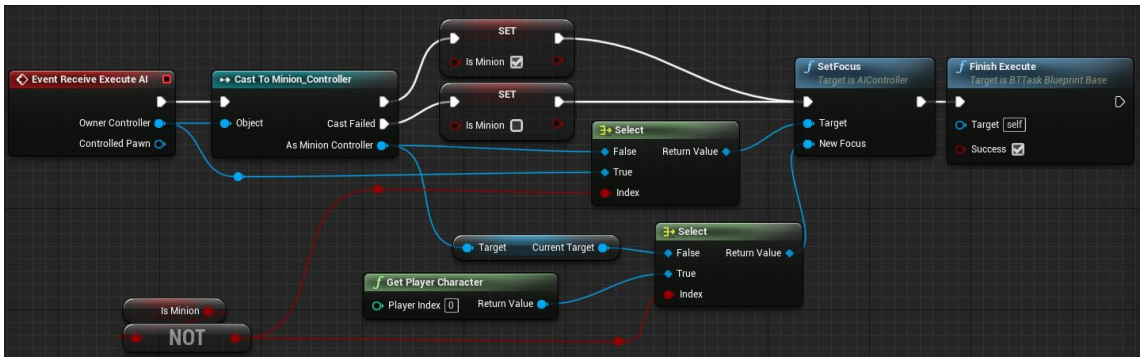

Figura 232. FocusTarget

6.9.2.3.7 GetDistanceToPlayer

Aquesta tasca obté la posició del *Minion*, calcula la distància amb el jugador i la guarda al BT. Veure Figura 233.

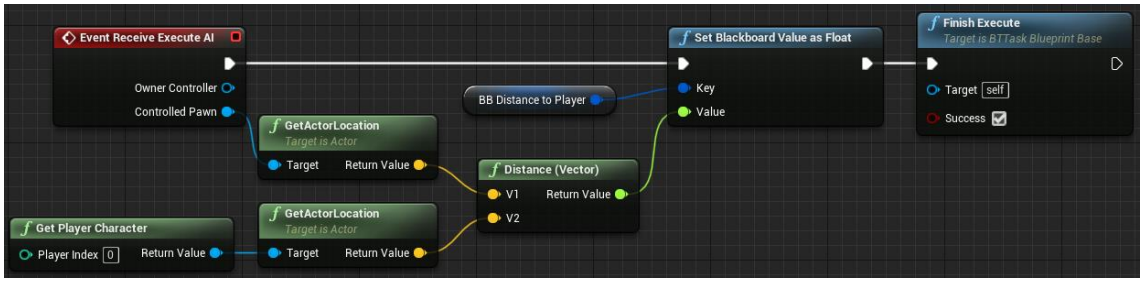

Figura 233. GetDistanceToPlayer

6.9.2.3.8 Reset_State

Un cop s'ha finalitzat l'atac correctament, es retorna a l'estat de *Holding* i es guarda al *Controller* i al BT. Veure Figura 234.

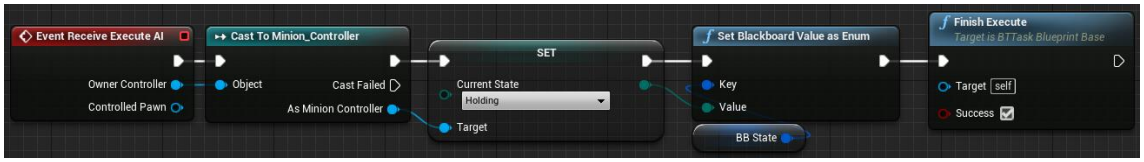

Figura 234. Reset_State

6.9.2.3.9 SetAnchorLocation

Quan s'arriba a prop del jugador, es guarda la posició del *Minion* per després realitzar els moviments al voltant d'aquest. No es guarda la posició del jugador com a tal perquè es podria obtenir una posició a l'altre cantó i això provocaria que l'enemic no arribés a l'objectiu perquè estaria el jugador al mig. Veure Figura 235.

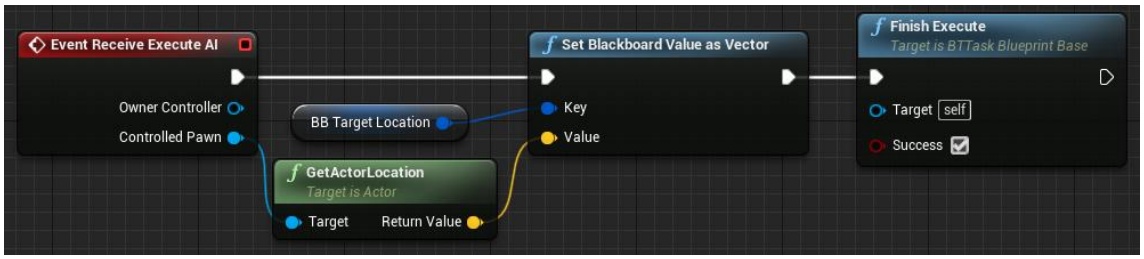

Figura 235. SetAnchorLocation

6.9.2.4 Minion_BT

El BT es composa de dues parts: l'arbre de comportament (veure Figura 236) i el *Blackboard* (veure Figura 237). El primer és on es defineixen els estats de l'enemic i què ha de fer en cada cas a través de les tasques. Algunes, com el *Wait* o *MoveTo*, però l'usuari pot definir les seves pròpies, les quals estan explicades a la secció anterior. El segon s'utilitza per definir les variables del BT, anomenades *Keys*.

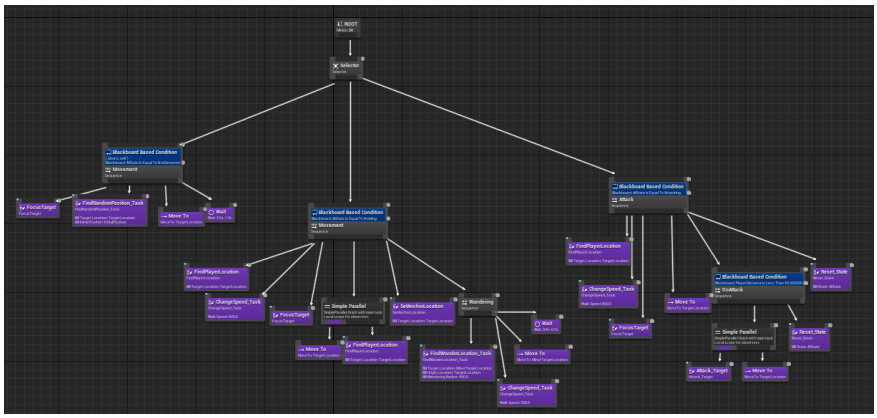

Figura 236. Behavior Tree - Visió General

Figura 237. Keys del Blackboard

El primer estat és el *NotDetected*. Primer s'executa el *FocusTarget*, esmentat a l'Apartat 6.9.2.3.6, per assegurar que el *Minion* no mira al jugador. A continuació es busca un objectiu amb *FindRandomPosition_Task*, explicat a l'Apartat 6.9.2.3.4, i s'executa el *MoveTo*. Finalment, espera entre un i tres segons i retorna a l'execució. Veure Figura 238.

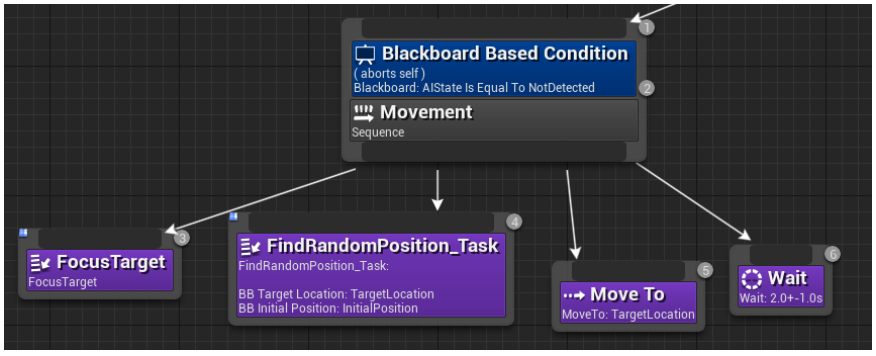

Figura 238. Behavior Tree – NotDetected

L'estat més complex és el de Holding. En aquest es guarda a *TargetLocation* la posició del jugador amb *FindPlayerLocation*, esmentat a l'Apartat 6.9.2.3.3. A continuació es canvia la velocitat a 600 amb *ChangeSpeed_Task*, veure Apartat 6.9.2.3.2, i s'obliga a mirar al jugador amb *FocusTarget*. Seguidament s'utilitza un *Parallel*, per tal que, mentre s'executa la tasca *MoveTo*, es vagi executant *FindPlayerLocation* per seguir al jugador. Un

cop s'acaba aquesta part es guarda la posició actual amb *SetAnchoLocation*, esmentat a l'Apartat 6.9.2.3.9 i comença la *Sequence* de *Wandering*.

Aquesta part consisteix en anar orbitant a prop del jugador fins que l'*AIState* canviï a *Attacking*. Per fer-ho, es busca una posició aleatòria que es guarda a *MinorTargetLocation* amb *FindWanderLocation*_Task, veure Apartat 6.9.2.3.5, es disminueix la velocitat a 300 amb *ChangeSpeed_Task* i s'executa *MoveTo* per anar al nou objectiu. En aquest cas només espera entre 0.3 i 1.3 segons per tornar a realitzar una acció. Veure Figura 239.

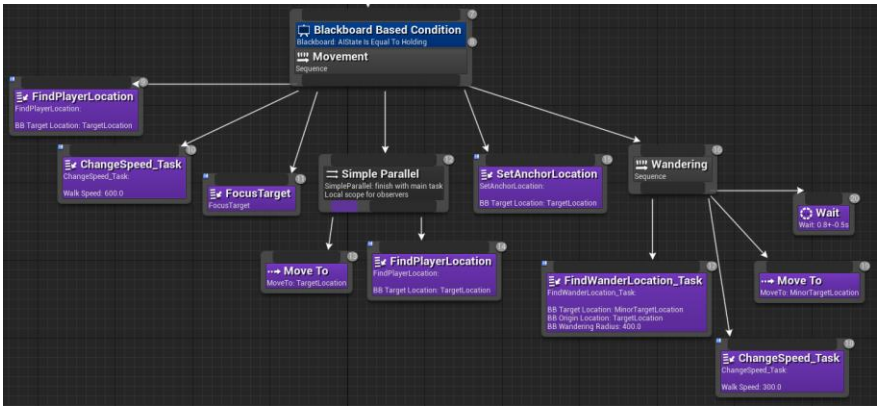

Figura 239. Behavior Tree – Holding

Un cop es dispara l'*Event* d'atac, explicat a l'Apartat 6.9.2.2.3, l'arbre executa el comportament de la Figura 240. En aquest cas es torna a guardar a *TargetLocation* la posició del jugador amb *FindPlayerLocation*. Després es canvia la velocitat a 800 amb *ChangeSpeed_Task*. A continuació assegura que el focus és correcte amb *FocusTarget* i executa la tasca *MoveTo*. Si arriba a una distància inferior a 40, executa un *Parallel* on utilitza l'*Attack_Target*, esmentat a l'Apartat 6.9.2.3.1. D'aquesta manera s'obté una major precisió a l'hora d'atacar, ja que si no, resulta massa fàcil d'esquivar. Finalment, executa *Reset_State*, explicat a l'Apartat 6.9.2.3.8, per tornar a l'estat Holding.

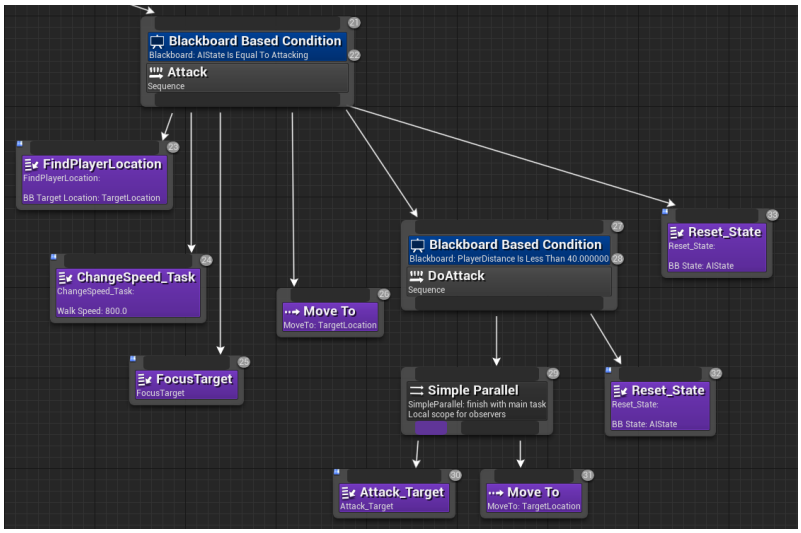

Figura 240. Behavior Tree – Attacking

6.9.3 Enemic final

La lògica de l'enemic final funciona igual que els *Minions*, ja que s'utilitzen un BT, un *Controller*, les *Tasks* i el Blueprint de l'enemic final, a partir d'ara *Boss*. Tot i així, no s'ha pogut reaprofitar els arxius anteriors, més enllà d'alguna tasca, perquè el comportament és diferent i té més estats. A diferència dels enemics bàsics, el *Boss* té diferents fases de combat:

- Primera fase: només utilitza el martell per atacar.
- Segona fase: té un 25% de possibilitats de llançar un atac a distància. Si no, ataca amb el martell.
- Tercera fase: té un 50% de possibilitats de llançar l'atac a distància. Si no, farà l'atac cos a cos.

En aquest cas s'inicia amb l'estat d'*Spawn*, on es fa una petita animació per aparèixer. Cal destacar que a l'hora del combat, principalment es tria entre dos estats: Holding (per l'atac cos a cos) i *SoulSiphon* (per l'atac a distància). Després de cada atac s'utilitza *AttackRecovery*, on s'executa una animació per donar espai al jugador per poder descansar i atacar. Quan acaba, es canvia a *ChangeState* per tornar a Holding o *SoulSiphon*. Finalment, quan mor canvia a *Dead*. Tots aquests valors es veuen definits a la Figura 241 i s'expliquen més a fons als següents apartats.

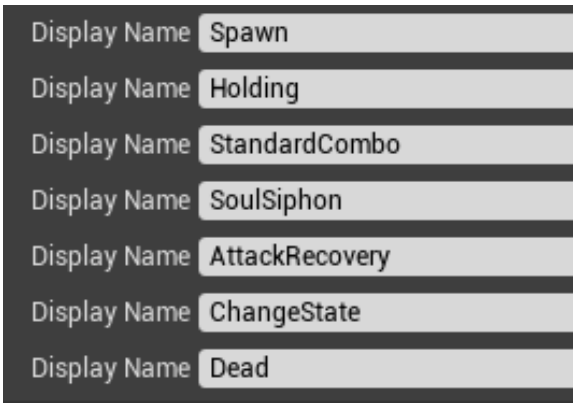

Figura 241. Valors de FinalBoss_AttackMode

6.9.3.1 FinalBoss

Aquest Blueprint és el propi del *Boss*. Conté les llistes de sons, el *Widget* de la barra de vida i les implementacions de les funcions d'*Enemy_Basic*, entre d'altres.

6.9.3.1.1 Construction

El *Construction Script* s'utilitza per executar quan el Blueprint es col·loca a l'escena. En aquest cas es fa servir per inicialitzar les tres llistes de sons: els d'*Spawn*, els de l'estat *SoulSiphon* i els de mort. Veure Figura 242.

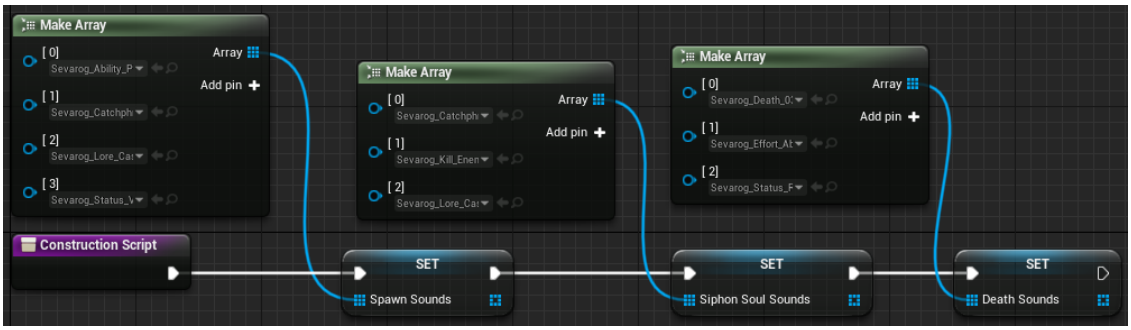

Figura 242. Construction Script

6.9.3.1.2 BeginPlay

Després de la construcció, s'executa el *BeginPlay*. Aquest executa un so de la llista d'*Spawn* i crida a les funcions *AddHealthBar* i *InitVariables*, explicades als apartats següents. El so es guarda a la variable *CurrentSound* perquè el valor enviat a *PlaySoundAtLocation* i *Duration* pot variar si s'extreu directament del *Random*. A més, quan el so es termina de reproduir, crida a *TryChooseNextState*, esmentat a l'Apartat 6.9.3.2.2. Veure Figura 243.

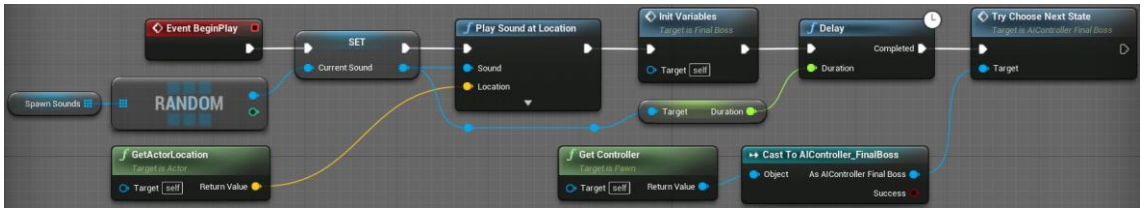

Figura 243. BeginPlay al FinalBoss

6.9.3.1.3 AddHealthBar

Aquesta funció crea la barra de vida del *Boss*, de la classe *BossHealthBar*. En guarda una referència al Blueprint i l'afegeix a la pantalla amb *AddToPlayerScreen*. Figura 244.

Figura 244. AddHealthBar

6.9.3.1.4 InitVariables

Inicialitza *Life* i *Attack*, que són d'*Enemy_Basic* i les pròpies del *FinalBoss*. Finalment, crida *SetStartStates* per iniciar la primera fase, explicada a l'Apartat 6.9.3.2.3. Veure Figura 245.

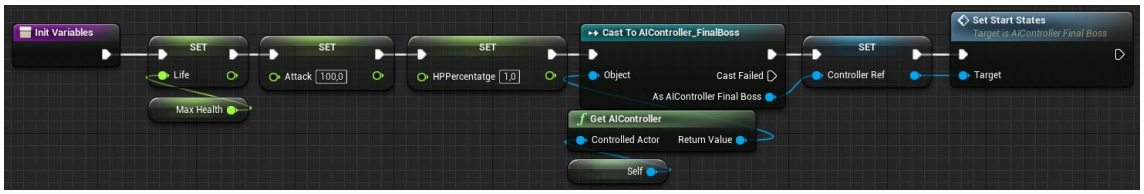

Figura 245. InitVariables

6.9.3.1.5 OnComponentBeginOverlap (HammerCollider)

Aquest *Event* s'activa quan un actor entra dins el Collider del martell. Es comprova que aquest actor sigui el jugador i que el *Boss* estigui executant l'atac cos a cos, tal com es veu a la Figura 246. Per saber que s'està en mig de l'atac amb martell s'utilitza la funció *IsOnStandardComboState*, mostrada a la Figura 248.

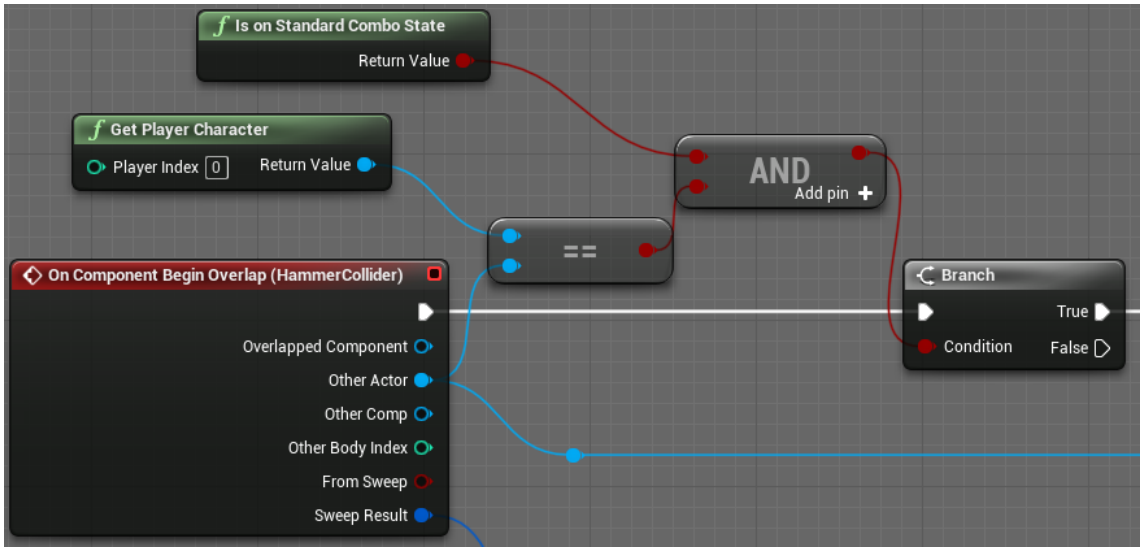

Figura 246. OnComponentBeginOverlap - Part 1

En cas que aquestes condicions es compleixin, crida a la funció *ReceiveDamage* per aplicar l'atac al jugador, esmentat a l'Apartat 6.4.1.3. A més, calcula el vector entre la posició del personatge i la posició del cop per llançar al personatge en aquesta direcció amb *LaunchCharacter*. Veure Figura 247.

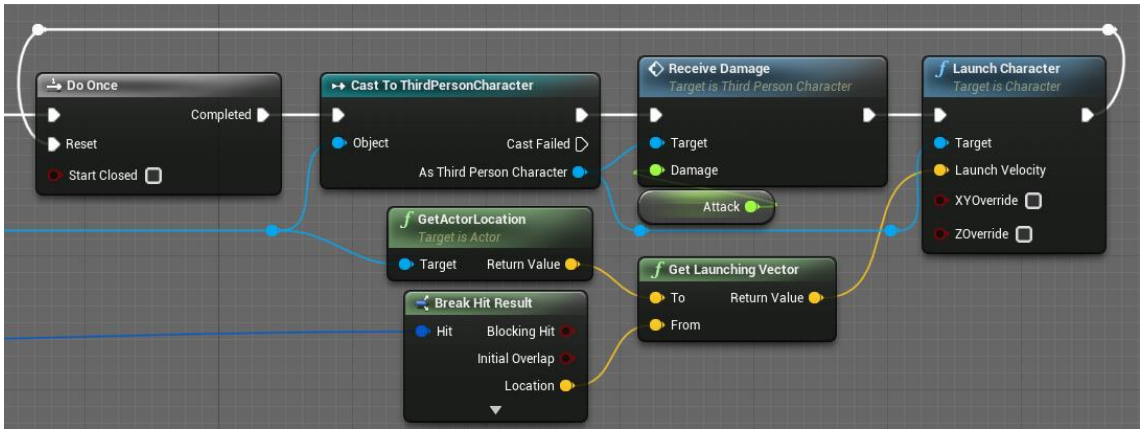

Figura 247. OnComponentBeginOverlap - Part 2

Per saber si s'està executant l'atac, s'obté l'*AttackState* del BT, esmentat a l'Apartat 6.9.3.4, i comprova si el valor és *StandardCombo*. Veure Figura 248.

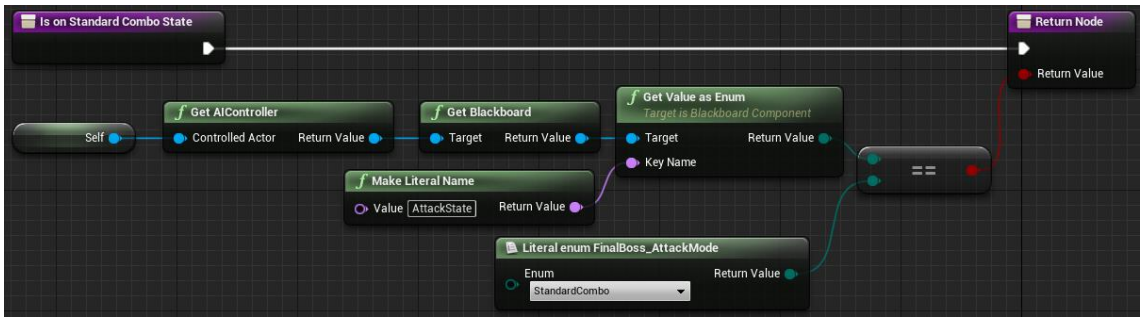

Figura 248. IsOnStandardComboState

6.9.3.1.6 PlayReceiveDamage

En aquest cas, no s'utilitza la funció de *PlayReceiveDamage* per mostrar una animació de rebre mal perquè així el Boss dona una sensació de poder. Tot i així, s'aprofita aquesta funció per actualitzar la barra de vida i, en cas que la vida baixi del 66 o del 33%, assignar el segon o tercer estat, respectivament. Tant *SetSecondPhaseStates* com *SetThirdPhaseStates* s'expliquen a l'Apartat 6.9.3.2.3. Veure Figura 249.

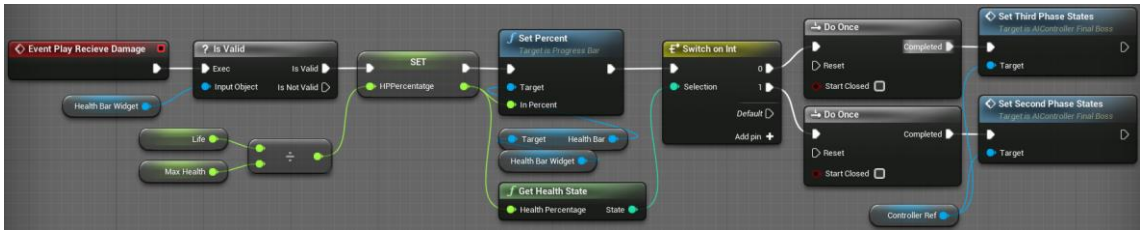

Figura 249. PlayReceiveDamage

6.9.3.1.7 Die

Quan l'enemic final mor es crida l'*Event* Die que només s'ha d'executar un cop, per això s'executa utilitzant el node *DoOnce*. El que es fa inicialment és posar el percentatge de vida a zero al HUD i s'indica al BT, esmentat a l'Apartat 6.9.3.4, que l'enemic ha mort. Veure Figura 250.

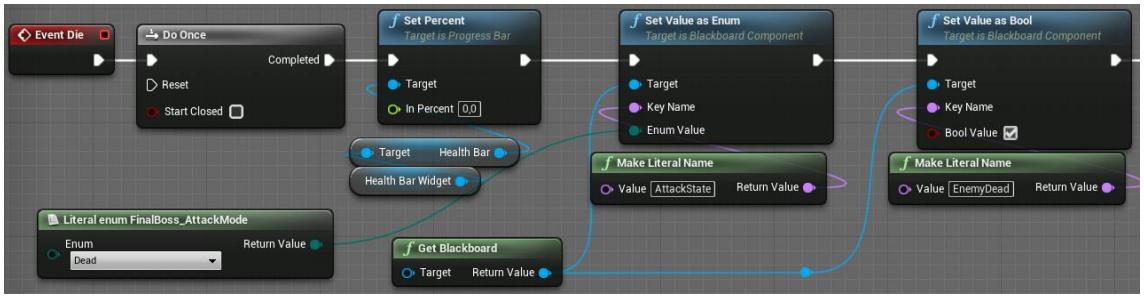

Figura 250. Die - Part 1

A continuació, s'indica també al *Blueprint* d'animacions, i s'atura el moviment amb *StopMovementImmediately*, una funció interna d'UE4. Després s'eliminen els possibles projectils restants amb *InterruptSoulSiphon*, explicada a l'Apartat 6.9.3.2.11, s'activa un efecte de partícules i s'elimina la barra de vida de la pantalla. Veure Figura 251.

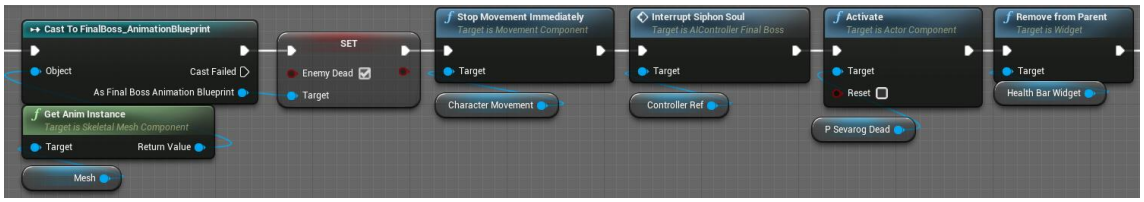

Figura 251. Die - Part 2

Finalment es crea un *Widget* amb els crèdits finals i ficant el joc en mode *UI Only* abans de portar-lo al menú d'inici. Veure Figura 252.

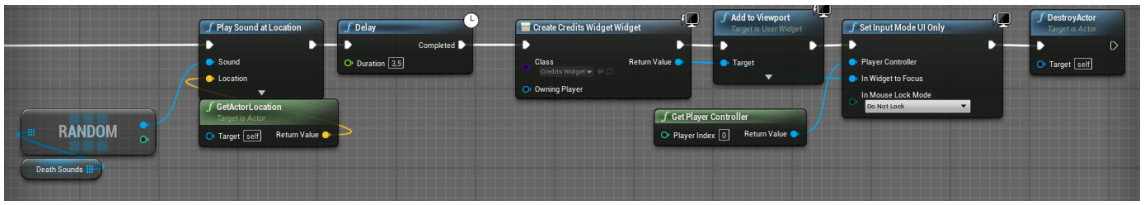

Figura 252. Die - Part 3

6.9.3.1.8 PlaySiphonSoulSound

Aquesta funció escull un so aleatori de la llista *SiphonSoulSounds* i el reprodueix. Veure Figura 253.

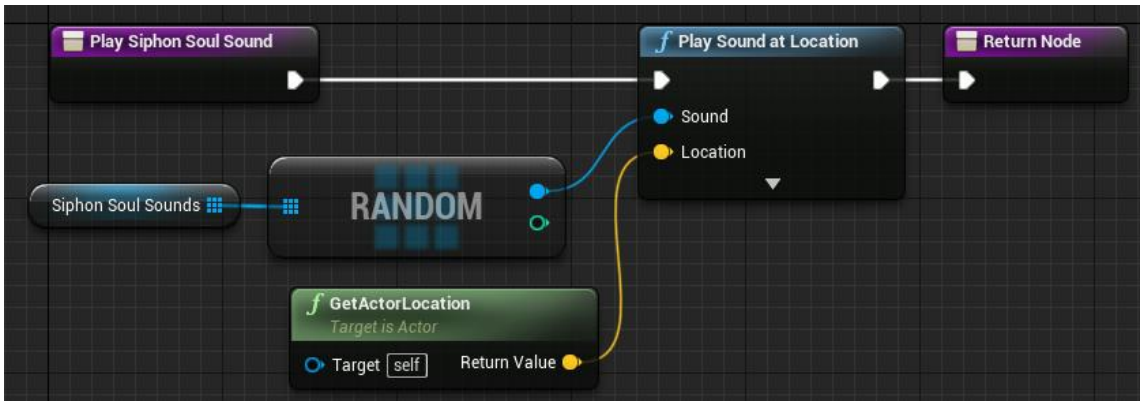

Figura 253. PlaySiphonSoulSound

6.9.3.2 AIController_FinalBoss

El *Controller* s'utilitza en el *Boss* per tal de controlar les decisions a l'hora de canviar d'estat i decidir els possibles estats quan s'inicia una nova fase.

6.9.3.2.1 BeginPlay

Quan s'inicia, es guarden unes referències al *FinalBoss* i l'*Animation Blueprint*, que permet triar les animacions que s'han d'executar durant els atacs. A més, inicia el comportament del *BT_FinalBoss*. Per acabar, deixa l'*AttackMode* com a *Spawn* i la variable *IsAttacking* a fals, per tal que faci l'animació corresponent sense atacar a l'usuari. Veure Figura 254 i Figura 255.

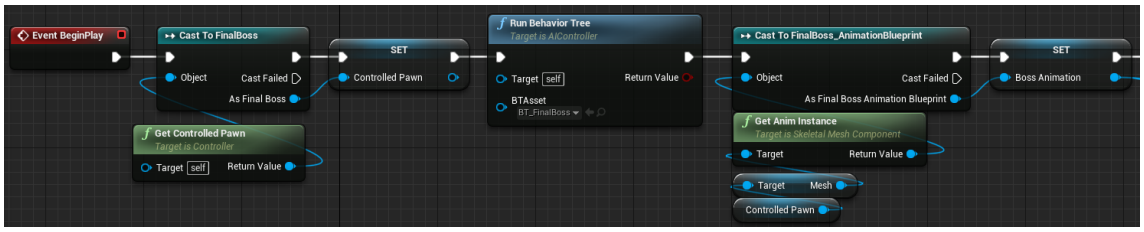

Figura 254. BeginPlay al Controller – Part 1

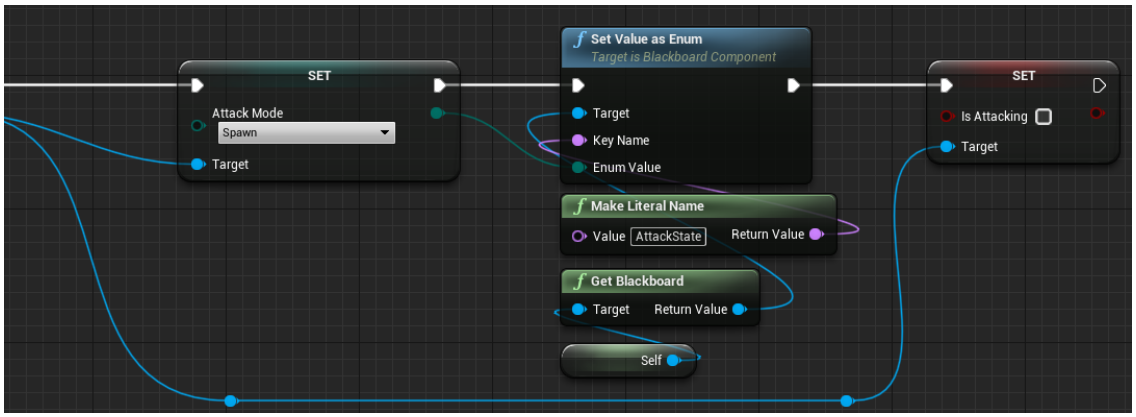

Figura 255. BeginPlay al Controller – Part 2

6.9.3.2.2 TryChooseNextState

Aquest *Event* intenta iniciar el combat, canviat l'estat del *Boss*, afegint la barra de vida i iniciant la música. Això només es podrà la segona vegada que es cridi aquesta funció: una a l'acabar l'animació inicial, i l'altra a l'acabar el so inicial. Veure Figura 256.

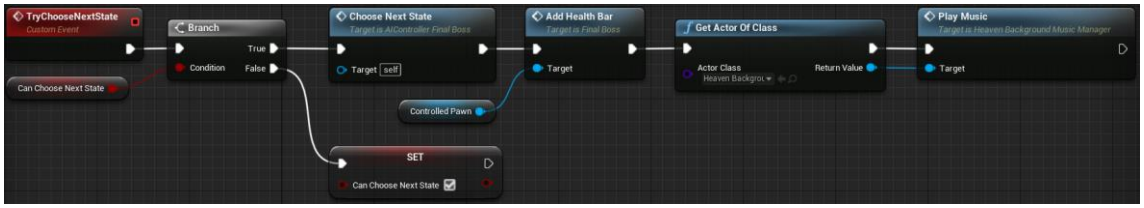

Figura 256. TryChooseNextState

6.9.3.2.3 SetPhases

En aquesta secció s'expliquen les funcions per inicialitzar cada fase. Serveixen per iniciar la primera, segona i tercera fase. Les tres realitzen la mateixa tasca: canviar el valor d'*StatesArray*. La diferència entre cadascuna és el valor que se li dona. Per la primera fase, s'utilitza *SetStartStates* (Figura 257) i en aquest cas es crea una llista amb una única entrada que té com a valor "Holding", ja que es vol que només ataqui cos a cos. Per la segona fase, es crida *SetSecondPhaseStates* (Figura 258) i la llista té quatre entrades on, tres d'elles tenen "Holding" com a valor i l'altra *SoulSiphon*. D'aquesta manera s'aconsegueix que l'atac a distància només tingui un 25% de possibilitats de ser utilitzat. Finalment, la tercera fase s'inicialitza amb *SetThirdPhaseStates* (Figura 259). En aquest cas la llista pren dues entrades: una per "Holding" i l'altra per *SoulSiphon*. Així s'obté un 50% per cada tipus d'atac.

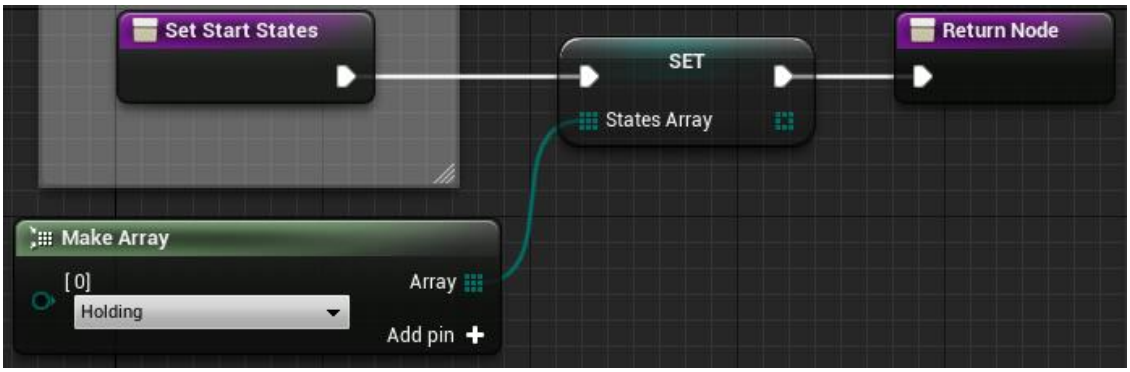

Figura 257. SetStartStates

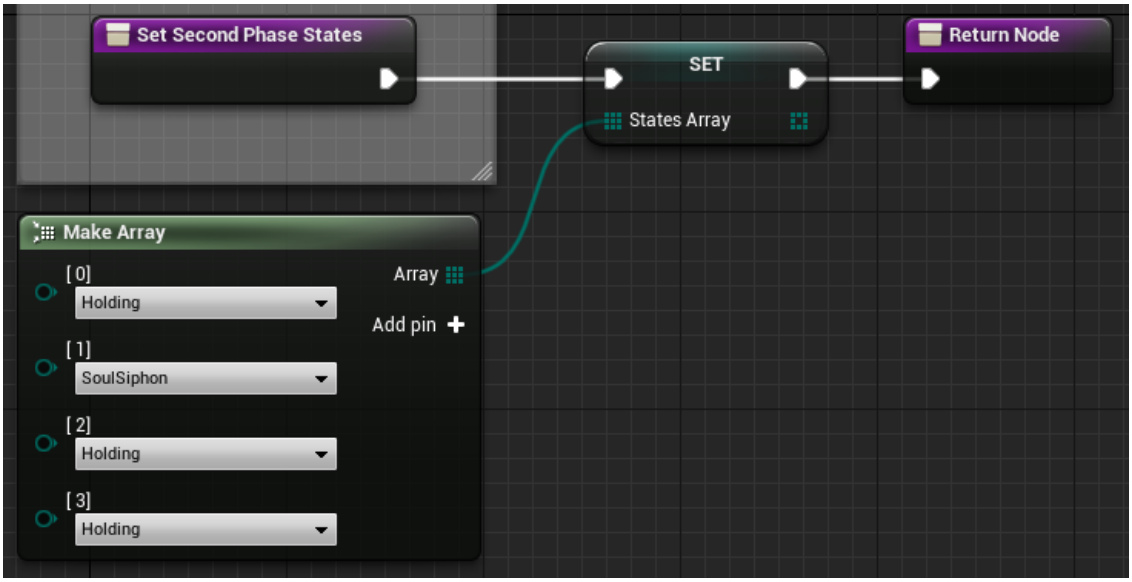

Figura 258. SetSecondPhaseStates

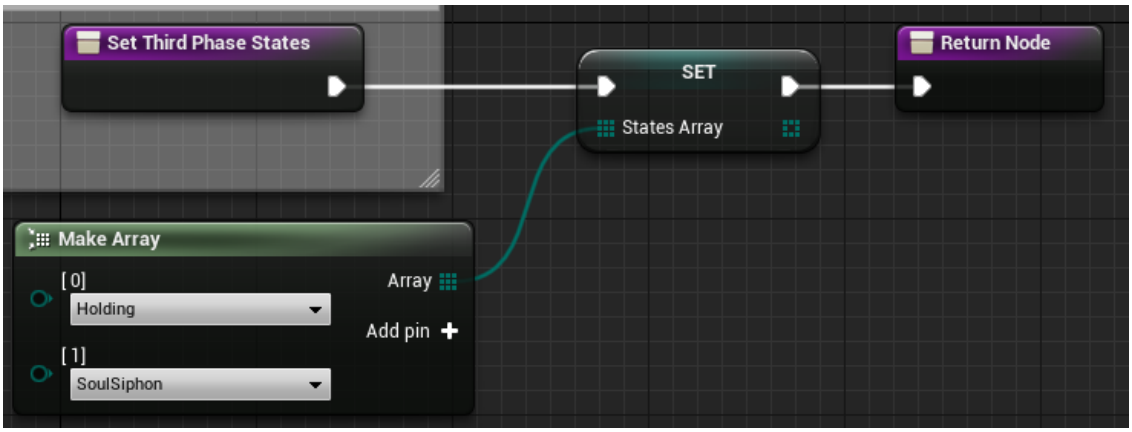

Figura 259. SetThirdPhaseStates

6.9.3.2.4 RestartBlackBoardParameters

Aquesta funció s'utilitza per indicar que ja no s'està executant la combinació cos a cos ni està disparant. Per fer-ho es posen les variables *IsAttacking* i *SoulSiphonCharged* a fals. Veure Figura 260.

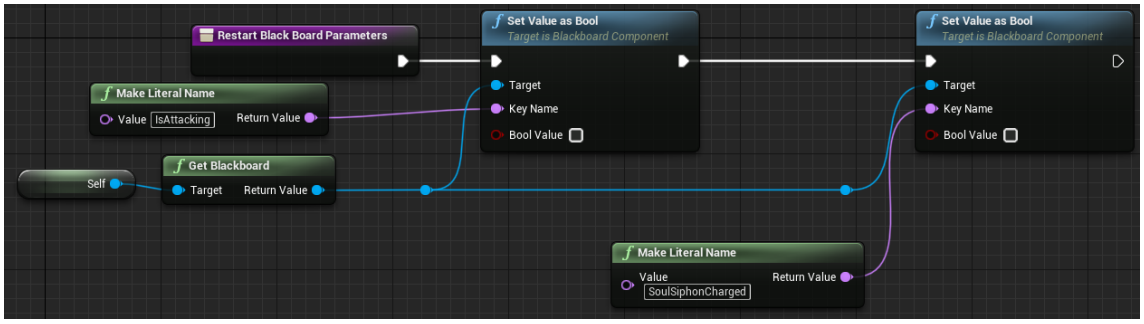

Figura 260. RestartBlackBoardParameters

6.9.3.2.5 ChooseNextState

Quan es finalitza l'animació de recuperació, és necessari escollir un nou estat d'atac. Per fer-ho s'utilitza aquesta funció, que agafa un valor aleatori d'*StatesArray* i el guarda a *NewState* i al BT. Finalment, reinicia els valor del BT amb *RestartBlackBoardParameters*, explicada a la secció anterior, i modifica les variables *IsAttacking*, *FireSoulSiphon* i *AttackMode* del *Blueprint* d'animacions. Cal destacar que la primera es posa a cert si és l'atac a distància perquè s'inicia al moment. En canvi, si s'ha de fer l'atac cos a cos, *IsAttacking* ha de ser fals perquè primer s'ha d'apropar al jugador. Veure Figura 261 i Figura 262.

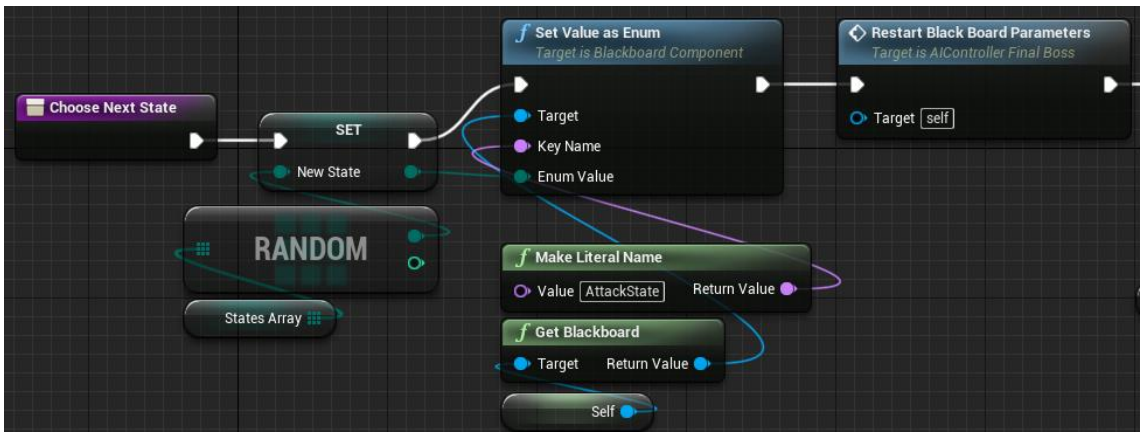

Figura 261. ChooseNextState – Part 1

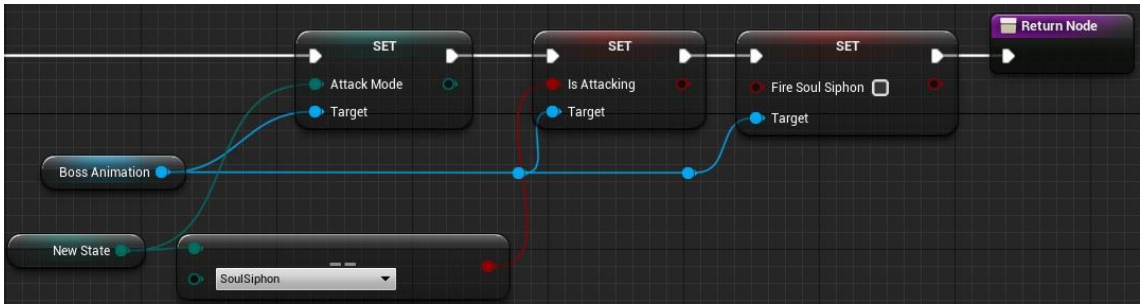

Figura 262. ChooseNextState – Part 2

6.9.3.2.6 SetRecovery

Un cop s'ha acabat de realitzar l'atac, es crida a la funció *SetRecovery*, que activa l'animació de recuperació. Com hi ha dues possibles, es tria quina s'ha de reproduir amb un booleà aleatori i posant l'*AttackMode* a *AttackRecovery*. Veure Figura 263.

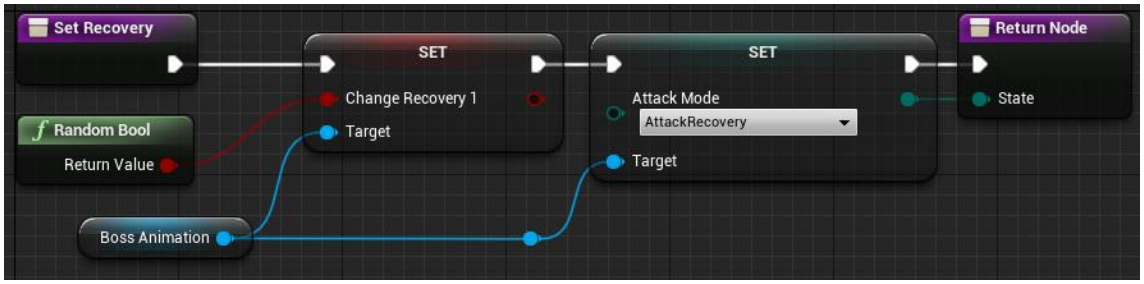

Figura 263. SetRecovery

6.9.3.2.7 MeleeAttack

Quan es decideix realitzar l'atac cos a cos es posa la variable *IsAttacking* a cert, tant a les animacions com al BT. En aquest estat, s'executen tres cops consecutius: un de dreta a esquerra, un d'esquerra a dreta i un de descendent. Veure Figura 264.

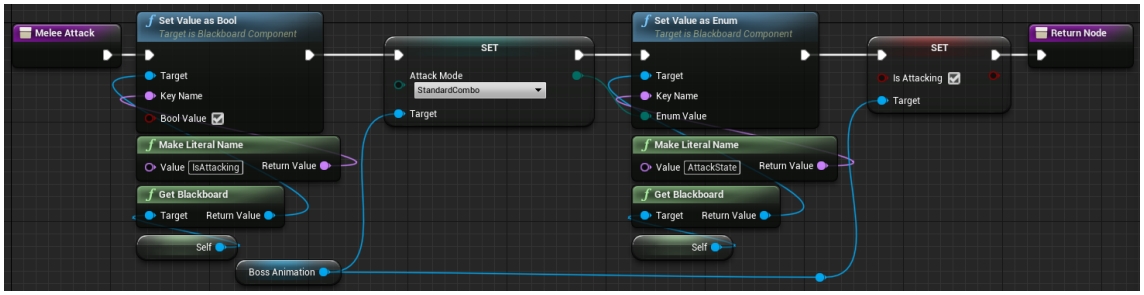

Figura 264. MeleeAttack

6.9.3.2.8 ChangeToMeleeState

Quan el *Boss* s'apropa suficientment al jugador (utilitzant l'estat de Holding), es canvia l'estat a *StandardCombo* per tal que el BT canviï de seqüència. Per fer-ho es canvien les variables d'estat a *StandardCombo* i a les animacions es posa *FireSoulSiphon* a fals i *IsAttacking* a cert, per assegurar que es reproduiran les animacions d'atac cos a cos i no les de l'altre atac. Veure Figura 265.

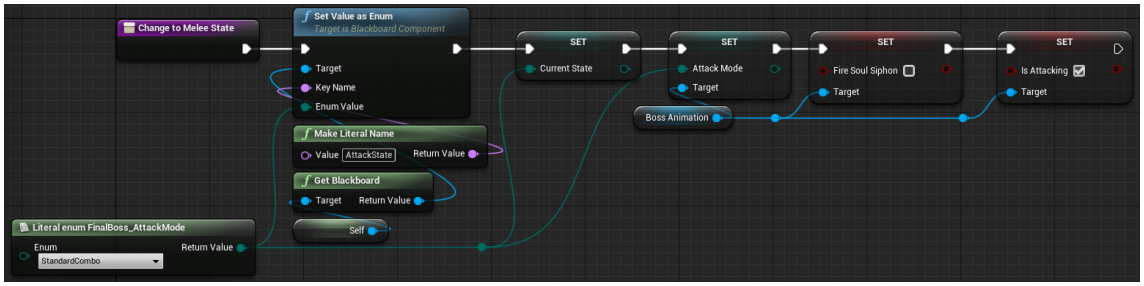

Figura 265. ChangeToMeleeState

6.9.3.2.9 ChargeRangedAttack

Per poder fer l'atac a distància, primer de tot cal crear els projectils. Així doncs, aquesta funció assegura que tant BT com les animacions tinguin *IsAttacking* a cert i, per evitar errors, buida la llista de projectils. A continuació reprodueix un so amb la funció *PlaySiphonSoulSound*, esmentada a l'Apartat 6.9.3.1.8. Veure Figura 266.

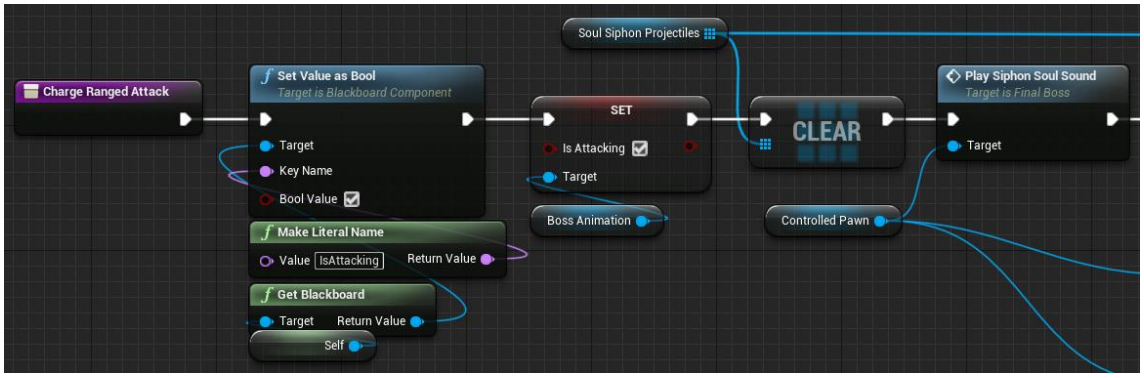

Figura 266. ChargeRangedAttack - Part 1

Per tal de crear tots els projectils, es realitza un For que va de -2 a 2 (d'aquesta manera s'aconsegueix que aparegui un projectil al centre i dos a cada costat). Dins d'aquest bucle es crea un objecte de la classe *BossProjectile*, explicada a l'Apartat 6.9.3.5. Per calcular la posició s'obté la posició del *Boss* i se li suma 400 a l'eix de les z (així apareix a sobre del seu cap) i el valor de l'índex multiplicat per 200 i pel vector *right* del *Boss*. D'aquesta manera s'aconsegueix que apareguin cinc projectils separats i en fila sobre l'enemic. Veure Figura 267.

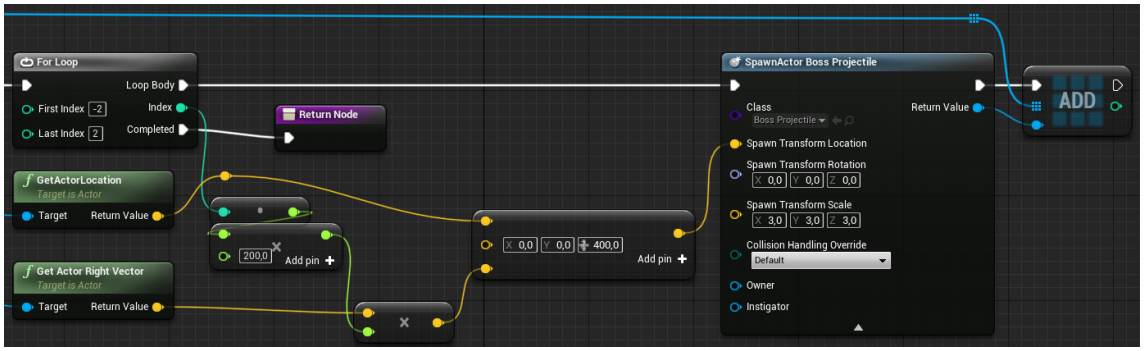

Figura 267. ChargeRangedAttack - Part 2

6.9.3.2.10 FireRangedAttack

Un cop l'enemic ha acabat de carregar els projectils, es crida la funció *FireRangedAttack* que es la responsable d'indicar al BT i al *Blueprint* d'animacions que el *Boss* ja pot disparar els projectils. Veure Figura 268.

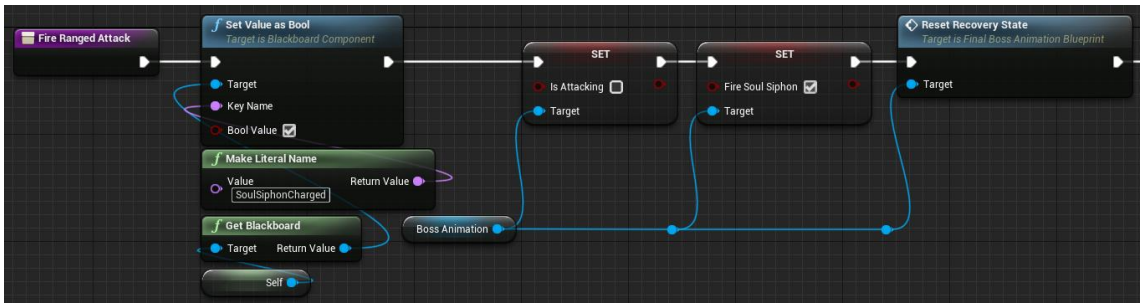

Figura 268. FireRangedAttack - Part 1

Finalment, es fa un recorregut per la llista de projectils i s'executa la funció *LaunchProjectile*, esmentada a l'Apartat 6.9.3.5.4. Per a aquest prototip s'ha decidit que tots els projectils es dispararan de cop, per això s'envia un *Delay* de zero. Veure Figura 269.

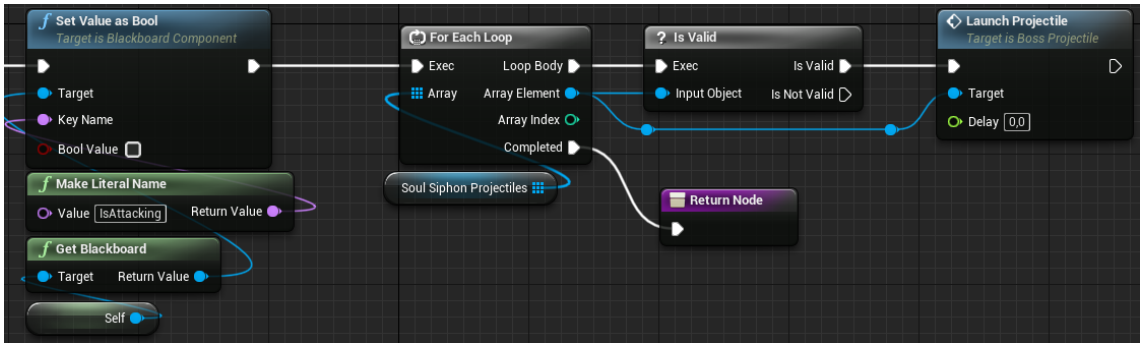

Figura 269. FireRangedAttack - Part 2

6.9.3.2.11 InterruptSiphonSoul

Aquesta funció està pensada especialment per si el *Boss* mor mentre està carregant els projectils. La seva tasca es comprovar si a la llista queda algun objecte i, si és així, la recorre per eliminar-los. Veure Figura 270.

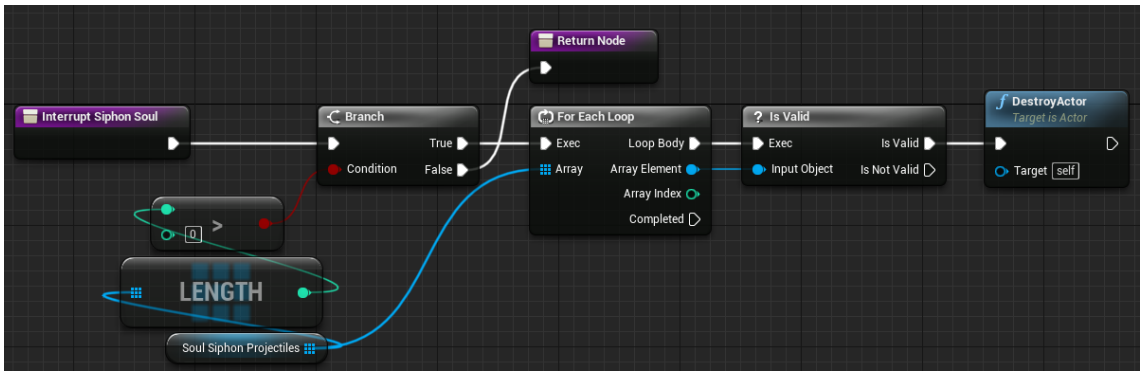

Figura 270. InterruptSiphonSoul

6.9.3.2.12 EndAttack

Un cop ha acabat l'animació pertinent d'atac, s'executa aquesta funció que modifica les variables del BT i *Animation* per indicar que ara s'ha de fer l'animació de recuperació. Utilitza la funció *SetRecovery*, veure Apartat 6.9.3.2.6, per obtenir el nou estat. Veure Figura 271.

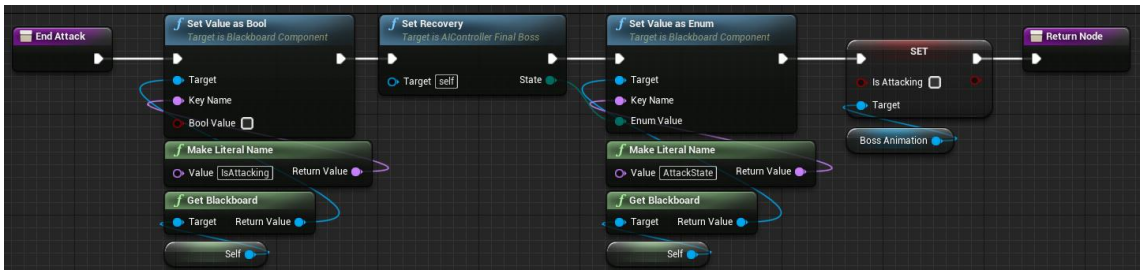

Figura 271. EndAttack

6.9.3.3 *Tasks*

Igual que amb els *Minion*, esmentats a l'Apartat 6.9.2.1, les *Task* s'utilitzen al BT per indicar què s'ha de fer dins de cada *Sequence*. En aquest cas s'han pogut reaprofitar algunes de les ja definides, però cal dir que s'han definit algunes amb un nom molt similar perquè el *Boss* no requereix de tanta complexitat a la implementació, ja que no té un sistema de detecció del jugador.

6.9.3.3.1 CastSoulSiphon

Aquesta tasca obté el controlador per tal de començar a carregar els projectils, utilitzant *ChargeRangedAttack*, explicada a l'Apartat 6.9.3.2.9. Veure Figura 272.

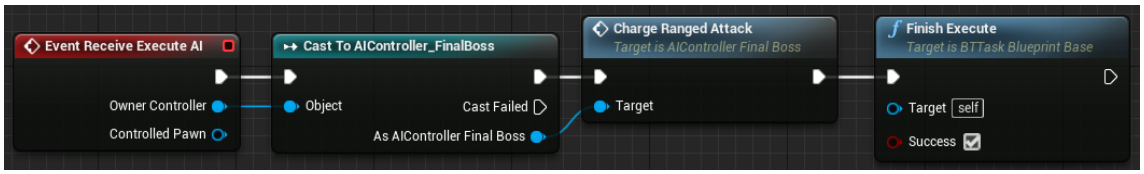

Figura 272. CastSoulSiphon

6.9.3.3.2 ChangeToAttack

Un cop s'arriba a la posició del jugador, es crida a la funció *ChangeToMeleeState* del controlador, esmentada a l'Apartat 6.9.3.2.8, per iniciar la seqüència d'atacs. Veure Figura 273.

Figura 273. ChangeToAttack

6.9.3.3.3 FindPlayerLocation_Task

Aquesta tasca és molt similar a la utilitzada al *Minion*, anomenada *FindPlayerLocation* (veure Apartat 6.9.2.3.3), però com es pot apreciar a la Figura 274, és més simple ja que sempre s'ha d'assignar la posició del jugador.

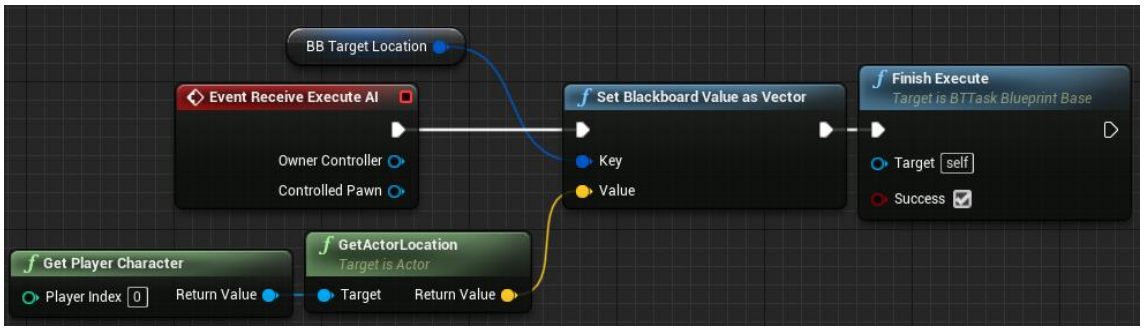

Figura 274. FindPlayerLocation_Task

6.9.3.3.4 FireSiphonSoul

Un cop carregats els projectils, s'utilitza aquesta *Task* per cridar a la funció *FireRangedAttack*, explicada a l'Apartat 6.9.3.2.10, i disparar-los. Veure Figura 275.

Figura 275. FireSoulSiphon

6.9.3.3.5 MeleeAttack_Task

Aquesta *Task* és qui té la funció de notificar al controlador que ja s'ha d'iniciar la combinació d'atacs cos a cos, utilitzant la funció *MeleeAttack*, esmentada a l'Apartat 6.9.3.2.7. Veure Figura 276.

Figura 276. MeleeAttack_Task

6.9.3.4 BT_FinalBoss

De mateixa manera que amb el BT del *Minion*, desenvolupat a l'Apartat 6.9.2.4, s'ha definit un arbre amb els estats de l'actor, en aquest cas el *Boss*, i què ha de fer en cadascun. Es pot apreciar una vista general de l'arbre a la Figura 277. A més, s'han definit les *Keys* que es poden apreciar a la Figura 278.

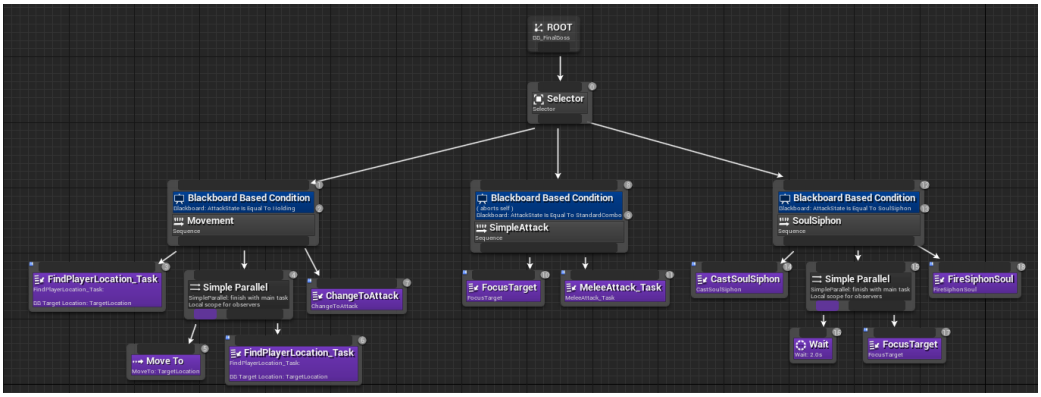

Figura 277. Boss Behavior Tree - Vista General

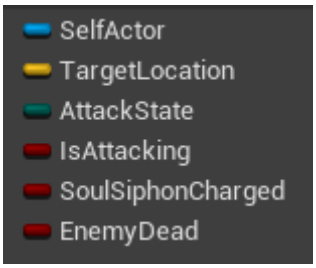

Figura 278. Keys del Blackboard

El primer estat del BT és Holding. En aquest cas es busca la posició del jugador amb *FindPlayerLocation_Task*, veure Apartat 6.9.3.3.3, i s'executa un *Parallel* per desplaçar-se fins l'objectiu. Quan s'apropa suficient, es realitza una crida a *ChangeToAttack*, veure Apartat 6.9.3.3.2, per canviar *AttackState* a *StandardCombo*. Veure Figura 279.

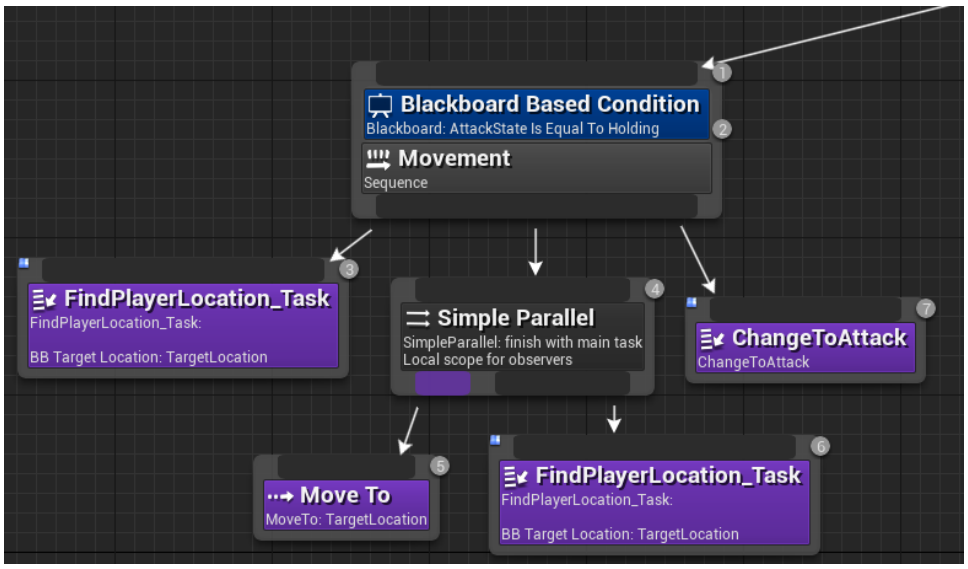

Figura 279. Boss Behavior Tree – Holding

Dins d'aquesta *Sequence* s'utilitza la *Task FocusTarget* per assegurar que s'apunta a l'objectiu i es crida a la tasca *MeleeAttack_Task*, explicada a la secció anterior, per iniciar els cops amb el martell. Veure Figura 280.

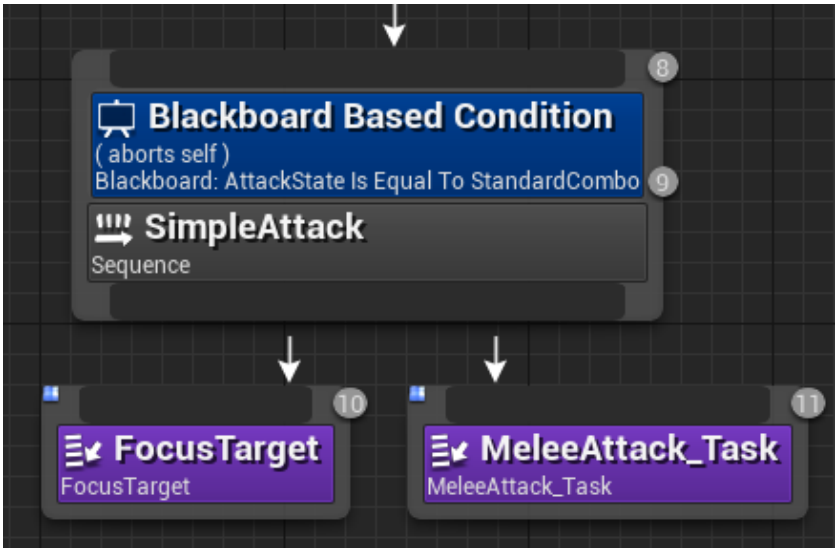

Figura 280. Boss Behavior Tree – StandardCombo

Finalment, la tercera *Sequence* definida al BT és la de *SoulSiphon*. Per aquest estat es crida a *CastSoulSiphon*, esmentada a l'Apartat 6.9.3.3.1, i s'executa un *Parallel* amb un *Wait* i un *FocusTarget*. Això permet que el *Boss* carregui els projectils sense perdre de vista al jugador. Quan han passat els dos segons, dispara els projectils amb la tasca *FireSiphonSoul*, explicada a l'Apartat 6.9.3.3.4. Veure Figura 281.

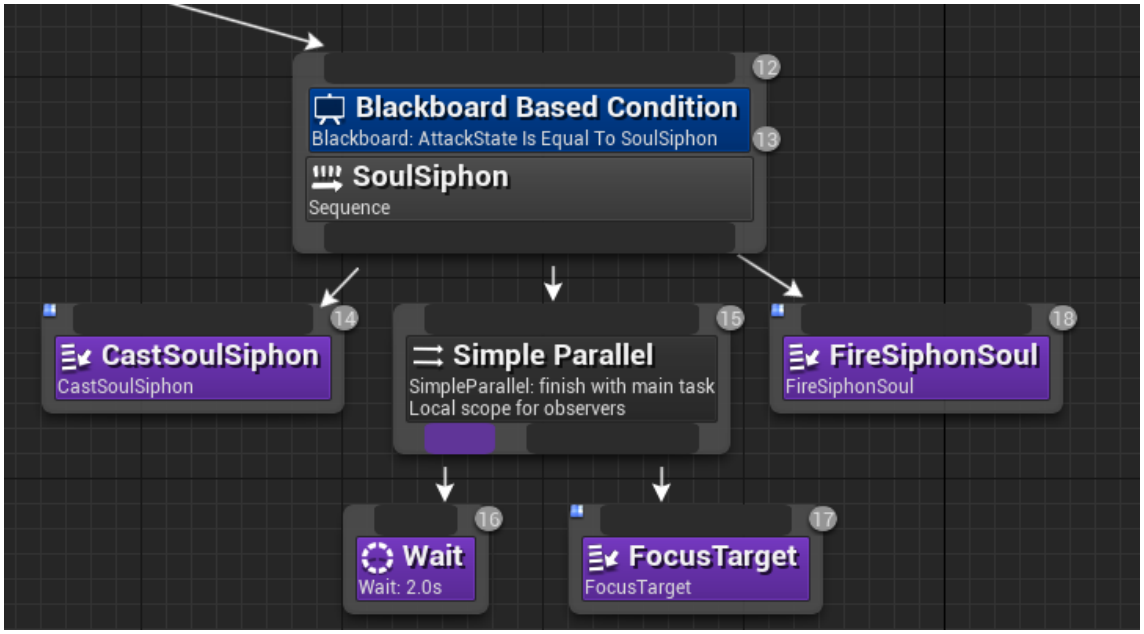

Figura 281. Boss Behavior Tree - SoulSiphon

La resta d'estats no requereixen de cap comportament, ja que s'ha de reproduir una animació sense alterar la posició. Per aquest mateix motiu no tenen cap *Sequence* associada.

6.9.3.5 BossProjectile

Aquest *Blueprint* és el que conté les partícules del projectil i gestiona el moviment des que és disparat fins que es destrueix.

6.9.3.5.1 BeginPlay

Quan es crea el projectil, es calcula un Offset aleatori en un rang entre 500 i -500, tant per l'eix de les x com per l'eix de les y. A continuació crida l'*Event* ChargeAttack, desenvolupat a la següent secció, i guarda la distancia a l'objectiu a *LastDistance* amb *DistanceToDecal*, explicada a l'Apartat 6.9.3.5.3. Veure Figura 282.

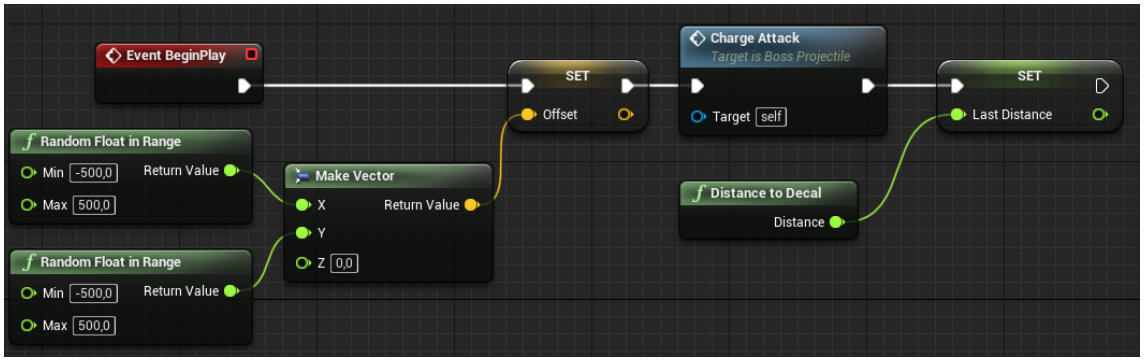

Figura 282. BeginPlay al BossProjectile

6.9.3.5.2 ChargeAttack

Aquest *Event* té la tasca de buscar l'objectiu del projectil per tal d'assignar una direcció. Quan s'inicia, es calcula una posició objectiu utilitzant l'*Offset*, prèviament calculat, i la posició del jugador. Amb aquests dos punts, es crea un vector que ignora al jugador per obtenir una col·lisió amb el terra. Veure Figura 283.

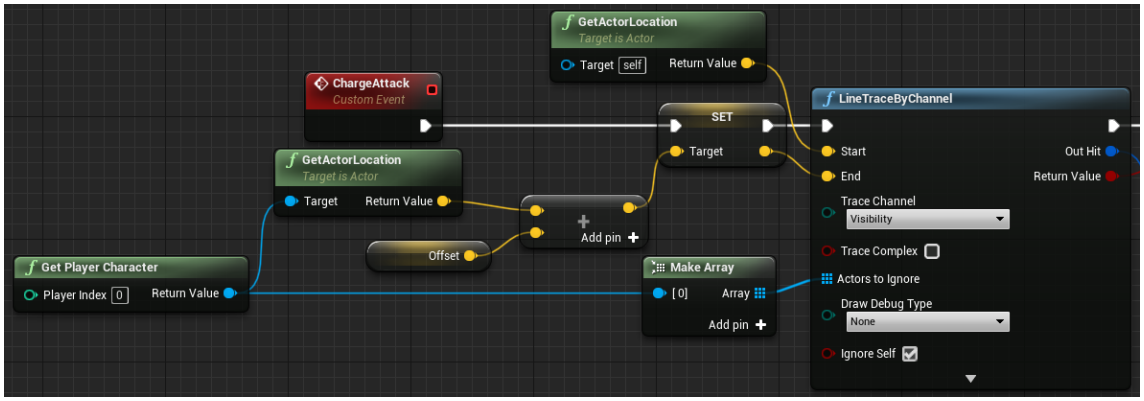

Figura 283. ChargeAttack - Part 1

Un cop s'aconsegueix un impacte, s'obté el punt de col·lisió per guardar el verdader objectiu. En aquest punt es crea un *Decal* de la classe *RangedAttackDecal*. Un *Decal* és una imatge que es renderitza sobre una superfície (en aquest cas s'ha creat un gràfic d'un cercle). Es guarda una referència del *Decal* i s'utilitza el *Delay* per esperar a ser disparat. Veure Figura 284.

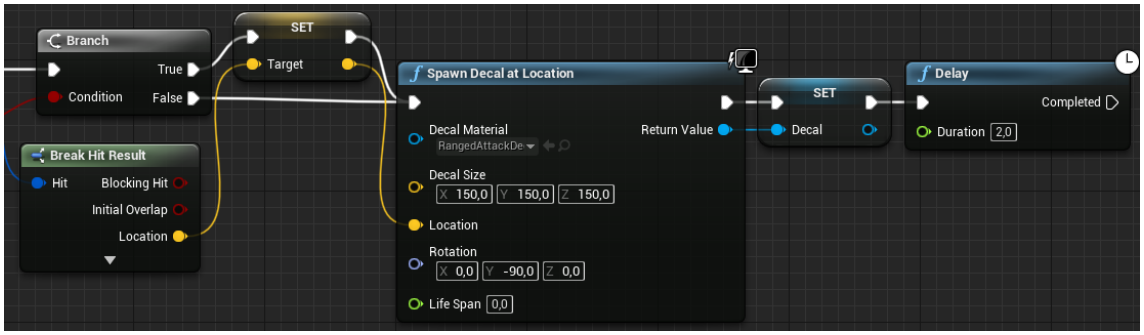

Figura 284. ChargeAttack - Part 2

6.9.3.5.3 DistanceToDecal

Aquesta funció obté la posició del projectil i la del *Decal* creat i en calcula la distància, la qual es retorna. Veure Figura 285.

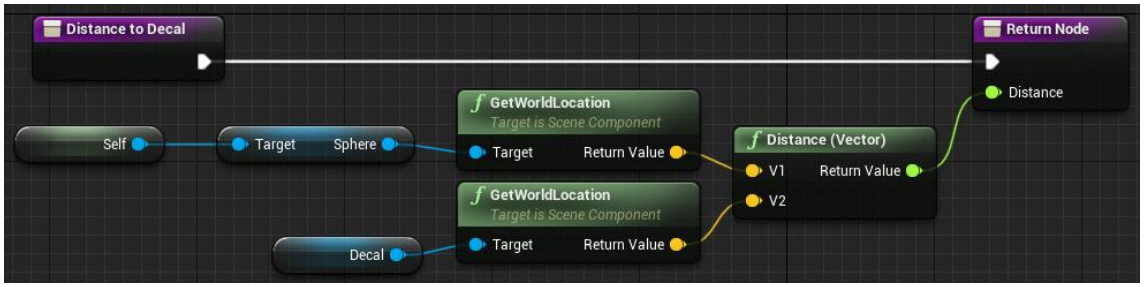

Figura 285. DistanceToDecal

6.9.3.5.4 LaunchProjectile

L'*Event LaunchProjectile* rep com a paràmetre el temps que ha d'esperar i, inicialment, executa un Delay amb aquest temps. A continuació activa les col·lisions de l'esfera i el moviment del component *ProjectileMovement*. Finalment, es calcula un vector entre la posició de l'esfera i l'objectiu per aplicar una velocitat. En aquest cas es multiplica per 1500 per obtenir una velocitat més adequada. Veure Figura 286.

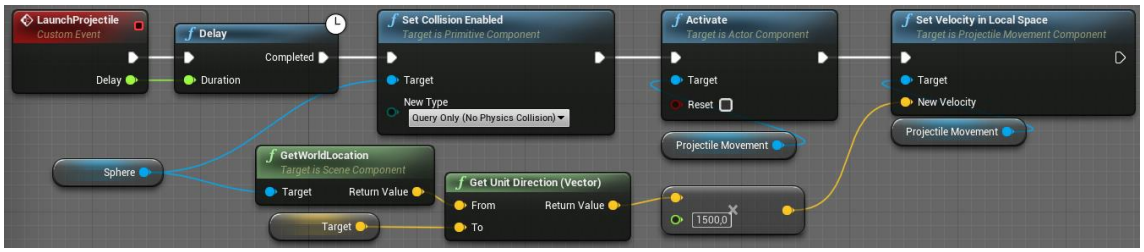

Figura 286. LaunchProjectile

6.9.3.5.5 Tick

L'*Event Tick*, en aquest cas, té dues funcions principals. La primera és recalcular la distància amb el *Decal*. Quan s'observa que ha augmentat en comptes de disminuït, es destrueix el projectil amb la funció *DestroyProjectile*, explicada a la següent secció. L'altra tasca és calcular el temps que porta creat el projectil. Inicialment es té activat el component *IsHomingProjectile* per donar una acceleració inicial, però 0.5 segons després s'ha de desactivar ja que a llarg termini es desitja que la velocitat del projectil sigui constant. Veure Figura 287 i Figura 288.

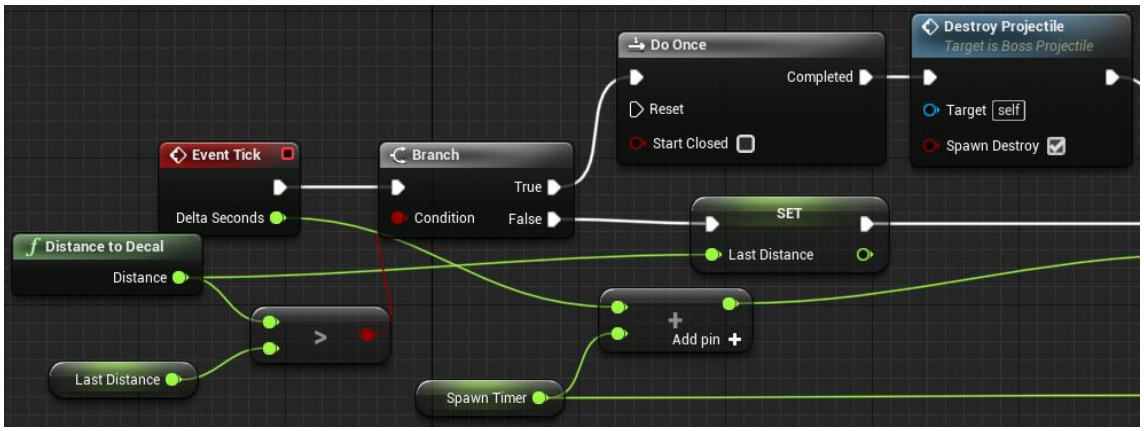

Figura 287. Tick al BossProjectile - Part 1

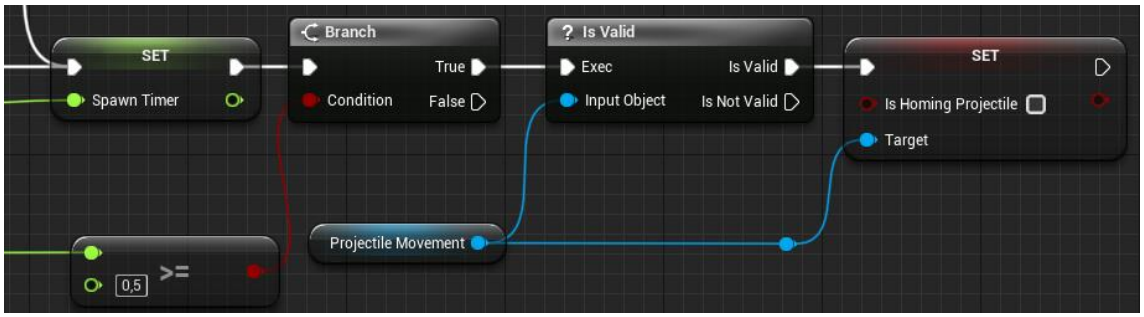

Figura 288. Tick al BossProjectile - Part 2

6.9.3.5.6 DestroyProjectile

Aquesta funció destrueix tant el component del *Decal* com el propi projectil un cop es crida. A més, en cas que *SpawnDestroy* sigui cert, crea un objecte de la classe *SoulSiphonDestroy*, esmentat a la següent secció. Per destruir s'utilitza primer el node *IsValid*, per evitar errors. Veure Figura 289.

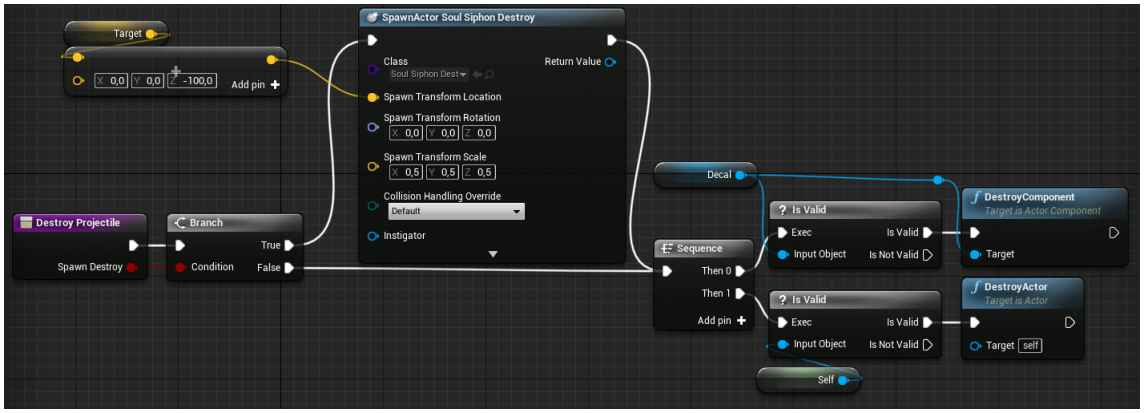

Figura 289. DestroyProjectile

6.9.3.6 SoulSiphonDestroy

Aquesta classe conté un *BoxCollider* i un sistema de partícules per tal de poder donar un efecte visual quan es destrueix un projectil. Per aquest motiu, quan es crea, es reprodueix un so d'explosió, s'espera tres segons i, finalment, es destrueix. Veure Figura 290.

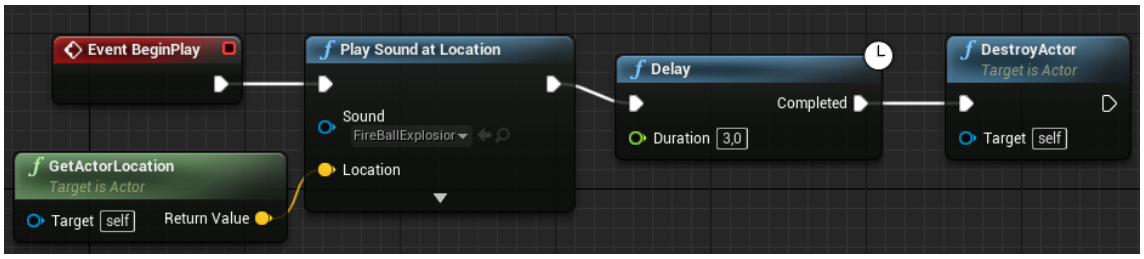

Figura 290. BeginPlay al SoulSiphonDestroy

Mentre està en actiu, es comprova si el jugador entra dins el C*ollider* i, si és així, crida a la funció *ReceiveDamage*, esmentada a l'Apartat 6.4.1.3. Veure Figura 291.

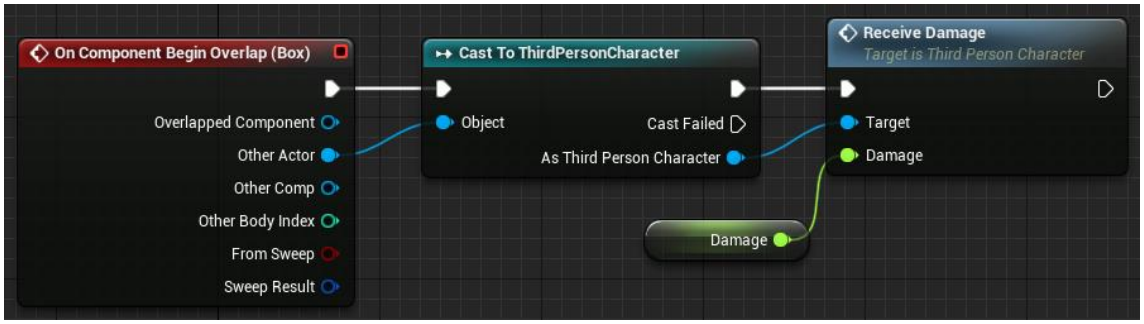

Figura 291. Overlap al SoulSiphonDestroy

6.10 Sistema de Marcatge d'Enemics

El sistema de marcatge és un sistema que determina qui és l'objectiu a atacar. És a dir, gestiona l'atribut *currentEnemy* que ja s'ha mencionat diverses vegades. *currentEnemy* és l'enemic a qui el jugador està apuntant els atacs. Aquest sistema té un efecte molt important sobre els atacs, ja que el desplaçament i direcció del jugador depèn completament de a qui s'està atacant. El sistema, per sí sol, determina un enemic com a l'objectiu. Es fa a partir de si l'enemic està suficientment a prop, quin és l'enemic més proper, i si el jugador l'està veient a la pantalla. A més, permet al jugador fixar un enemic com a l'objectiu de forma permanent, fins que aquest mori o es cancel·li com a objectiu permanent. Mentre un enemic sigui objectiu permanent, la càmera mirarà sempre en la seva direcció.

6.10.1 Variables del jugador

Per a implementar el sistema de marcatge, cal que el jugador tingui diverses variables per a gestionar quin serà l'objectiu. Algunes d'aquestes ja s'han mencionat a apartats anteriors, perquè, com ja s'ha dit, afecta a diverses part del joc. Les variables en qüestió són les que es veuen en la Figura 292.

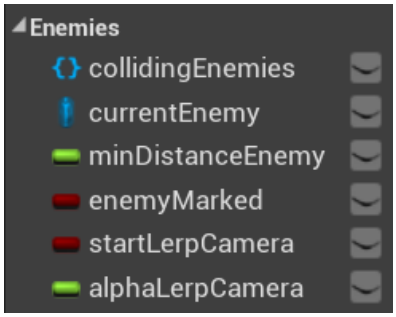

Figura 292. Sistema de Marcatge - Variables

collidingEnemies són tots aquells enemics que estan suficientment a prop com per a poder considerar-los objectius, *currentEnemy* és l'enemic objectiu, *minDistanceEnemy* és la distància a l'enemic més proper, *enemyMarked* indica si algun enemic és objectiu permanent, i *startLerpCamera* i *alphaLerpCamera* serveixen per girar de forma suau en direcció a l'objectiu permanent.

6.10.2 Visualització del Marcatge

Per a saber a qui s'està prenent com a objectiu en cada moment, els enemics tenen certs elements necessaris. Aquests estan afegits a *EnemyBasic*, i s'expliquen a continuació.

6.10.2.1 EnemyMarked_Nameplate

Aquest és un *Widget* senzill que s'afegeix a l'estructura d'*EnemyBasic* per a tenir un element visual que indiqui a qui s'està apuntant. Es tracta simplement d'un text de color (vermell si es tracta d'objectiu permanent, i lila si no) que posa "ENEMY MARKED", tal com es pot veure a la Figura 293.

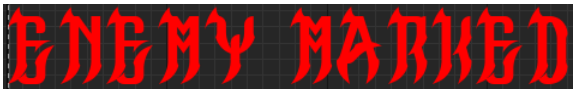

Figura 293. Sistema de Marcatge – EnemyMarked_Nameplate

6.10.2.2 I_Enemy

Aquesta és un interfície que genera dues funcions: Mark, amb el paràmetre d'entrada *forceMark*, i *Unmark*. Aquesta interfície s'afegeix a *EnemyBasic*, a on s'haurà d'implementar. També se li afegeix un paràmetre *isMarked* que indica si l'enemic és objectiu o no.

6.10.2.2.1 Mark

Aquest *Event* es crida quan s'escull l'enemic com a objectiu. El què fa és activar la visibilitat de l'*EnemyMarked_Nameplate* i posar-li el color indicat segons si el paràmetre *forceMark* és cert o fals. Veure Figura 294.

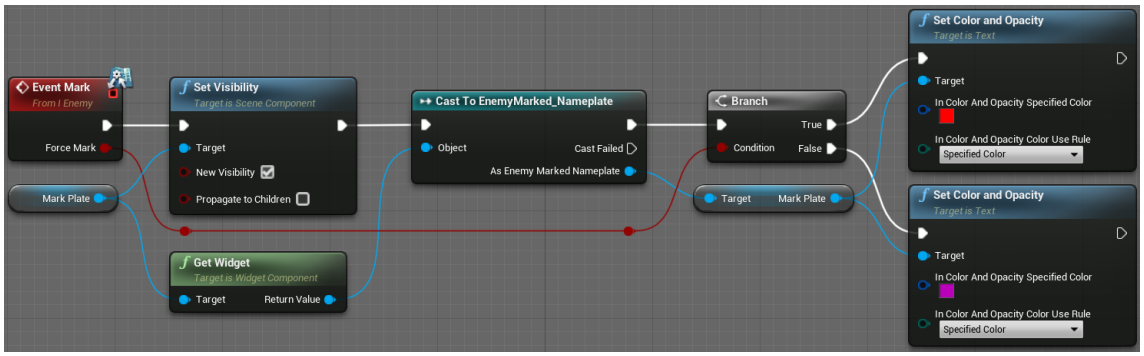

Figura 294. Mark

6.10.2.2.2 Unmark

Aquest *Event* es crida quan l'enemic deixa de ser objectiu. Simplement desactiva la visibilitat de l'*EnemyMarked_Nameplate*, tal com es pot veure a la Figura 295.

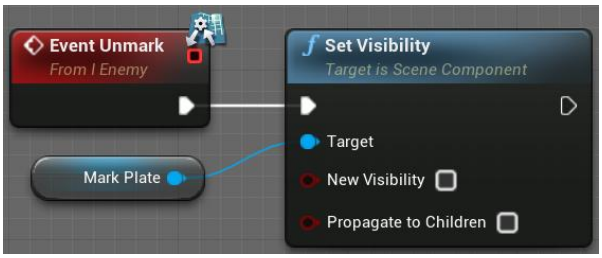

Figura 295. Unmark

6.10.3 Detecció i Marcatge d'Enemics

La detecció d'enemics no és més que un sistema fet per a saber quins enemics estan suficientment a prop com per a poder-los considerar objectius. Per fer-ho, s'afegeix al personatge una esfera gegant (anomenada *EnemiesDetection*) i, a partir de les funcions d'inici i final d'intersecció, es manté una llista dels enemics que estan a prop.

A partir de la detecció, es pot fer el sistema de marcatge. Aquest s'encarrega de determinar quin serà l'objectiu a qui atacar. Com ja s'ha explicat a la introducció de l'Apartat 6.10, hi ha un sistema automàtic per determinar l'objectiu segons la distància i si esta dins la pantalla, i un sistema manual per forçar que el personatge tingui com a objectiu fixe un enemic en concret.

6.10.3.1 OnComponentBeginOverlap (EnemiesDetection)

Aquest *Event* es crida quan un objecte comença a interseccionar amb l'objecte de detecció. El què fa és afegir l'objecte a la llista, si és que es tracta d'un *EnemyBasic*. Veure Figura 296.

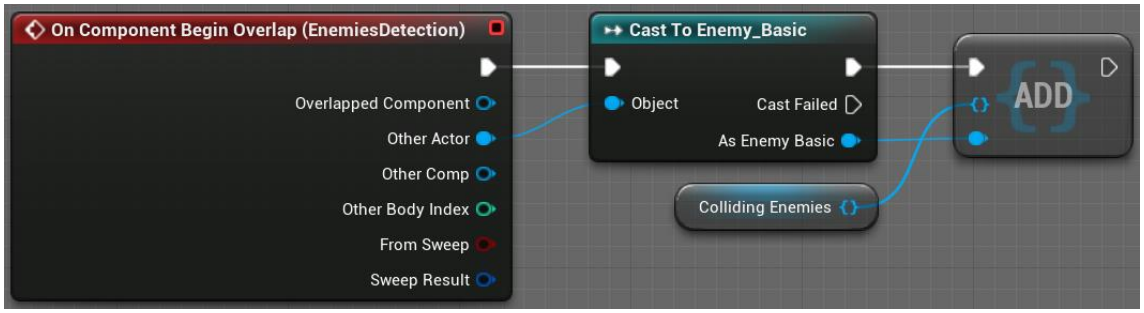

Figura 296. OnComponentBeginOverlap (EnemiesDetection)

6.10.3.2 OnComponentEndOverlap (EnemiesDetection)

Aquest *Event* es crida quan un objecte deixa d'interseccionar amb l'objecte de detecció. S'encarrega d'eliminar de la llista l'objecte, si és que es tracta d'un *EnemyBasic*. A més, si l'enemic és objectiu, el desmarca. Veure Figura 297.

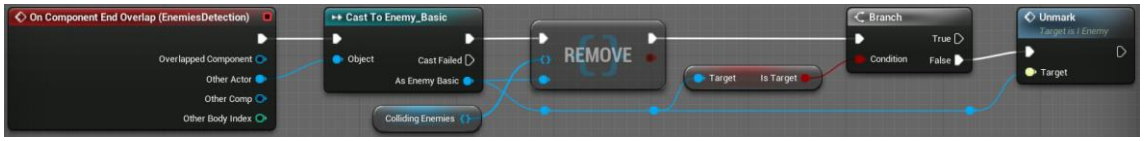

Figura 297. OnComponentEndOverlap (EnemiesDetection)

6.10.3.3 WhoAttack

Aquesta és una funció que es crida a cada *Tick* del *ThirdPersonCharacter*. L'objectiu d'aquesta és determinar quin és l'enemic objectiu en cada moment de forma automàtica.

El primer que es mira és que es compleixin certes condicions per a poder executar el sistema. Es mira que la llista d'enemics propers no estigui buida, que el personatge no estigui atacant, per evitar un canvi d'objectiu en un moment no desitjat, i que no hi hagi un objectiu fixe. Si això es compleix, es posen els valors de les variables *currentEnemy* a buida i *minDistanceEnemy* a 1000000 (un valor molt gran que no es pugui donar en distàncies amb enemics objectiu). Veure Figura 298.

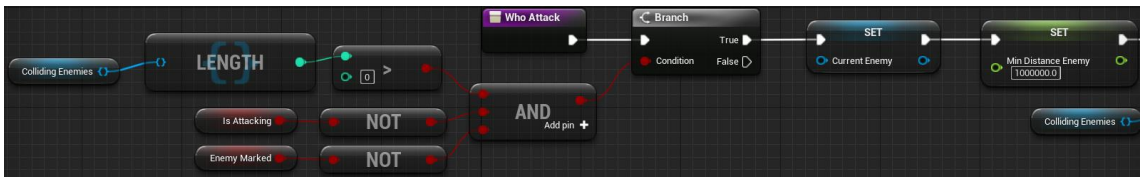

Figura 298. WhoAttack - Part 1

Just després, es fa un bucle per a tots els enemics de la llista, tal com es veu a la Figura 299.

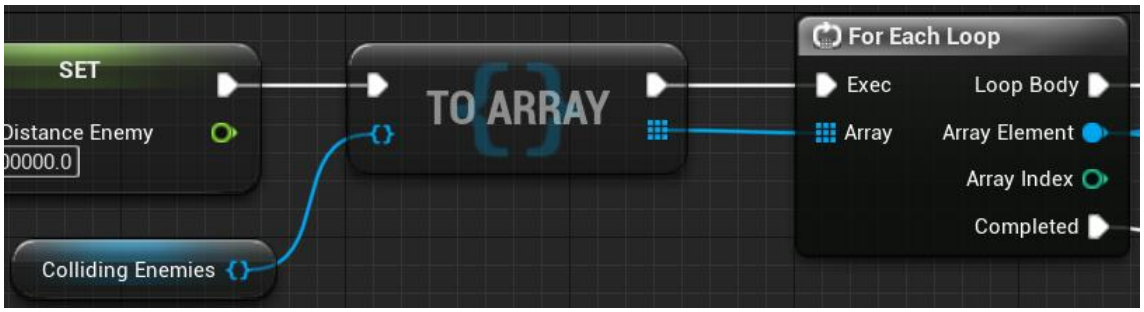

Figura 299. WhoAttack - Part 2

Per a cada enemic, es calcula la distancia entre l'enemic i el jugador, i es mira si és inferior a l'actual *minDistanceEnemy*, i es calcula també si està dins de la pantalla, tal com es veu a la Figura 300. Abans de comprovar les condicions, es desmarca l'enemic amb *UnmarkEnemy* i, després, si es compleixen, es guarden l'enemic a *currentEnemy* i la nova distància mínima a *minDistanceEnemy*, tal com es veu a la Figura 301.

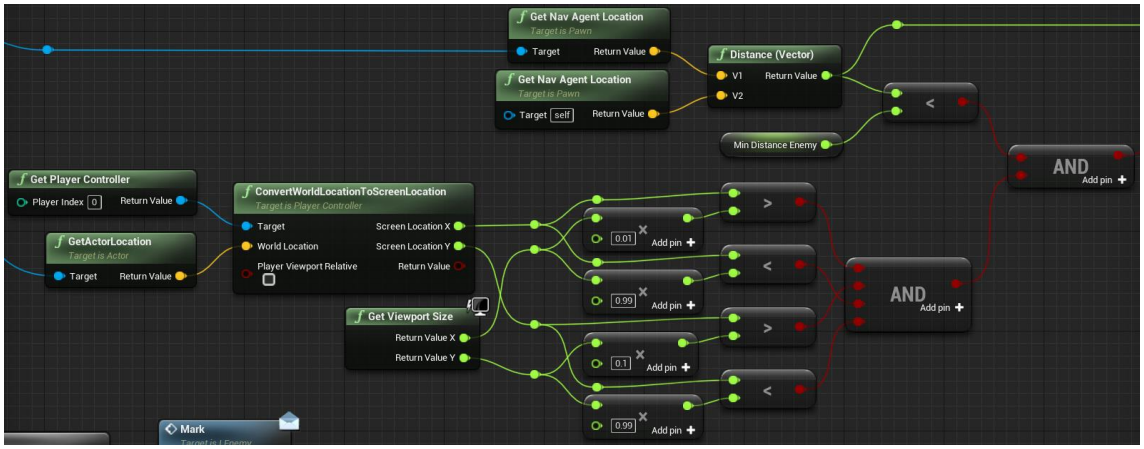

Figura 300. WhoAttack - Part 3

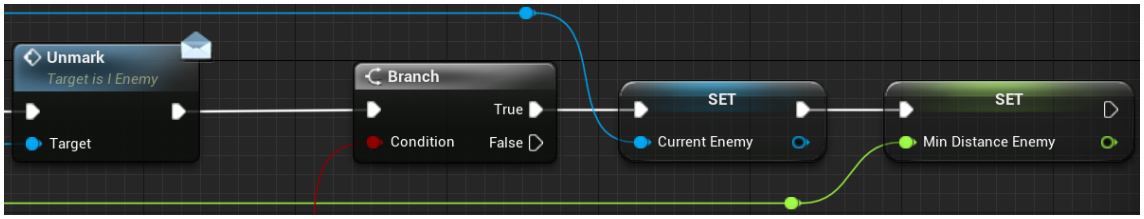

Figura 301. WhoAttack - Part 4

A l'acabar el bucle, si hi ha un *currentEnemy* vàlid, es marca el com a objectiu. Veure Figura 302.

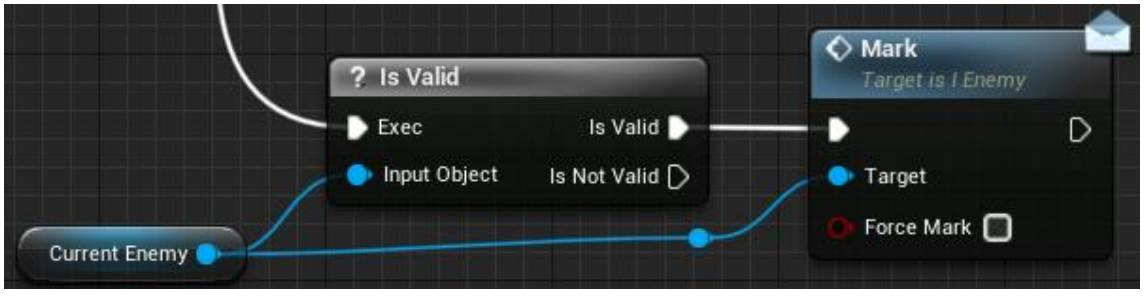

Figura 302. WhoAttack - Part 5

6.10.3.4 Input MarkEnemy

Aquest *Event* es crida quan es prem el botó de marcar/desmarcar un enemic. El què fa l'*Event* és justament això, fer que l'enemic que s'ha determinar objectiu a la funció de l'apartat anterior, sigui fixe o que deixi de ser-ho. El primer que es fa, per tant, es comprovar si es té un enemic fixe o no, tal com es veu a la Figura 303.

Figura 303. MarkEnemy - Part 1

Quan no hi ha cap enemic marcat, primer comprova *currentEnemy* sigui vàlid i li fa un cast a *EnemyBasic*. Veure Figura 304. Després, comprova si realment està a la llista d'enemics propers i, si és així, li posa *isTarget* a cert, el marca amb *forceMark* a cert, i posa *enemyMarked* també a cert. Veure Figura 305.

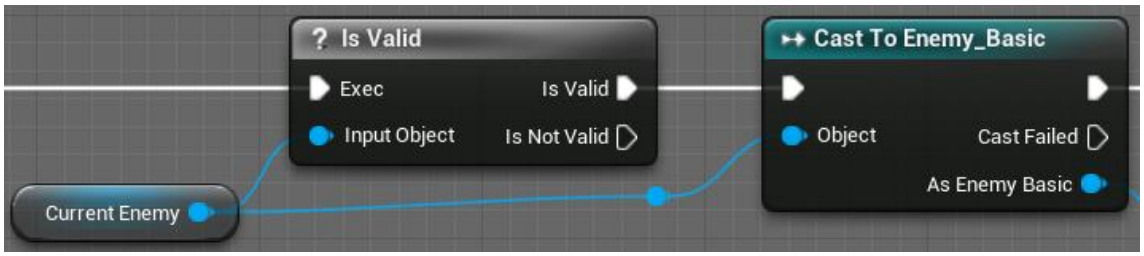

Figura 304. MarkEnemy - Part 2

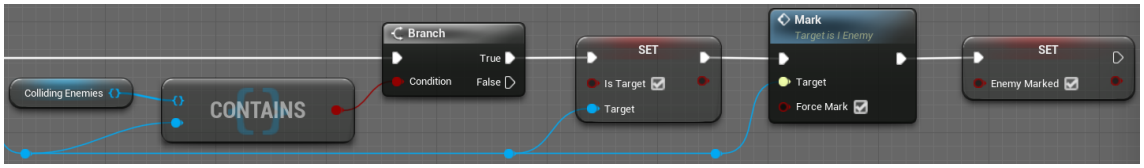

Figura 305. MarkEnemy - Part 3

En cas que sí hi hagi un enemic marcat, fa el procés invers, li posa *isTarget* a fals, el desmarca, i posa *enemyMarked* també a fals. Veure Figura 306.

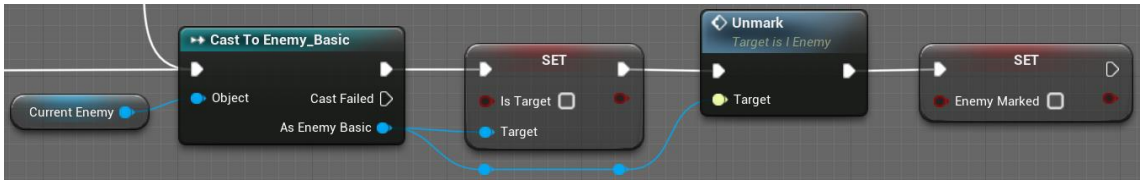

Figura 306. MarkEnemy - Part 4

6.10.3.5 UnmarkEnemy

UnmarkEnemy és un *Event* que executa el mateix codi que a la Figura 306.

6.10.3.6 CameraLookAtEnemyMarked

Aquesta funció té l'objectiu de fer que la càmera apunti cap a l'objectiu fixe, si és que existeix. A més, quan s'escull un objectiu com a fixe, el moviment de la càmera cap a la rotació en què apunti a l'objectiu està suavitzat, també en aquesta funció.

El primer que es fa és mirar si hi ha un objectiu fixe (*enemyMarked* a cert). Si no és així, es posa *startLerpCamera* a cert. Aquesta variable indicarà quan s'ha de fer el suavitzat comentat. En cas que sí hi hagi un objectiu fixe, es calcula la distància fins al jugador. Segons si és o no menor a la mínima distància configurada per un enemic marcat, en surten dues branques. Veure Figura 307.

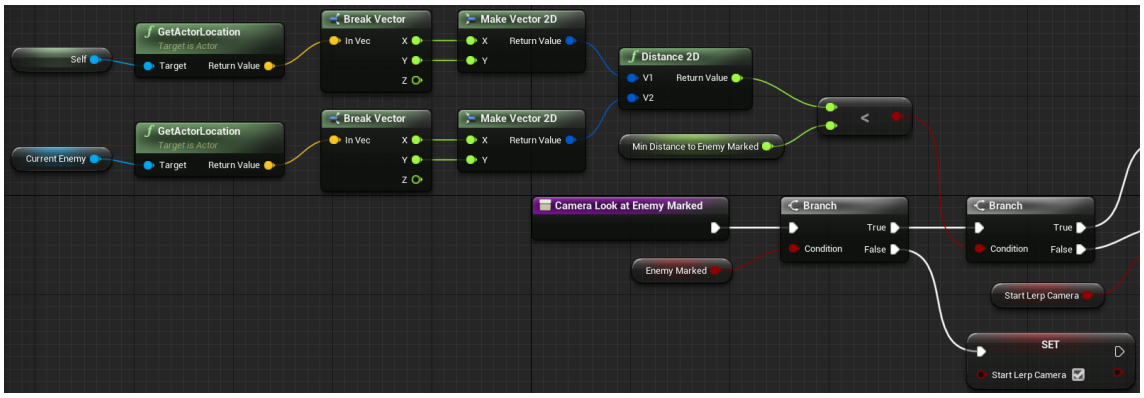

Figura 307. CameraLookAtEnemyMarked - Part 1

En cas que la distància sigui menor, es crida l'*Event* explicat a l'Apartat 6.10.3.5, per desmarcar l'enemic. Això està fet per si el jugador passa per sobre o per sota de l'enemic. En cas contrari, es segueix l'execució. Ara es mira *startLerpCamera*. Si és cert, es posa a fals i *alphaLerpCamera* es posa a 0. Aquesta és la variable que serveix per suavitzar el moviment de la càmera. En cas que *startLerpCamera* sigui fals, no es fa res. Totes dues branques s'uneixen en aquest punt. Veure Figura 308.

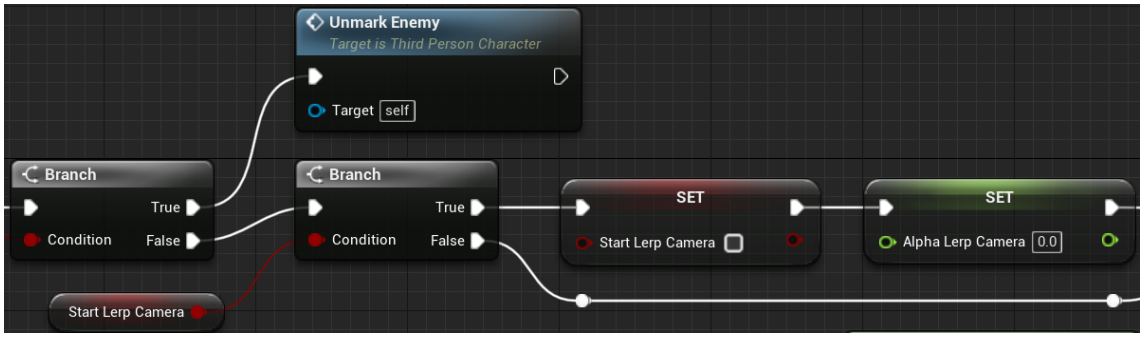

Figura 308. CameraLookAtEnemyMarked - Part 2

Ara es calcula la rotació de la càmera segons on estava inicialment i on hauria d'estar per mirar directament l'enemic. Es fa una combinació d'aquestes dues rotacions amb el paràmetre *alphaLerpCamera*, que indica en quin percentatge s'han de tenir en compte cadascuna de les dues. El càlcul es fa a la Figura 309 i la Figura 310. El valor ja calculat s'assigna a la rotació de la càmera, i s'augmenta el valor d'*alphaLerpCamera* amb el triple de temps que hagi passat des de l'últim *Tick*. Tot això últim es pot veure a la Figura 311.

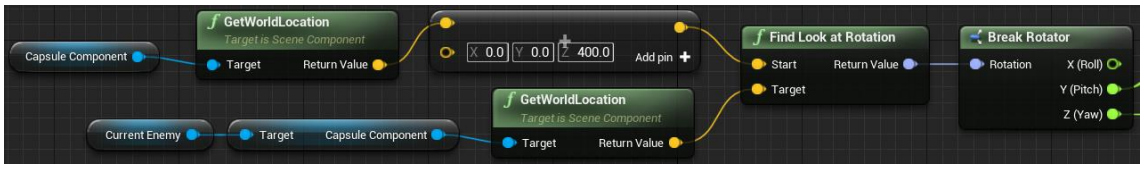

Figura 309. CameraLookAtEnemyMarked - Part 3

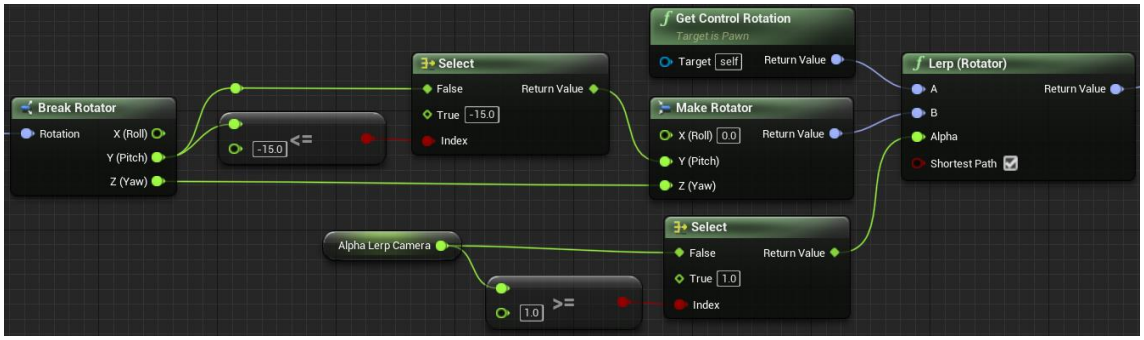

Figura 310. CameraLookAtEnemyMarked - Part 4

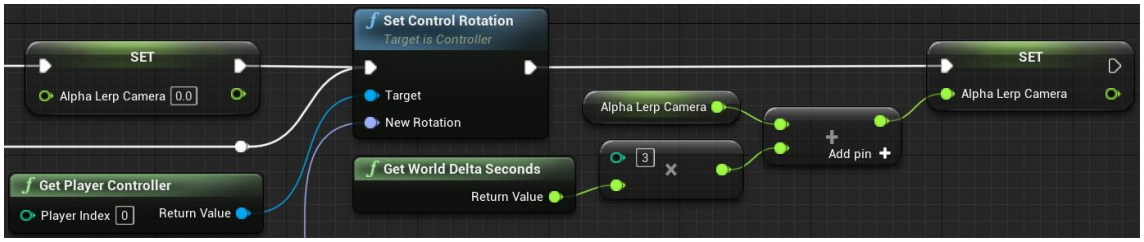

Figura 311. CameraLookAtEnemyMarked - Part 5

6.11 Sistema de Diàlegs

El Sistema de Diàlegs és tot el sistema necessari per a poder tenir converses amb diferents tipus d'elements. Està construït a base d'una sèrie de components que es poden afegir a qualsevol objecte per tenir un diàleg prèviament establert. Aquest sistema s'ha basat en l'estructura que té UE4 per a la intel·ligència artificial. Hi ha diàlegs que només es poden llegir, i hi ha també vegades on es pot escollir una d'entre diverses respostes. Cada resposta portarà a una seqüència del diàleg diferent. Tot i així, el sistema està pensat per a poder repetir tantes vegades com es vulgui un diàleg, i es puguin veure totes les línies de diàleg sense problema.

6.11.1 Visor dels Diàlegs

Aquí es tracten tots els *Blueprints* necessaris per poder veure els diàlegs, tant els diàlegs normals com les respostes. Els diàlegs normals no requereixen de cap objecte específic, però les respostes sí.

6.11.1.1 DialogueReplyObject

Aquest és un objecte que conté una possible resposta. No és més que això, un objecte amb una variable de tipus *Text* que conté el text a mostrar. A més, té un *Event Dispatcher* (o disparador), per a quan es fa clic a sobre aquest.

6.11.1.2 DialogueEntryWidget

Aquest és el *Widget* que mostrarà una possible resposta. No és més que un botó amb un text, tal com es veu a la Figura 312.

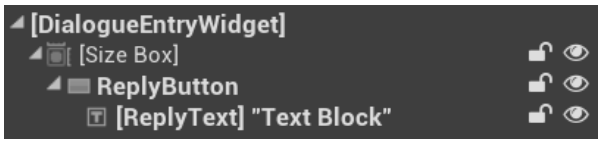

Figura 312. DialogueEntryWidget - Estructura

Per tal d'indicar si s'està a sobre del botó o no, s'implementen els mètodes *OnHovered* (Figura 313) i *OnUnhovered* (Figura 314) del botó, que simplement canvien el color del text amb els colors configurats.

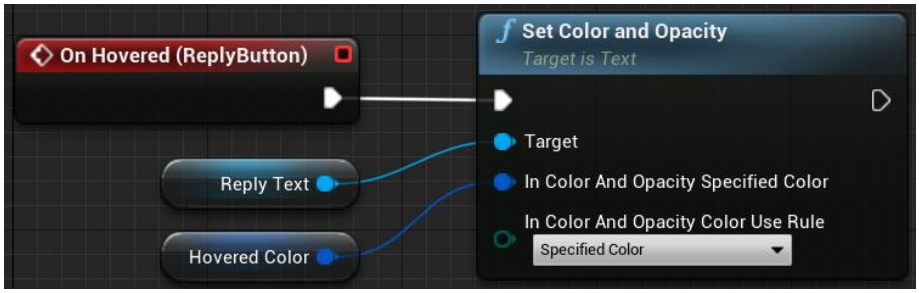

Figura 313. DialogueEntryWidget - OnHovered

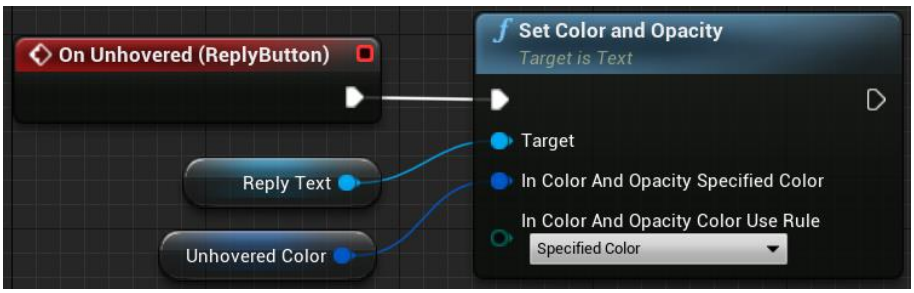

Figura 314. DialogueEntryWidget - OnUnhovered

A més, aquest *Widget*, requereix de l'objecte *DialogueReplyObject*. Per a tenir-lo, s'ha d'afegir la interfície *User Object Entry List*, pròpia d'UE4, i implementar el mètode *EventOnListItemObjectSet*. La implementació obté un objecte de tipus *DialogueReplyObject*, i el guarda a una variable anomenada *dialogueReplyObject*. Després, canvia el text per defecte del *DialogueEntryWidget* pel del *DialogueReplyObject*. Veure Figura 315.

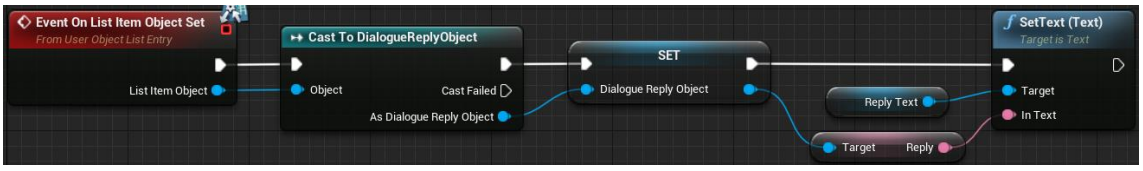

Figura 315. DialogueEntryWidget - EventOnListItemObjectSet

Per acabar, s'implementa la funció *OnClicked*, que s'executa al prémer el botó. Aquesta, únicament, crida l'*OnClicked* del *DialogueReplyObject*. Veure Figura 316.

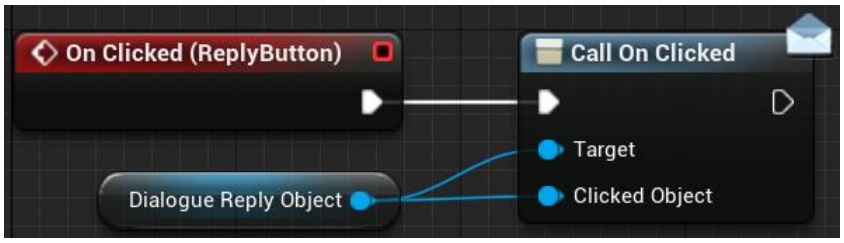

Figura 316. DialogueEntryWidget – OnClicked

6.11.1.3 DialogueWidget

DialogueWidget és el *Widget* complet dels diàlegs. Conté tant els diàlegs normals com les respostes, fent ús del *DialogueEntryWidget*.

6.11.1.3.1 Apartat Visual

L'estructura d'aquest component és senzilla. Té tres blocs de contingut: el primer, és el bloc on es veurà el nom de qui està parlant, el segon és on es podran veure els diàlegs, i el tercer, on es posen les diverses respostes, on cada resposta és un *DialogueEntryWidget*. Els blocs 2 i 3 es superposen, i només hi ha un activat a la vegada. L'estat de diàleg es pot veure a la Figura 317 i l'estat de resposta a la Figura 318.

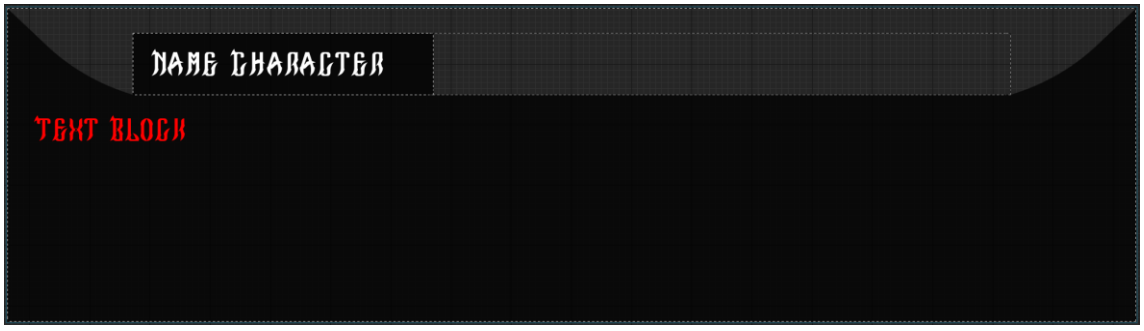

Figura 317. DialogueWidget - Apartat Visual – Diàleg

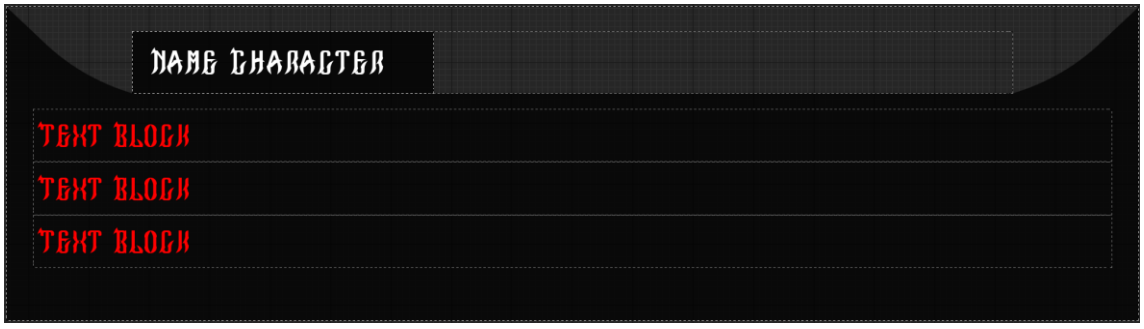

Figura 318. DialogueWidget - Apartat Visual - Resposta

6.11.1.3.2 Implementació de la Lògica

Per implementar la lògica, primer cal tenir algunes coses. Per saber quin dels dos blocs de l'apartat visual cal mostrar, es necessita tenir una variable de l'estat anomenada *dialogueState*. Aquesta es determina a partir d'un nou *Enumeration* anomenat *EDialogueState*, i té els estats d'*Speak* pels diàlegs normals, i *Reply* per les respostes. A més, també es necessita configurar el nom del personatge per defecte amb la variable *characterName*. Per acabar, es creen 3 *Event Dispatcher*: *OnSpeakFinished*, *OnReplyFinished* i *OnExit*. Veure Figura 319.

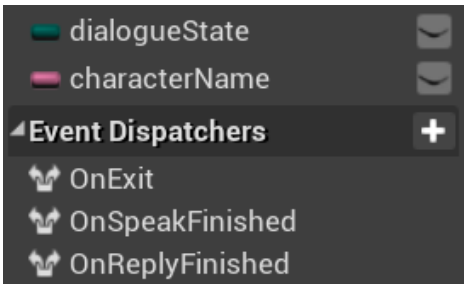

Figura 319. DialogueWidget - Variables i Event Dispatcher's

A més de les variables, hi ha les funcions *SetDialogueState* i *OnMouseButtonDown*, i els *Events SetName*, *Speak*, *Reply* i *Exit*.

6.11.1.3.2.1SetDialogueState

Canvia la variable d'estat segons l'entrada i actualitza els blocs de diàleg i resposta. Si l'estat entrat és *Speak*, mostra el bloc de diàleg i amaga el de resposta i, si és *Reply*, a l'inrevés. Veure Figura 320.

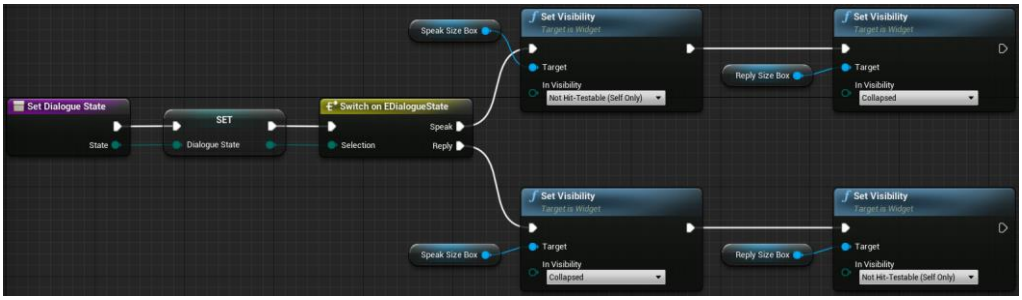

Figura 320. SetDialogueState

6.11.1.3.2.2OnMouseButtonDown

OnMouseButtonDown és una funció sobreescrita per a implementar la lògica quan es toca el *Widget*. El què es fa és, primer de tot, comprovar la variable d'estat. Si s'està en mode diàleg, i el botó premut és el clic esquerre, es crida l'*Event Dispatcher OnSpeakFinished*. Veure Figura 321.

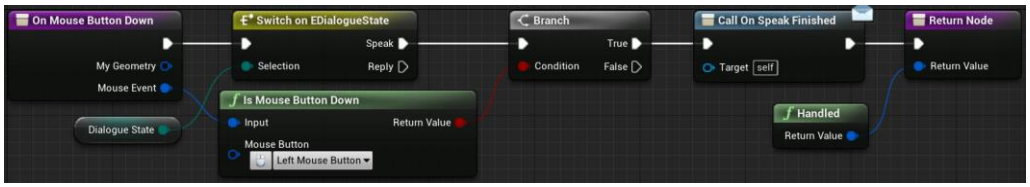

Figura 321. OnMouseButtonDown

6.11.1.3.2.3SetName

Aquest *Event* serveix per actualitzar el nom en els diàlegs i canviar l'estat a *Speak*, tal i com es veu a la Figura 322.

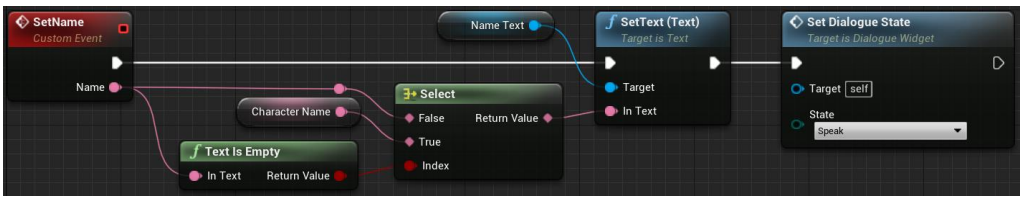

Figura 322. SetName

6.11.1.3.2.4Speak

Aquest *Event* serveix per actualitzar el text del diàleg. Veure Figura 323.

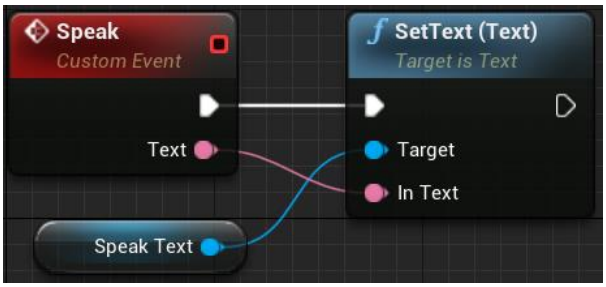

Figura 323. Speak

6.11.1.3.2.5Reply

Aquest *Event* serveix per actualitzar el text de les respostes i fer assignacions a les funcions corresponents quan es prem una. Primer de tot, es buida la llista *repliesListView*, que conté els *DialogueReplyObject* que generen els seus *DialogueEntryWidget* automàticament amb el codi explicat a l'Apartat 6.11.1.2. Després, per cada resposta rebuda, es crea un *DialogueReplyObject* i se li assigna el text corresponent. Veure Figura 324.

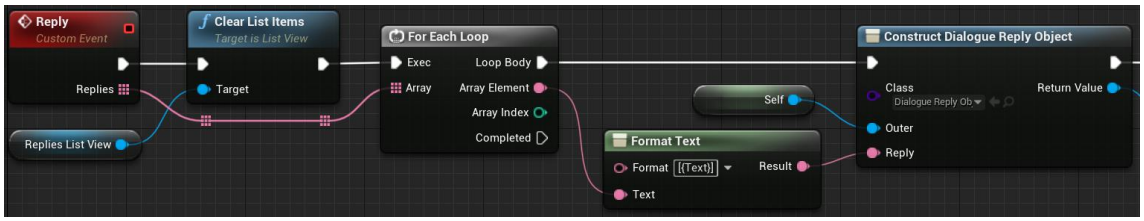

Figura 324. Reply - Part 1

Després, s'afegeix a *repliesListView*, el qual genera els *DialogueEntryWidget*. A més, a cadascun es fa un *Binding* de l'*Event Dispatcher OnClicked*, que saltarà quan es premi sobre l'objecte. A més, es posa el nom del personatge configurat i es canvia l'estat a *Reply*. També s'implementa l'*Event* assignat a l'*OnClicked*. Aquest obté l'índex de la resposta premuda i crida *OnReplyFinished* passant l'índex. Veure Figura 325.

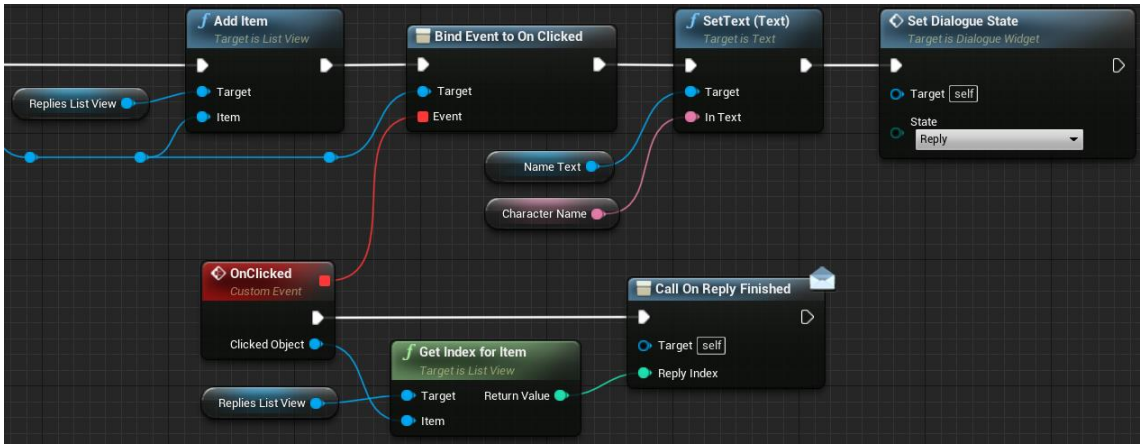

Figura 325. Reply - Part 2

6.11.1.3.2.6Exit

L'*Event* d'*Exit* simplement crida l'*OnExit*. Veure Figura 326.

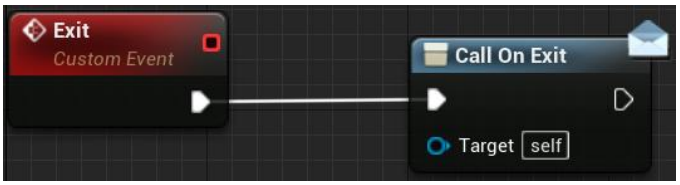

Figura 326. Exit

6.11.2 Intel·ligència Artificial dels Diàlegs

Ara ja es pot crear el sistema d'intel·ligència artificial que farà que el sistema de diàlegs funcioni correctament, fent ús de tots els components explicats. Per implementar-ho, es requereixen d'un controlador, un *Blackboard*, les diverses tasques requerides i l'arbre de diàlegs que determinarà quins són els diàlegs, quines són les respostes, i què es mostra abans i després.

6.11.2.1 DialogueIAController

Aquest és un *Blueprint* que hereta de *AIController*, un *Blueprint* propi d'UE4. Aquest no implementa res nou, simplement és necessari per a executar la intel·ligència artificial.

6.11.2.2 DialogueBlackboard

El *Blackboard* és un component d'UE4 que serveix per a generar les claus que s'utilitzaran als arbres creats. *DialogueBalckboard* té tres claus: *DialogueWidget* conté un *DialogueWidget*, *ReplyIndex* conté l'índex de la resposta escollida, i *SelfActor* és una referència a sí mateix, que està per defecte. Veure Figura 327.

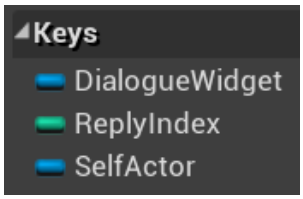

Figura 327. DialogueBalckboard

6.11.2.3 Tasques

Les tasques són les diverses funcions que s'executen en un arbre. Poden rebre paràmetres, que corresponen a les claus explicades a l'apartat anterior. Cada tasca ha de tenir un *Event* inicial (anomenat *ReceiveExecuteAI*), el qual es crida quan s'executa la tasca.

6.11.2.3.1 Speak

Speak és la tasca que s'encarrega de tota la lògica dels diàlegs normals.

6.11.2.3.1.1Event ReceiveExecuteAI

A l'inici d'aquest *Event* es reinicien les diverses variables. *maxLetters* rep el valor de la mida del text entrat, *letters* es posa a 0, i *finishWriting*, a fals. Veure Figura 328.

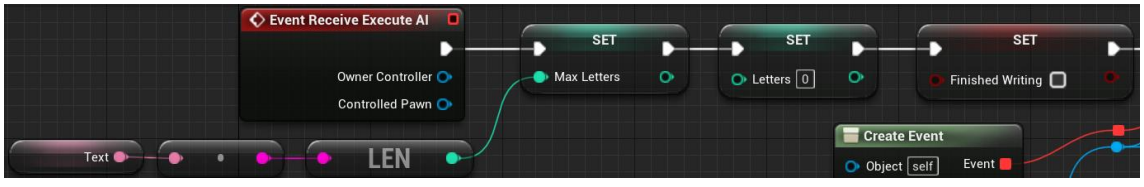

Figura 328. Speak - Event ReceiveExecuteAI - Part 1

Just després, es crea un *Event OnSpeakFinished_Event* que es crida quan el *DialogueWidget* crida *OnSpeakFinished*. Al final, es crida l'*Event SetName* del *DialogueWidget* amb el nom del personatge que parla. Veure Figura 329.

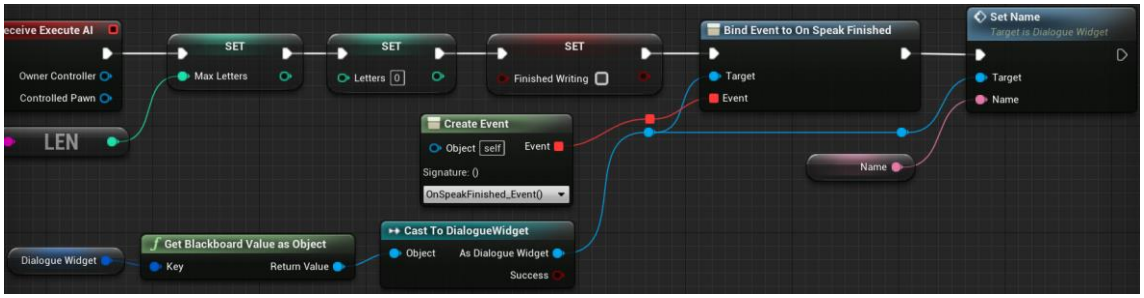

Figura 329. Speak - Event ReceiveExecuteAI - Part 2

6.11.2.3.1.2OnSpeakFinished_Event

Aquest és l'*Event* que s'executa quan el *DialogueWidget* crida *OnSpeakFinished*. El primer que fa és comprovar la variable *finishedWriting*. En cas que estigui a fals, la posa a cert i crida *Speak* amb el text sencer. En cas contrari, elimina la relació amb *OnSpeakFinished* i finalitza l'execució de la tasca. Veure Figura 330.

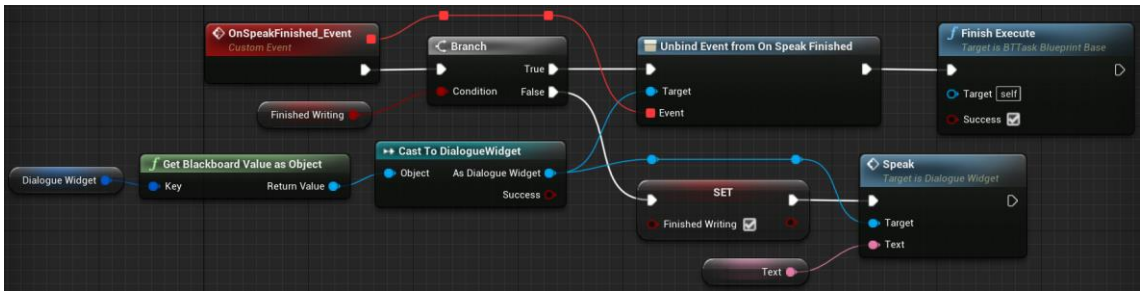

Figura 330. Speak - OnSpeakFinished_Event

6.11.2.3.1.3Tick

Aquesta funció s'executa cada vegada que el rellotge fa Tick. El primer que fa és comprovar si el valor de "letters" és igual o menor a *finishedLetters*, i si *finishedWriting* és fals. Es cas de que aquesta condició sigui falsa, es posa *finishedWriting* a fals. Veure Figura 331.

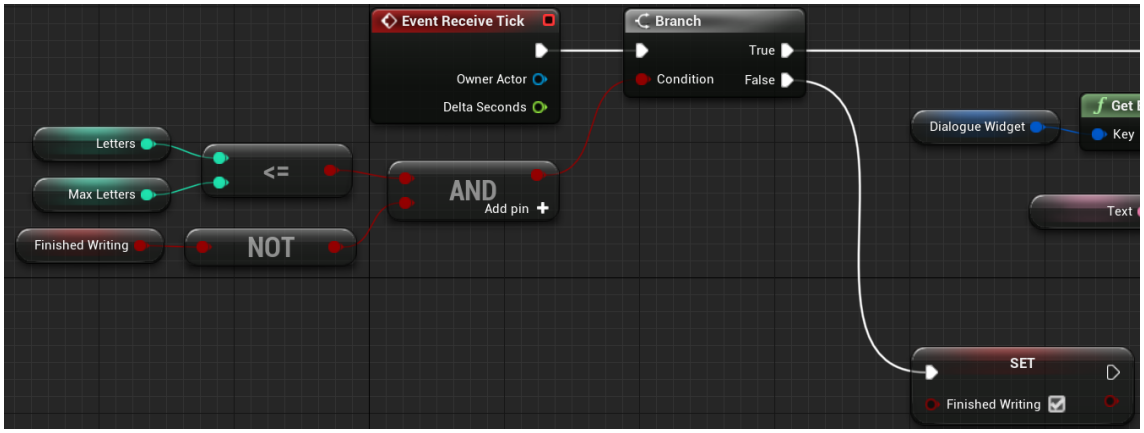

Figura 331. Speak - Tick - Part 1

En cas contrari, es crida *Speak* amb una part del text sencer, indicada la mida amb el *letters*. Al final, es suma 1 a *letters*. Veure Figura 332.

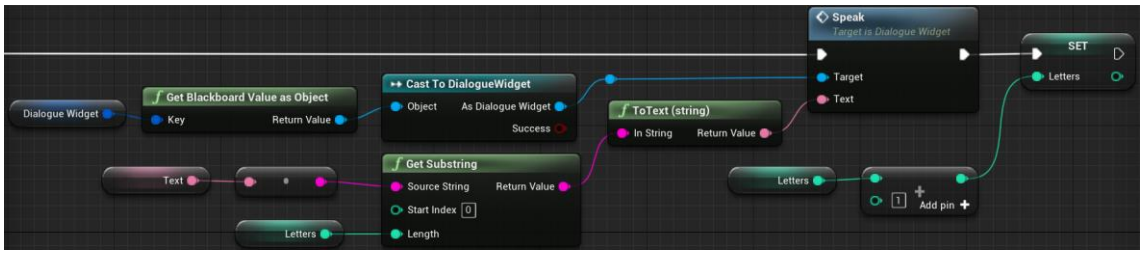

Figura 332. Speak - Tick - Part 2

6.11.2.3.2 Reply

Reply és la tasca que s'encarrega de tota la lògica de les respostes.

6.11.2.3.2.1Event ReceiveExecuteAI

En aquest *Event*, tal i com es veu a la Figura 333, simplement, es crea un *Event OnReplyFinished*_Event que es crida quan el *DialogueWidget* crida *OnReplyFinished*, i es crida el *Reply* del *DialogueWidget* amb totes les respostes.

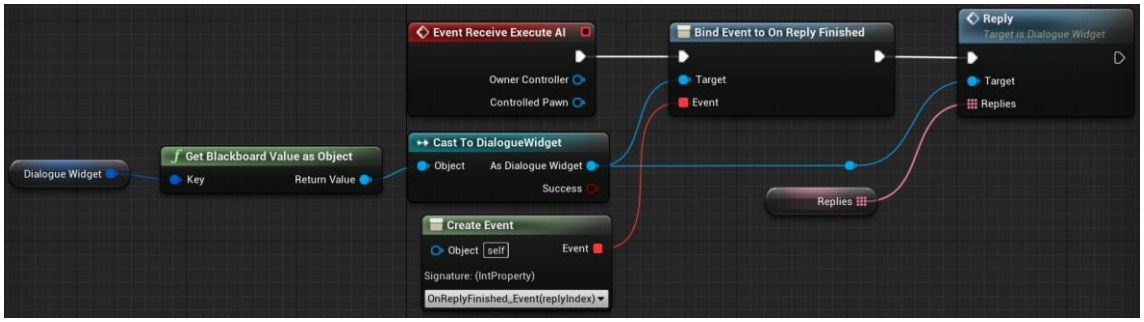

Figura 333. Reply - Event ReceiveExecuteAI

6.11.2.3.2.2OnReplyFinished_Event

En aquest *Event*, s'elimina la relació amb *OnReplyFinished*, s'assigna a *replyIndex(out)* el valor entrat, i es finalitza l'execució de la tasca. Veure Figura 334.

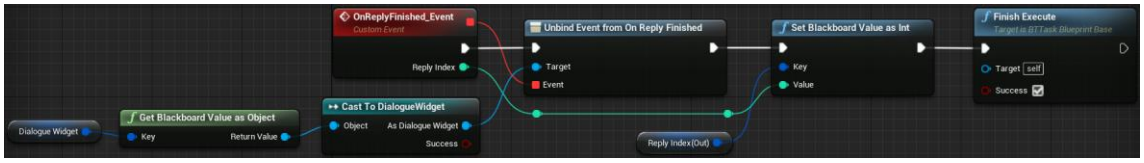

Figura 334. Reply - OnReplyFinished_Event

6.11.2.3.3 Exit

Exit és la tasca que s'encarrega d'acabar l'execució del sistema de diàlegs. Només té l'*Event ReceiveExecuteAI*, i simplement crida l'*Event Exit* del *DialogueWidget* i finalitza l'execució de la tasca. Veure Figura 335.

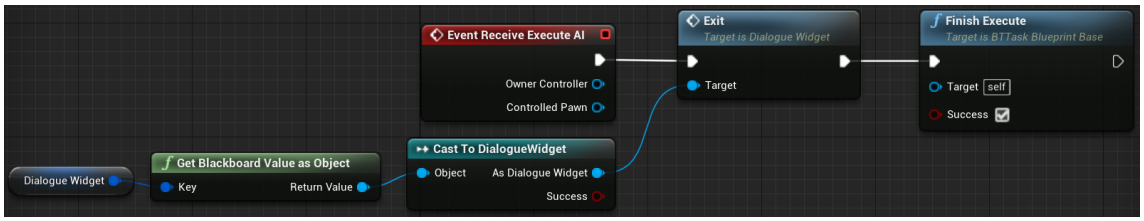

Figura 335. Exit - Event ReceiveExecuteAI

6.11.2.4 Arbres de Diàleg

Els arbres de diàleg són *Blueprints* del tipus *Behavior Tree*, que s'utilitzen per fer crides a les tasques i escriure els diàlegs i respostes que el jugador haurà de poder veure. Cada arbre té una implementació pròpia, però tots requereixen de les mateixes tasques.

A les figures Figura 336 i Figura 337 es pot veure un exemple molt simple del funcionament de les tasques i implementació d'un arbre. En aquest cas, s'executaran els 3 primers *Speak* de la Figura 336, posant com a diàleg el text escrit al paràmetre "Text" i el nom indicat a *Name* (o el configurat, en cas d'estar buit). Després, s'executa la tasca *Reply*, la qual guardarà un valor a *replyIndex*. Segons aquest valor, es decidirà la branca a seguir amb el selector. En el cas que *replyIndex* sigui igual a 2, s'executarà l'*Speak* i després l'*Exit*, acabant per complet el diàleg. En cas contrari, s'executarà l'*Speak* corresponent i l'arbre tornarà a començar l'execució des de l'inici.

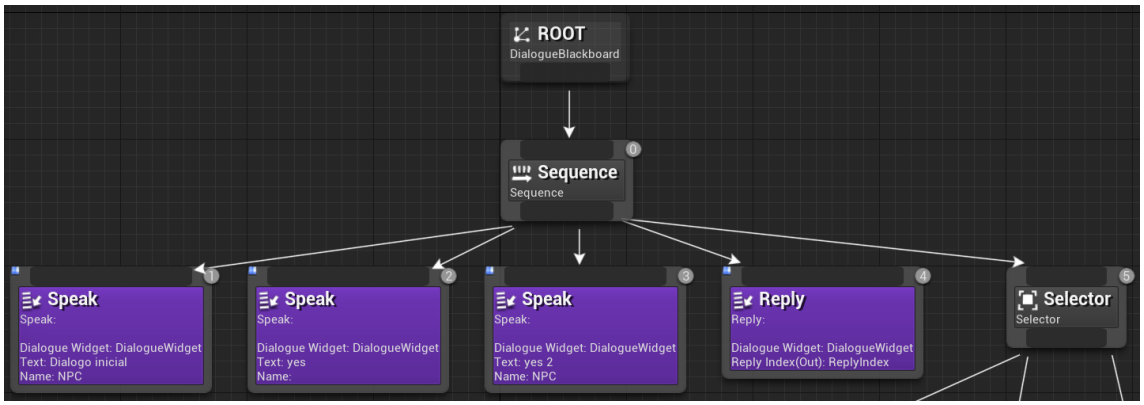

Figura 336. Arbre de Diàleg - Part 1

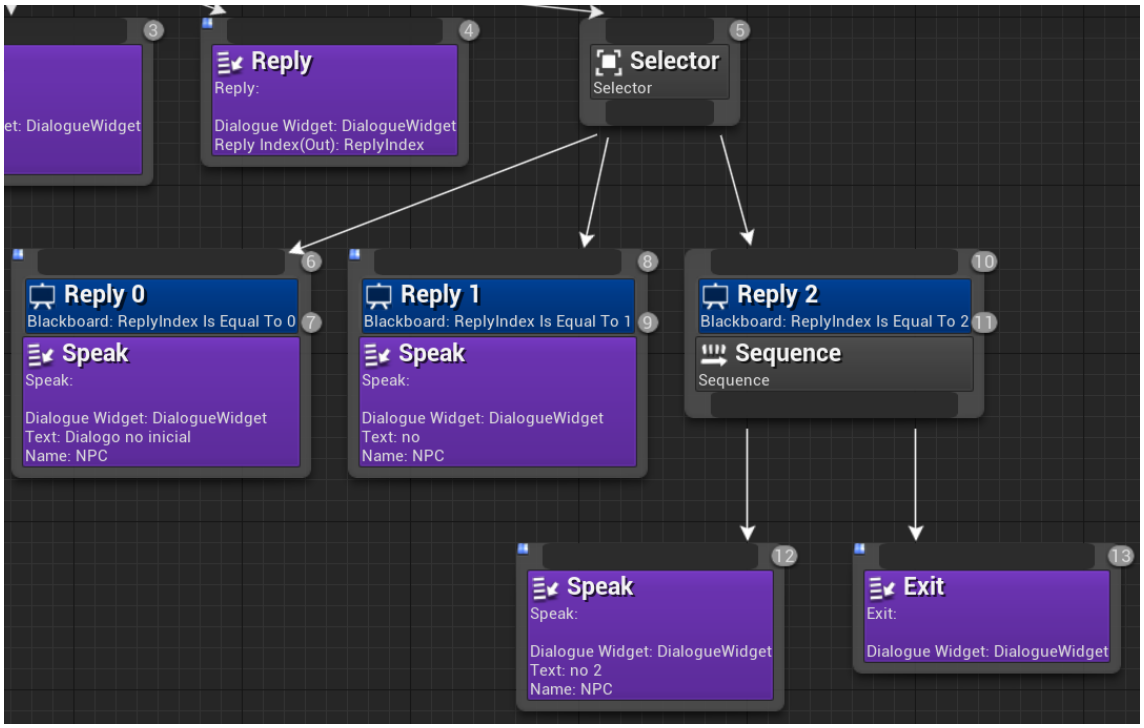

Figura 337. Arbre de Diàleg - Part 2

6.11.3 Component del Diàleg

El Component del Diàleg és el què conté tota la lògica necessària per unificar totes les parts creades en els anteriors passos. Aquest component, heretat d'*Actor Component*, un *Blueprint* propi d'UE4, es pot afegir a qualsevol objecte, de forma que és completament genèric i extrapolable. Per crear el component, primer hi s'ha fet una interfície, i després el component en sí, el qual implementa la interfície.

6.11.3.1 I_Dialogue

Interfície que conté la funció per iniciar un diàleg, anomenada *StartDialogue*, que rep com a paràmetre un *ThirdPersonCharacter.*

6.11.3.2 DialogueComponent

Aquest és el component comentat, el qual implementarà tota la lògica necessària per unificar i executar el sistema de diàlegs per complet. Implementa la interfície *I_Dialogue*.

6.11.3.2.1 Event BeginPlay

Aquest *Event* es crida a l'executar el joc. El què fa és crear un *DialogueIAController* i assignar-ho a una variable local *IAController*. Veure Figura 338.

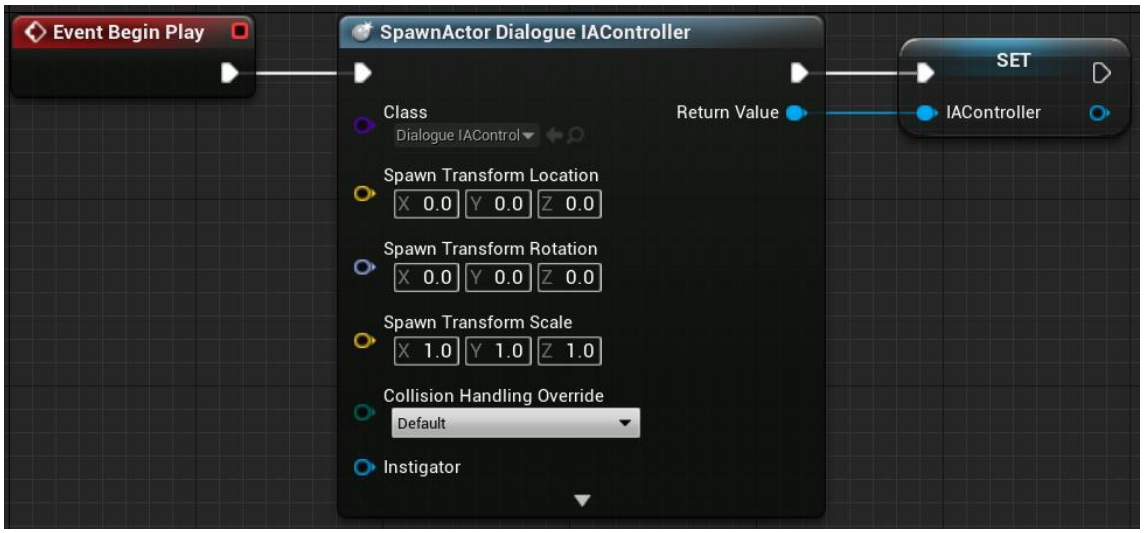

Figura 338. DialogueComponent - BeginPlay

6.11.3.2.2 StartDialogue

StartDialogue és la funció de la interfície *I_Dialogue*. El primer que fa és crear un *DialogueWidget* i assignar-lo a una variable local. Veure Figura 339.

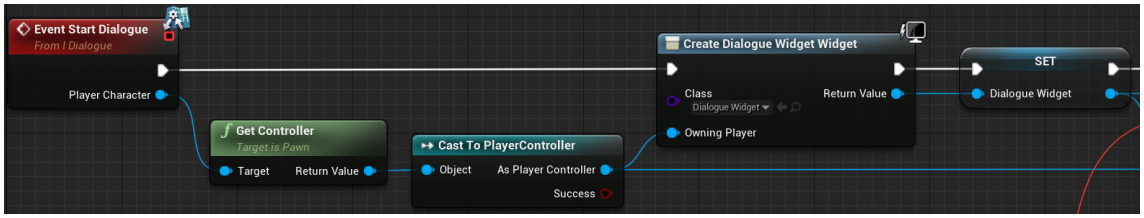

Figura 339. DialogueComponent - StartDialogue - Part 1

Just després, crea una relació entre l'*OnExit* del *DialogueWidget* i l'*Event OnExit_Event*, fent que la crida d'*OnExit* executi l'altre. A més, s'afegeix el *DialogueWidget* a la pantalla, canvia el mode del jugador per poder interactuar amb els botons amb el ratolí, i mostra el ratolí, per tal de poder veure on està situat. Veure Figura 340.

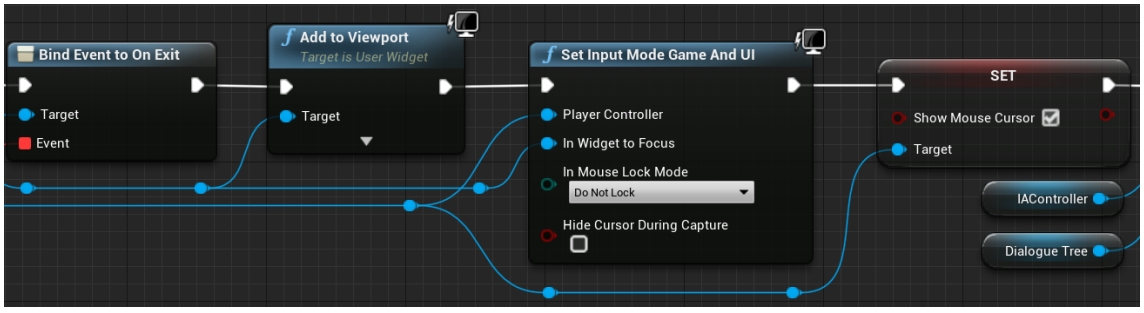

Figura 340. DialogueComponent - StartDialogue - Part 2

Per acabar, s'executa l'arbre escollit (es configura en una variable pública anomenada *dialogueTree*), s'assigna el *DialogueBlackboard* i, a aquest mateix, se li assigna el *DialogueWidget* a la clau pertinent. Veure Figura 341.

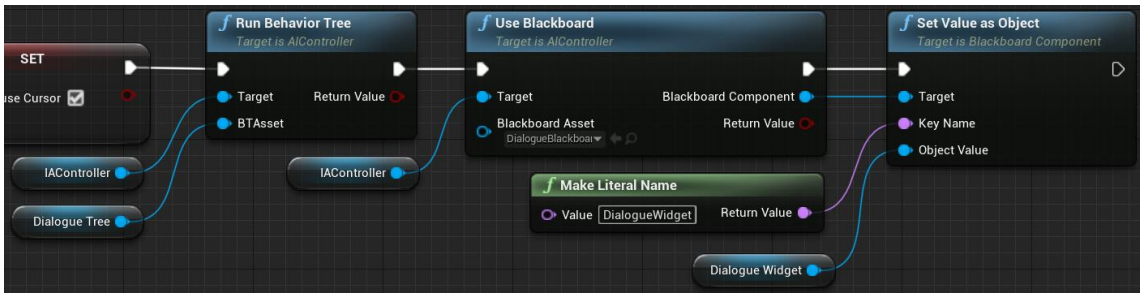

Figura 341. DialogueComponent - StartDialogue - Part 3

6.11.3.2.3 OnExit_Event

Aquest *Event* s'executa quan es crida l'*OnExit* del *DialogueWidget*. Tal com es veu a la Figura 342, el primer que fa és posar el mode a només joc i amagar el ratolí.

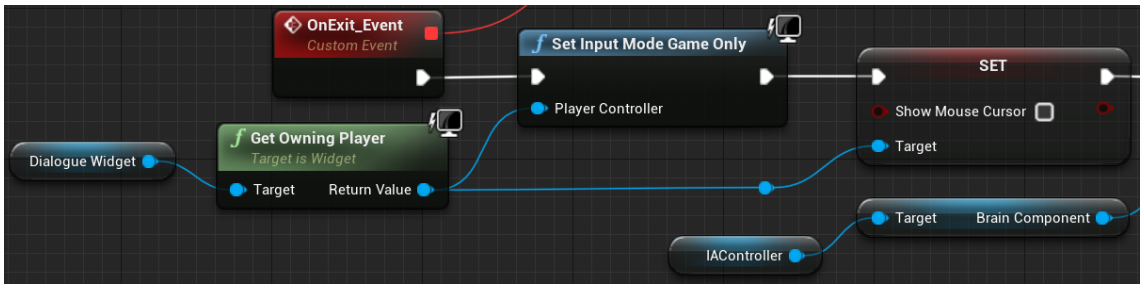

Figura 342. DialogueComponent - OnExit_Event - Part 1

Just després, atura la intel·ligència artificial, elimina el *DialogueWidget* de la pantalla i, si el pare d'aquest component és un *NPC_Basic* (explicat a l'Apartat 6.12), crida la funció *EndDialogue*. Veure Figura 343.

Figura 343. DialogueComponent - OnExit_Event - Part 2

6.12 Sistema d'Interacció amb NPC's

El Sistema d'Interacció amb NPC's és un sistema fet per a que el jugador pugui interactuar i iniciar diàlegs amb els diferents personatges de forma senzilla i entenedora. A més, és un sistema pensat de forma extrapolable a tots els NPC's. Està fet per poder agafar el personatge base que s'implementa en aquest apartat, i heretar d'ell tantes classes diferents de personatge com calgui.

En el sistema hi ha dues parts importants. La primera és la implementació de l'NPC, que implica tota la part d'interacció, i inici i final dels diàlegs. La segona és la part que decideix amb qui interacciona el jugador en cada moment, si és que decidís interactuar. Per la primera part, es requereixen una interfície i el *Blueprint* que farà pròpiament d'NPC (a més d'algun component auxiliar). Per la segona part, només cal afegir lògica al *ThirdPersonCharacter*.

6.12.1 I_NPC

Interfície que indica les funcions que requereix el sistema d'interacció amb NPC's. Les funcions són les que es veuen a la Figura 344: *Interact* (que rep un paràmetre pel *ThirdPersonCharacter*), *StartDialogue*, *EndDialogue*, *ActivateInteraction* i *DeactivateInteraction*.

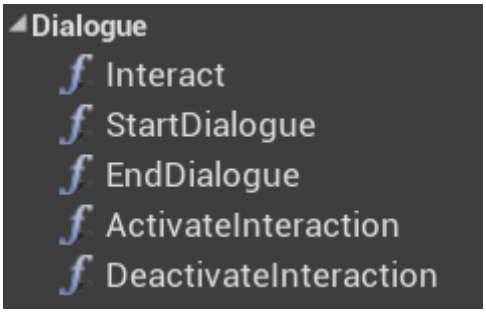

Figura 344. I_NPC

6.12.2 Dialogue_Nameplate

Aquest és un *Widget* auxiliar que s'utilitza per indicar al jugador que pot iniciar un diàleg i com fer-ho. Està compost simplement per un text. Veure Figura 345.

Figura 345. Dialogue_Nameplate

6.12.3 BlackScreenPop

Aquest component és un *Widget* auxiliar per a generar un efecte de *FadeIn* i *FadeOut* (de imatge a pantalla negra, i viceversa). Està format per una imatge negra que ocupa tota la pantalla. A aquesta, se li afegeix una animació que varia la opacitat durant el temps, començant a 0, posant-se a 1 als 0.2 segons, i tornant a 0 als 0.4. Aquesta animació es crida en la construcció de l'objecte. Veure Figura 346.

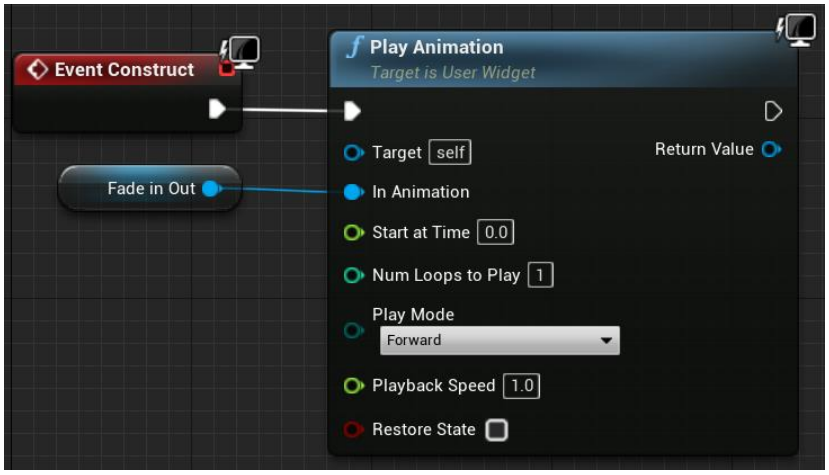

Figura 346. BlackScreenPop

6.12.4 NPC_Basic

NPC_Basic és el *Blueprint* que implementa tot el sistema d'interacció. Aquí s'utilitzen els components explicats a l'Apartat 6.11 Sistema de Diàlegs i els propis d'aquest apartat. Aquest *Blueprint* hereta del tipus *Character* d'UE4, que es tracta d'un personatge que es pot moure. Té una estructura concreta explicada a continuació, i implementa les funcions indicades a I_NPC.

6.12.4.1 Estructura

L'estructura inicial és, òbviament la del *Character*, que conté, el *Capsule Component*, l'*Arrow Component*, el *Mesh* i el *Character Movement* (el component que li dona moviment al personatge). Les parts afegides són: el *Cone*, que indica a quina posició anirà col·locat el jugador quan estigui en un diàleg, el *NamePlate*, que es tracta del *Widget Dialogue_Nameplate*, l'*InteractionCollider*, el col·lisionador que indicarà al jugador si pot interactuar amb l'NPC, una càmera que s'utilitza durant els diàlegs, i el *DialogueComponent* explicat a l'Apartat 6.11.3. Tot això es pot veure a la Figura 347. L'aspecte resultant és el de la Figura 348

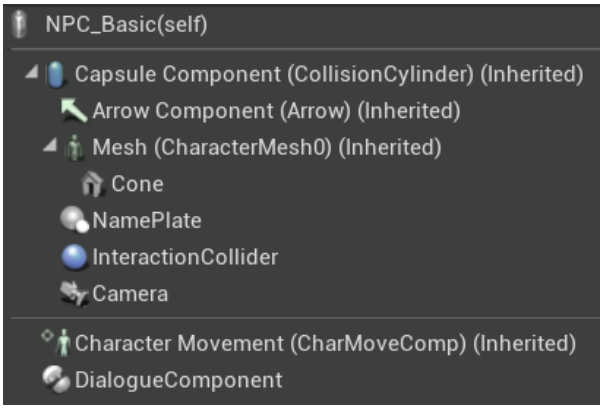

Figura 347. NPC_Basic - Estructura

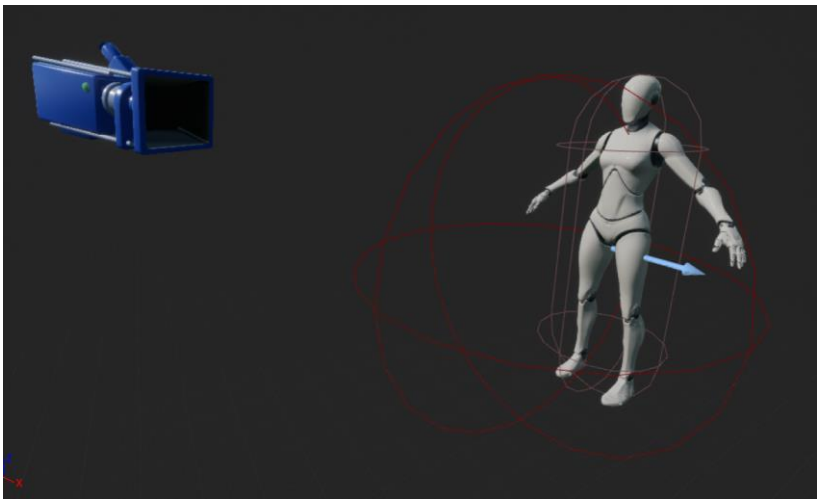

Figura 348. NPC_Basic - Aspecte

6.12.4.2 Lògica

Quan ja es té l'estructura feta, cal implementar la lògica.

6.12.4.2.1 Construction Script

Aquesta és una funció que s'executa al crear-se l'objecte. El què fa és situar la càmera en la posició i rotació configurades a la variable *cameraTransform*, i el *Cone* a la posició configurada a la variable *playerLocation*. Veure Figura 349.

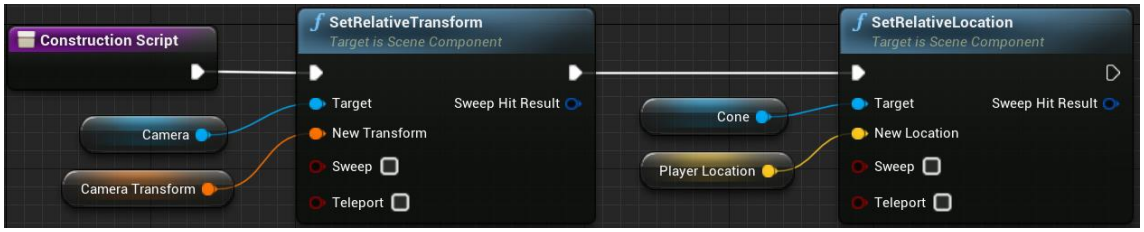

Figura 349. NPC_Basic - Construction Script

6.12.4.2.2 ActivateInteraction

Implementació de la funció *ActivateInteraction* de la interfície I_NPC. El què fa és, si *isInteracting* és fals, posa el *NamePlate* a visible i posa *isPlayerNear* a cert. Veure Figura 350.

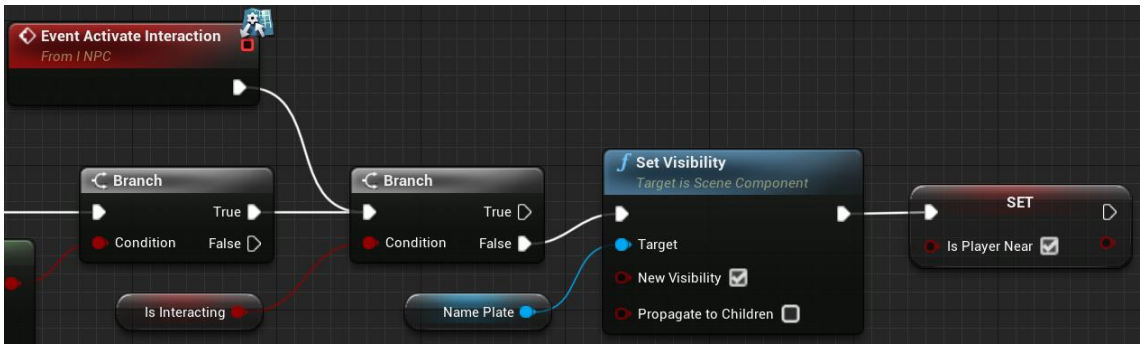

Figura 350. NPC_Basic - ActivateInteraction

6.12.4.2.3 OnComponentBeginOverlap (InteractionCollider)

Event que s'executa si algun objecte comença a interseccionar amb *InteractionCollider*. Comprova que l'objecte sigui el *ThirdPersonCharacter*. Si ho és, executa el mateix codi que *ActivateInteraction*. Veure Figura 351.

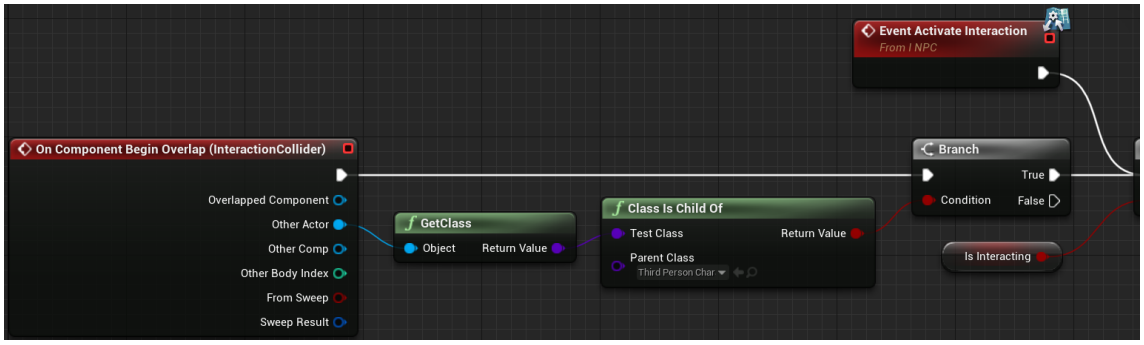

Figura 351. NPC_Basic - OnComponentBeginOverlap (InteractionCollider)

6.12.4.2.4 DeactivateInteraction

Implementació de la funció *DeactivateInteraction* de la interfície I_NPC. El què fa és desactivat la visibilitat del *NamePlate* i es posa *isPlayerNear* a fals. Veure Figura 352.

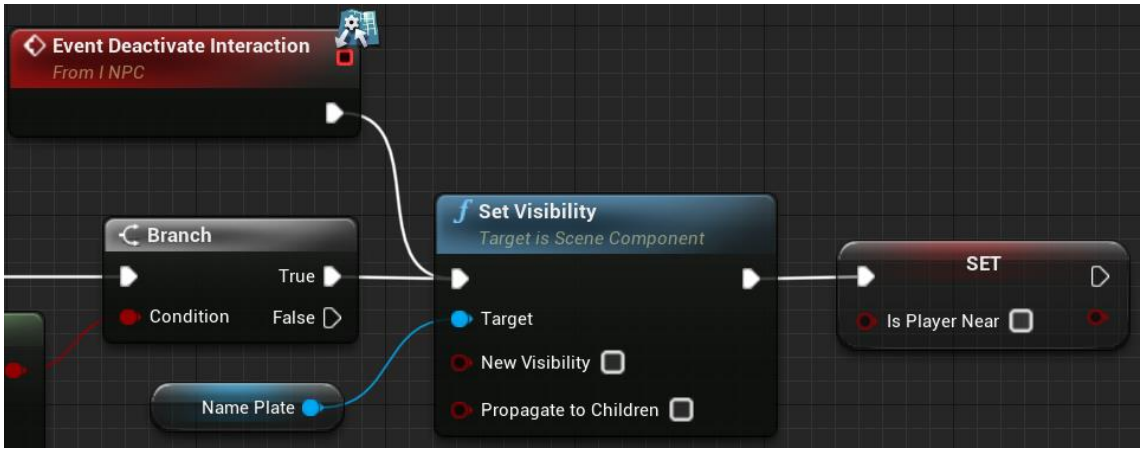

Figura 352. NPC_Basic - DeactivateInteraction

6.12.4.2.5 OnComponentEndOverlap (InteractionCollider)

Event que s'executa si algun objecte deixa d'interseccionar amb *InteractionCollider*. Comprova que l'objecte sigui el *ThirdPersonCharacter*. Si ho és, executa el mateix codi que *DeactivateInteraction*. Veure Figura 353.

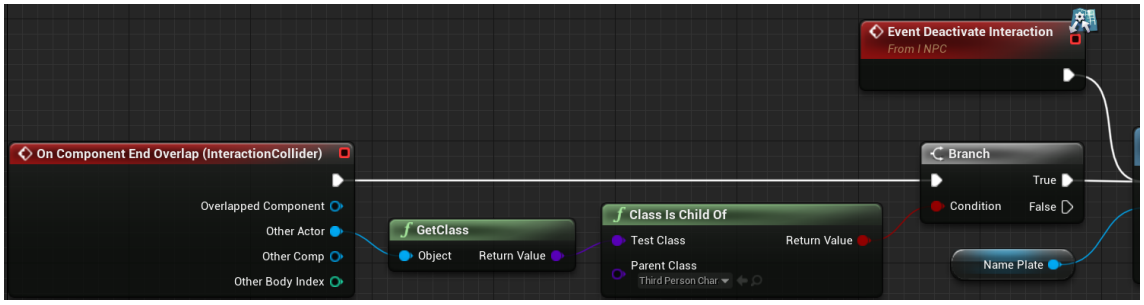

Figura 353. NPC_Basic - OnComponentEndOverlap (InteractionCollider)

6.12.4.2.6 Interact

Implementació de la funció *Interact* de la interfície I_NPC. Aquesta funció rep un paràmetre amb el *ThirdPersonCharacter*. El primer que fa és assignar-lo a una variable *ThirdPersonCharacter*. Després, li treu la visibilitat al *NamePlate*, i posa les variables *isInteracting*, tant la pròpia com la del jugador, a cert. Al final, crida les funcions *PlayerAdjustments* i *StartDialogue*. Veure Figura 354.

Figura 354. NPC_Basic – Interact

6.12.4.2.7 PlayerAdjustments

Aquesta funció s'encarrega de fer la transició cap a l'inici d'un diàleg. El primer que fa és crear un *BlackScreenPop*, assignar-lo a la pantalla i esperar els 0.2 segons que triga en deixar la pantalla negra. A l'acabar, fa la transició de la càmera del jugador a la pròpia de l'NPC. Veure Figura 355.

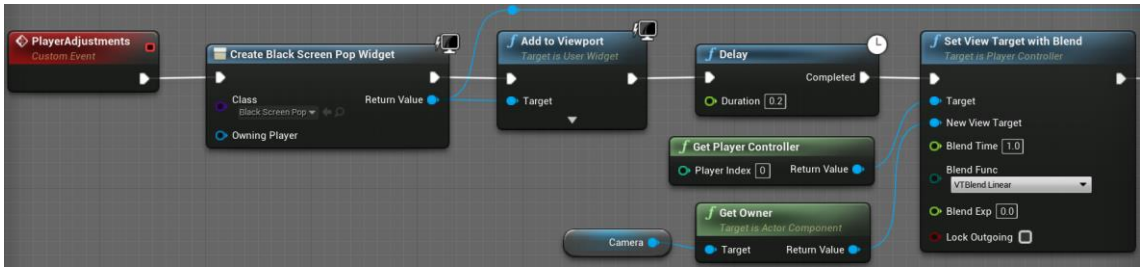

Figura 355. NPC_Basic - PlayerAdjustments - Part 1

A continuació, situa al jugador en el lloc i rotació indicat pel *Cone*. Després, s'espera els 0.2 segons que triga el *BlackScreenPop* a quedar-se invisible de nou, i l'elimina. Veure Figura 356.

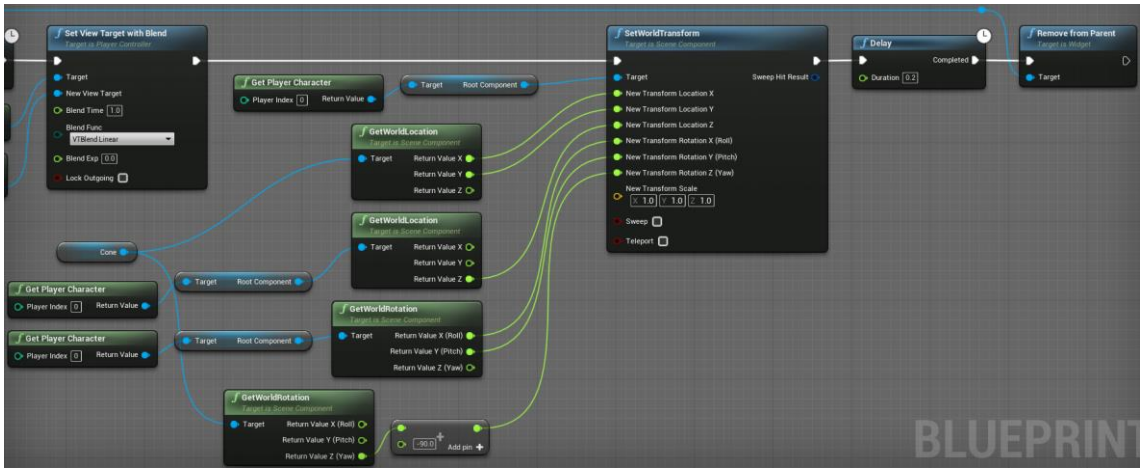

Figura 356. NPC_Basic - PlayerAdjustments - Part 2

6.12.4.2.8 StartDialogue

Implementació de la funció *StartDialogue* de la interfície I_NPC. Aquesta funció únicament s'espera el temps que triga *PlayerAdjustments* a executar-se i executa l'*StartDialogue* del *DialogueComponent* (Apartat 6.11.3.2.2). Veure Figura 357.

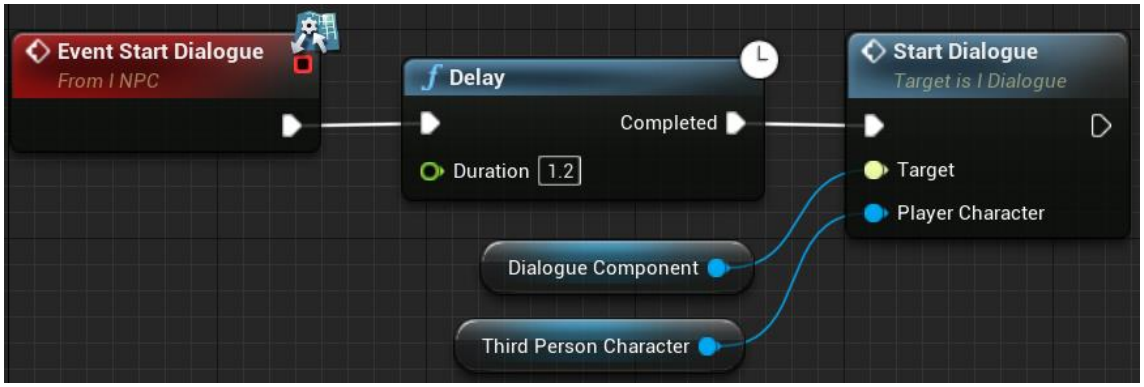

Figura 357. NPC_Basic - StartDialogue

6.12.4.2.9 EndDialogue

Implementació de la funció *EndDialogue* de la interfície I_NPC. Aquesta funció es crida quan s'acaba l'execució del diàleg. El primer que fa és tornar a situar la càmera del jugador on estava abans de començar el diàleg, i esperar a que acabi la transició (Figura 358). A l'acabar, fa que es deixi de veure el ratolí i posa les dues variables *isInteracting* a fals, tal com es veu a la Figura 359.

Figura 358. NPC_Basic - EndDialogue - Part 1

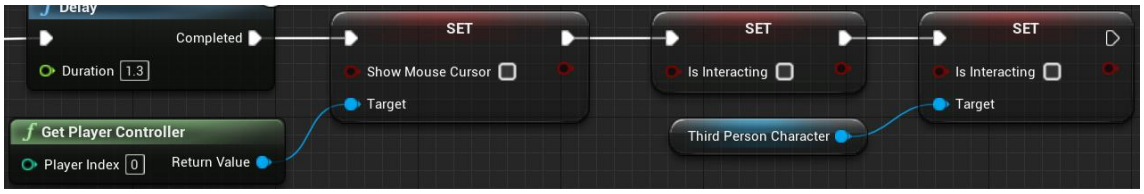

Figura 359. NPC_Basic - EndDialogue - Part 2

6.12.5 Lògica al ThirdPersonCharacter

La lògica situada al *ThirdPersonCharacter* serveix simplement per decidir si es pot interactuar amb algun NPC i amb quin, i per iniciar la interacció. Per saber amb quin NPC s'interactuarà, es té una llista "*collidingNPCs*" amb els disponibles i un "*currentNPC*" amb l'actual.

6.12.5.1 OnComponentBeginOverlap (CapsuleComponent)

Aquest *Event* es crida quan un objecte comença a interseccionar amb el *CapsuleComponent* del *ThirdPersonCharacter*. El què fa és comprovar que es tracti d'un *NPC_Basic* i, si és així, el guarda a la llista *collidingNPCs*. Veure Figura 360.

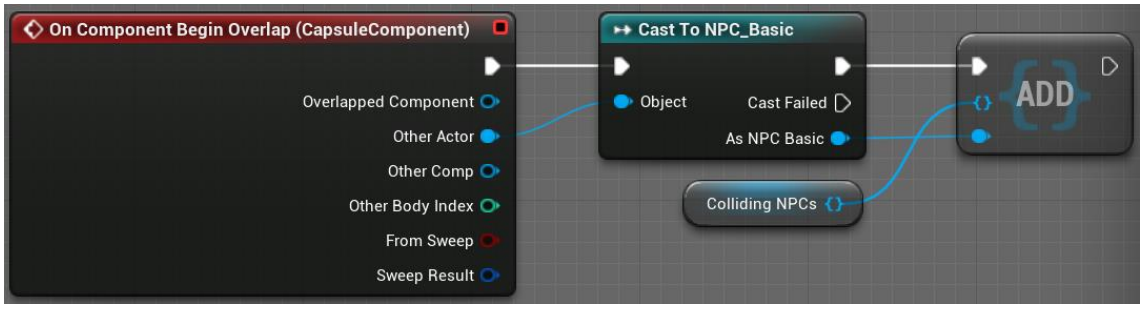

Figura 360. ThirdPersonCharacter - OnComponentBeginOverlap

6.12.5.2 OnComponentEndOverlap (CapsuleComponent)

Aquest *Event* es crida quan un objecte deixa d'interseccionar amb el *CapsuleComponent* del *ThirdPersonCharacter*. El què fa és comprovar que es tracti d'un *NPC_Basic* i, si és així, l'elimina de la llista *collidingNPCs*. Veure Figura 361.

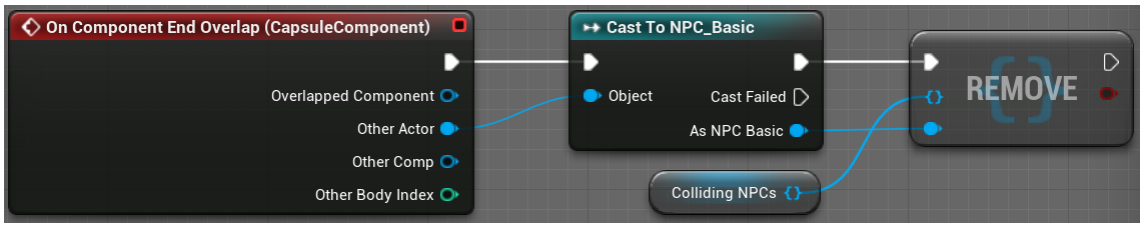

Figura 361. ThirdPersonCharacter - OnComponentEndOverlap

6.12.5.3 WhoCanInteract

WhoCanInteract és una funció que es crida a cada *Tick* del rellotge de *ThirdPersonCharacter*. El primer que fa és comprovar que hi hagi algun element a *collidingNPCs*. Si és així, posa la distància més petita a un NPC (*minDistanceNPC*) a una d'impossible, 1000000, si és que n'hi hagués algun. A més, fa un bucle a través de tots els elements de *collidingNPCs*. Veure Figura 362.

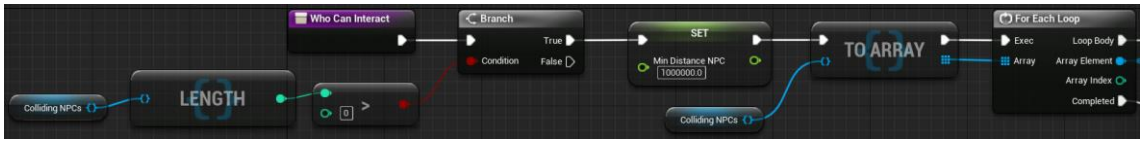

Figura 362. ThirdPersonCharacter - WhoCanInteract - Part 1

Per cada element, es desactiva la interacció amb *DeativateInteraction*. Després es comprova la distància de l'element al jugador. Si aquesta és inferior a la *minDistanceNPC* actual, es guarda l'element com a l'actual a *currentNPC* i la distància a *minDistanceNPC*. Veure Figura 363.

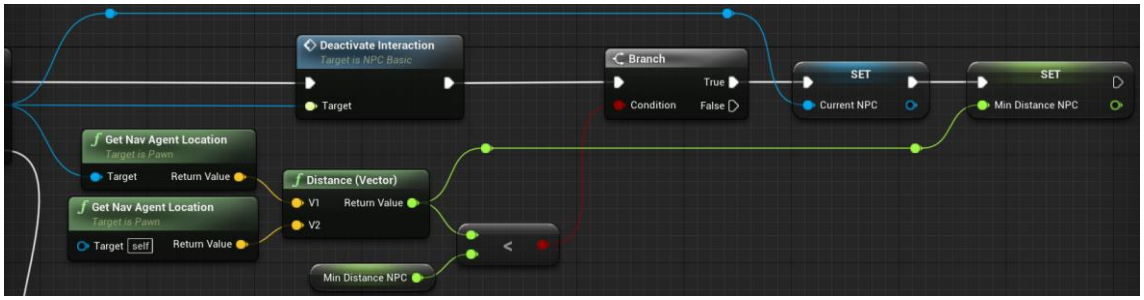

Figura 363. ThirdPersonCharacter - WhoCanInteract - Part 2

Per acabar, quan el bucle a finalitzar, s'activa la interacció de l'NPC més proper amb *ActivateIteraction*. Veure Figura 364.

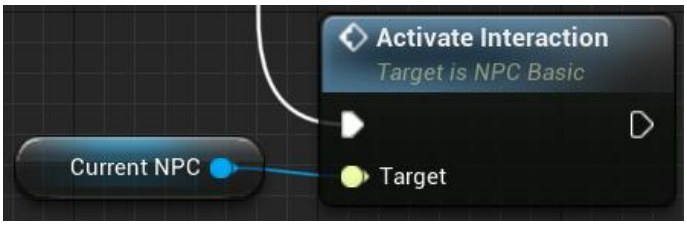

Figura 364. ThirdPersonCharacter - WhoCanInteract - Part 3

6.12.5.4 Interact

Aquest és un *Event* que s'activa quan es prem el botó d'interactuar. Primer de tot, comprova que *currentNPC* és vàlid, i si té la variable *isPlayerNear* a cert. Si és així, es crida la funció *StopAttackingAndDash* i l'*Interact* de l'*NPC_Basic*. Veure Figura 365.

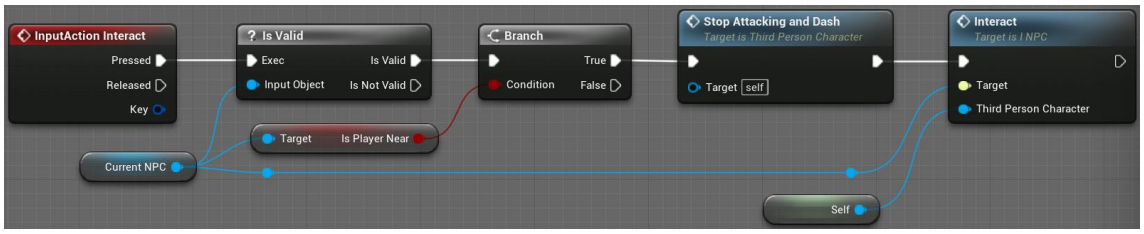

Figura 365. ThirdPersonCharacter - Interact

6.13 NPC's

Els NPC's utilitzats en el joc hereten tots de l'*NPC_BasicImplementation*. Aquest no és més que un component heretat de l'*NPC_Basic* explicat a l'Apartat 6.12.4. Això es fa per, si en algun moment cal implementar alguna cosa, que no s'hagués d'afegir a l'*NPC_Basic*. En aquest cas, no té gaire valor, ja que no afegeix res, però aquesta podria ser una opció vàlida si hi hagués diferents tipus d'NPC's.

Per a cada NPC diferent creat, s'ha buscat un model 3D i una animació que es repeteixi en bucle per a que no estigui sempre estàtic. Per a fer-ho, simplement s'ha creat un *AnimationBlueprint* propi de cada NPC que executi sempre la mateixa animació en bucle. Una vegada creat un NPC, s'assigna l'*SkeletalMesh* i l'*AnimationBlueprint* creat a l'apartat *Mesh* de l'estructura.

Hi ha 3 tipus d'NPC, segons la funcionalitat que tenen. El primer no té cap funcionalitat afegida. Simplement el canvi de *Mesh*. El segon tipus afegeix la possibilitat d'obrir un o diversos cofres si es té una conversa. En aquests, simplement s'afegeixen els cofres a l'estructura (el tipus *ChestClosed* explicat a l'Apartat 6.2.5) i un *Event SetChestsCanBeOpened* (veure Figura 366) que crida la funció *SetCanBeOpened* de tots aquests. El tercer es considera especial, perquè s'utilitza per a la recuperació de vida. Tot i així, no afegeix res més que un *StaticMesh* a l'estructura amb el model escollit.

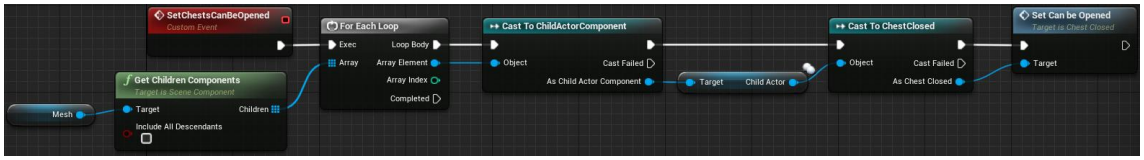

Figura 366. NPC - SetChestsCanBeOpened

Per els tipus 2 i 3 s'han creat *Tasks* especials per l'arbre de diàlegs. Les tasques del tipus 2 són totes iguals: obtenen l'NPC en concret i crida l'*Event SetChestsCanBeOpened*. Totes reben el nom *NPC_OpenChests* amb la localització i el nom de l'NPC al final. Com totes són iguals, només es posa el codi d'una d'elles com a exemple a la Figura 367. Aquestes tasques simplement s'han de col·locar a l'arbre de diàleg quan es vulgui que es permeti l'obertura dels cofres, i els cofres estaran disponibles a l'acabar el diàleg.

Figura 367. Task - NPC_OpenChests_Forest_Monk

Pel tipus 3, també s'ha creat una *Task*, encarregada de recuperar la vida al jugador. Per fer-ho, s'obté el *ThirdPersonCharacter* i es crida la funció *AddLife* amb la suma de la vida base i extra del jugador. Per acabar, acaba l'execució de la *Task*. Veure Figura 368.

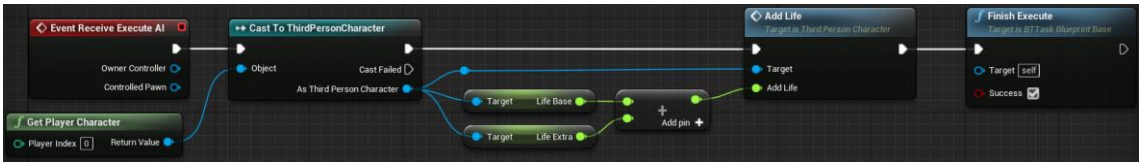

Figura 368. Task – Heal

6.14 Mapes

En el joc hi ha 2 mapes diferents. El primer i principal és el mapa del bosc, anomenat *ForestMap*. Aquest conté tot el joc, amb l'excepció de l'enemic final. L'altre mapa, anomenat *HeavenMap*, és on es té el combat final, i on acaba el joc.

6.14.1 ForestMap

ForestMap representa la part del joc que es juga en el món terrenal. Aquest és un món obert, amb un poble al principi, i un camí molt marcat cap a la zona del final del joc.

Una cosa molt important a comentar és que tota la part de textures i vegetació està generada a partir d'un paquet d'UE4 anomenat *Procedural Biomes*. Aquest implementa diversos tipus de terra i diferents grups d'arbres generats de forma procedural sobre el terreny segons el tipus de terra pintat. Per tant, la creació del mapa ha estat simplificada a la part de disseny i ús de les eines extretes de la botiga d'*Epic Games*.

6.14.1.1 Creació del Terreny

Per a la creació del terreny s'ha utilitzat un *Landscape*, un objecte d'UE4 pensat justament per aquesta feina, i hi ha hagut diversos passos.

El primer pas va ser obtenir un *heightmap* i aplicar-lo sobre el *Landscape*, donant una primera versió de les diferents elevacions. Just després, es van modificar i suavitzar les elevacions a mà per a fer-les molt més adients al joc. Aquesta part és important que es relacioni directament amb l'experiència del jugador en cada zona, perquè no es voldria tenir un terreny gens pla per als combats, ni un terreny absolutament pla que no dona opció a explorar. A més, es van haver de fer els límits del mapa, generant elevacions altíssimes en el propi *Landscape*.

Un cop obtingut el mapa amb les elevacions desitjades, cal passar al segon pas: pintar el terra amb les diverses textures. S'han utilitat 4 tipus de textura, dues amb herba per a les zones amb arbres, i les altres dues per a les zones sense. Primer es van utilitzar les textures per a pintar tot amb herba. Una vegada fet això, es sobreescriuen les textures amb les altres dues a les zones que no interessa que hi hagi herba. Una de les textures s'utilitza per fer els camins que principalment seguirà el jugador. L'altra s'utilitza en zones on no és adient que hi hagi herba i no sigui un camí. Tot i que sembla que les 4 textures tenen funcions molt diferents, en la realitat es combinen unes amb altres per a donar una sensació més realista.

Per últim pas, es fa la generació automàtica dels arbres i les pedres. El resultat final és el què es pot veure a la Figura 369.

Figura 369. ForestMap

6.14.1.2 Poble

Amb el terreny acabat, cal afegir tots els elements que no formen part del *Landscape*. Un d'ells és el poble. Aquest no està fet més que per uns quants models extrets d'Internet i col·locats com millor s'ha cregut que funcionaria. A més dels edificis, també s'han afegit tots els NPC's amb diàlegs. El resultat final es pot veure a la Figura 370 i la Figura 371.

Figura 370. ForestMap - Poble - Part 1

Figura 371. ForestMap - Poble - Part 2

6.14.1.3 Camí al Cel

A més del Poble, hi ha un altre element important en el mapa: el Camí al Cel. Aquest està format per una escala de núvols que porta fins a una porta que desprèn molta llum. Aquest és el final del mapa, i porta cap al mapa final. El resultat final es pot veure a la Figura 372.

Figura 372. ForestMap - Camí al Cel

6.14.1.4 Col·locació de Cofres i Estàtues

L'últim element important és la col·locació tant dels cofres com de les estàtues de recuperació de vida. De cofres n'hi ha de molts, alguns d'ells estan situats de forma estratègica i tenen assignada una runa en concret, i altres són molt més aleatoris. Això es fa per poder obtenir sí o sí una runa de cada tipus, si és que s'intentés. El cofres amb una runa assignada estan situats al final de camins secundaris marcats, per a que siguin més o menys fàcils de trobar, mentre que els aleatoris estan situats també de forma més aleatòria. Pel què fa a les estàtues, estan totes situades al que es considera el camí principal, i hi ha una cada cert espai. D'aquesta forma, el jugador pot anar a recuperar la seva vida i llocs lògics i fàcils de recordar.

6.14.2 HeavenMap

HeavenMap és la zona de joc per combatre l'enemic final. Aquest és molt més senzill, però té un disseny molt conceptual, intentant representar el què podria arribar a ser el cel. Està format simplement per un cel, unes portes tancades, i un terra reflectant. El resultat es pot veure a la Figura 373.

Figura 373. HeavenMap

Per a fer que el terra sigui reflectant, cal fer diverses coses. Primer de tot, cal crear un material especial, posant-li que sigui de color blanc, que tingui un valor de metàl·lic al màxim i una rugositat de 0. Veure Figura 374. Quan ja es té, simplement s'assigna al terra.

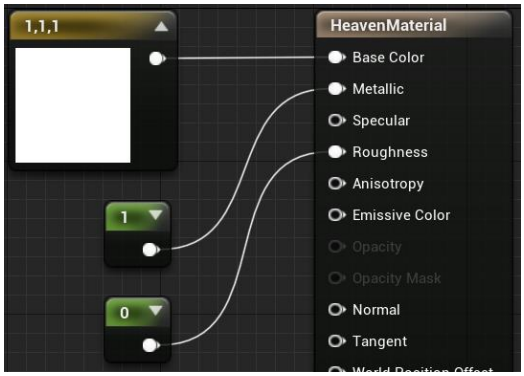

Figura 374. HeavenMaterial

Per acabar, cal activar el suport d'UE4 per a reflexió especular a la configuració del projecte (veure Figura 375), i afegir un objecte *PlanarReflection* sobre el terra.

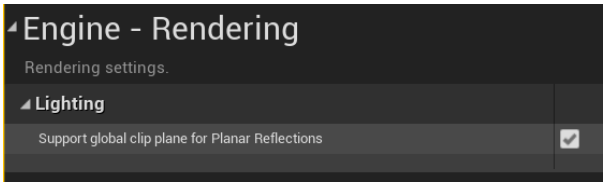

Figura 375. Configuració per a la reflexió especular

6.15 Sistema de Guardat

Per a poder guardar la partida a UE4 es necessiten dues coses. La primera és una estructura de dades que obtingui tota la informació rellevant i es pugui guardar. La segona cosa important és la lògica que s'encarrega de guardar la informació i d'obtenir-la de nou al carregar la partida.

6.15.1 Estructura de Dades

De l'estructura, hi ha hagut un parell d'elements que ja s'han comentat. Per a guardar l'inventari, s'utilitza l'*ST_Item* explicat a l'Apartat 6.1.1. Aquest element s'utilitza per crear les estructures de l'inventari i les runes incrustades a l'objecte *GI_Inventory*, explicat a l'Apartat 6.1.3. L'*inventoryData* es va utilitzant en la lògica de l'inventari, però *runesData* no, ja que s'utilitza gairebé exclusivament en el sistema de guardat. L'únic lloc on s'utilitza apart d'aquest sistema és a l'Apartat 6.1.6.5. A més a més, es crea una altra estructura de dades anomenada *ChestData*. Aquest serveix per guardar l'estat dels cofres dels mapes. Té només 3 atributs: nom, posició i si està obert o no, tal com es veu a la Figura 376.

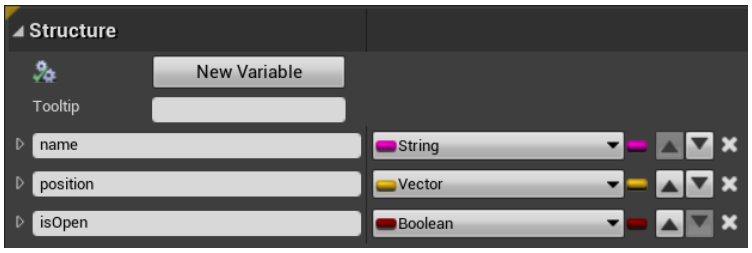

Figura 376. ChestData

Amb tots aquests elements, ja es pot crear l'objecte que es guardarà. Aquest s'anomena *BP_SaveGame*, i es tracta d'un *Blueprint* que hereta del tipus *SaveGame* propi d'UE4, un objecte pensat justament per aquesta funció: guardar un estat d'una partida de forma persistent.

L'estructura del *BP_SaveGame* cobreix diferent tipus d'informació de la partida. Primer de tot, guarda el mapa en què es troba el jugador i la seva posició i rotació, amb les variables *levelName* i *characterTransform*, respectivament. A més, guarda l'estat del jugador amb les variables de vida, atac i experiència (*life*, *lifeBase*, *attackBase*, *level*, *experience* i *experienceNextLevel*). A més, guarda la informació de l'inventari i les runes amb les variables *inventoryData* i *runesData*, llistes del tipus *ST_Item*. Per acabar, té un sistema per guardar quins cofres de cada mapa han estat oberts. El què es fa és tenir una llista amb els noms dels possibles mapes, i una llista de tipus *ChestData* per a cadascun d'ells. Llavors, depenent del mapa a guardar o carregar, s'agafa una estructura o una altra. Aquesta estructura està pensada per a tenir diversos mapes amb cofres, tot i que aquest no sigui el cas. Veure Figura 377.

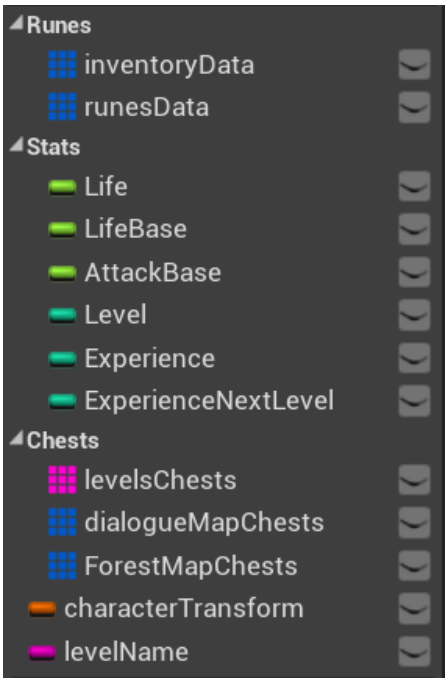

Figura 377. BP_SaveGame

6.15.2 Lògica

Amb l'estructura de dades ja feta, ja es poden implementar tots els *Events* que es necessitaran per a fer la gestió d'un partida guardada. Aquestes funcions són les que es cridaran quan es vulgui guardar partida, carregar, reiniciar, etc. Estan totes implementades al *Blueprint ThirdPersonGameMode*, el qual es configura com a "Game Mode" a les configuracions del projecte.

6.15.2.1 SaveGame

Aquest *Event* serveix per a guardar la partida. És a dir, agafa les dades indicades i les guarda a un *BP_SaveGame*. Per fer-ho, primer de tot, obté el *BP_SaveGame* amb el nom configurat. Just després, obté el nom del mapa actual i el guarda. Obté també el personatge i guarda la informació de posició i rotació i de les variables de vida, atac i experiència. Veure Figura 378 i Figura 379.

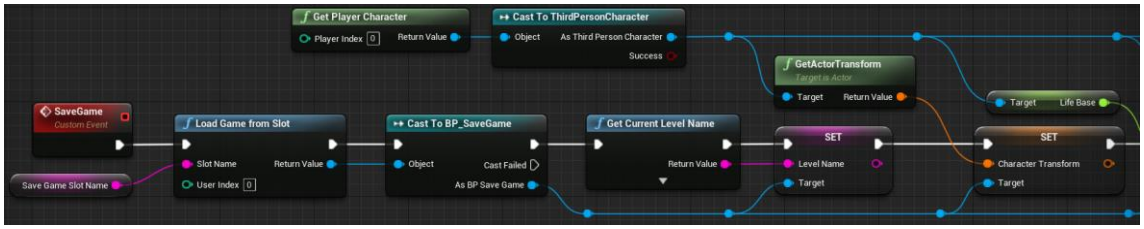

Figura 378. SaveGame - Part 1

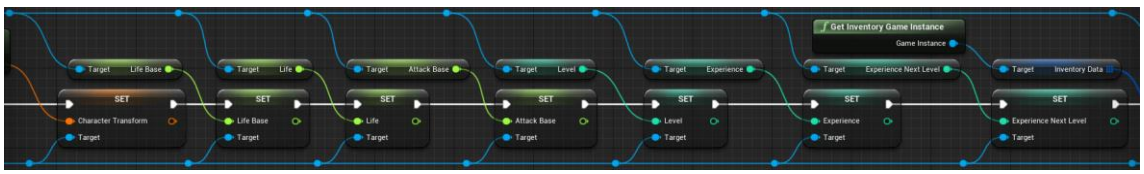

Figura 379. SaveGame - Part 2

A continuació, obté la instància de l'inventari, i el guarda. També guarda les runes incrustades, obté la variable amb la llista de cofres corresponent i li borra la informació,

tal com es veu a la Figura 380. S'obtenen tots els cofres del mapa i es torna a emplenar la variable de cofres. Per acabar, guarda l'estructura de dades amb el nom configurat. Veure Figura 381.

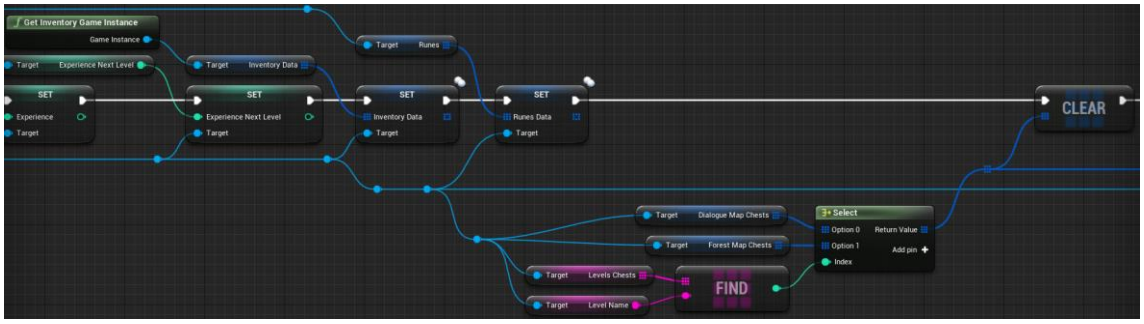

Figura 380. SaveGame - Part 3

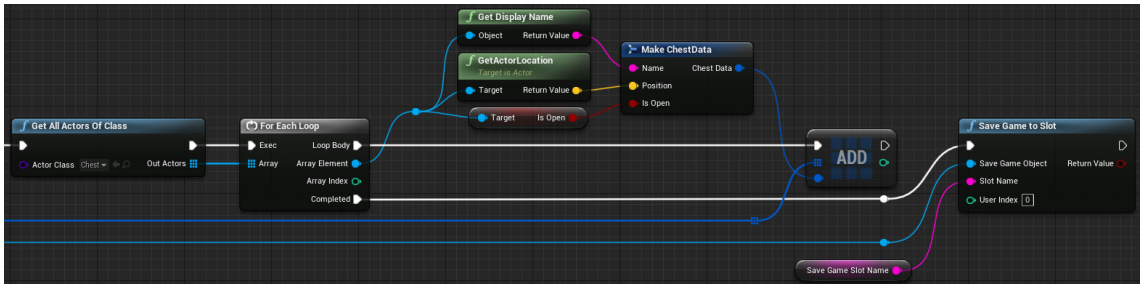

Figura 381. SaveGame - Part 4

6.15.2.2 LoadGame

Aquest *Event* té l'objectiu invers de l'anterior: obtenir la informació i escriure-la sobre els objectes afectats. Primer de tot, obté la informació guardada amb el nom configurat, i comprova que el mapa sigui el mateix al guardat. Si no es així, carrega el mapa configurat. En cas contrari, segueix l'execució. Veure Figura 382.

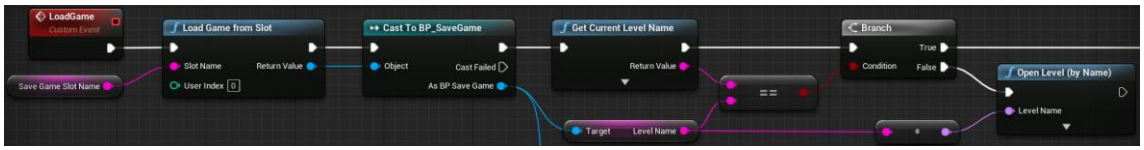

Figura 382. LoadGame - Part 1

Es cas que el mapa sigui el mateix, s'obtenen la posició i rotació, i les variables de vida, atac i experiència, i s'assignen els valors al jugador. Veure Figura 383 i Figura 384.

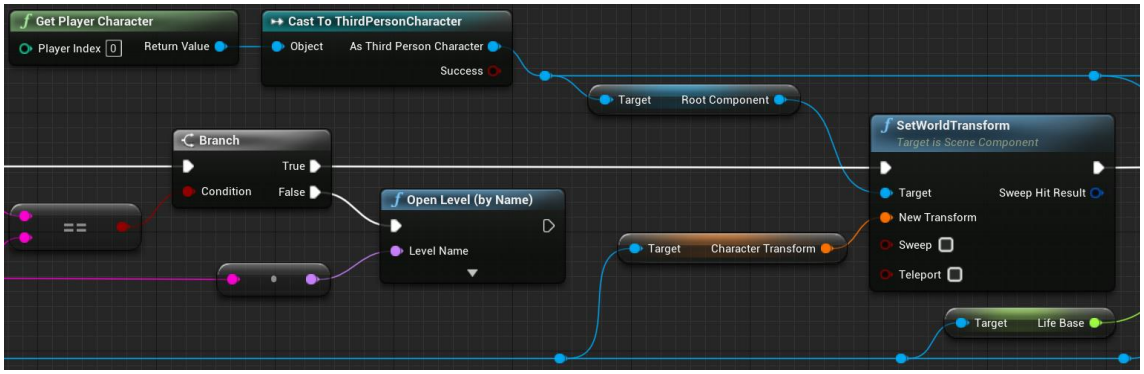

Figura 383. LoadGame - Part 2

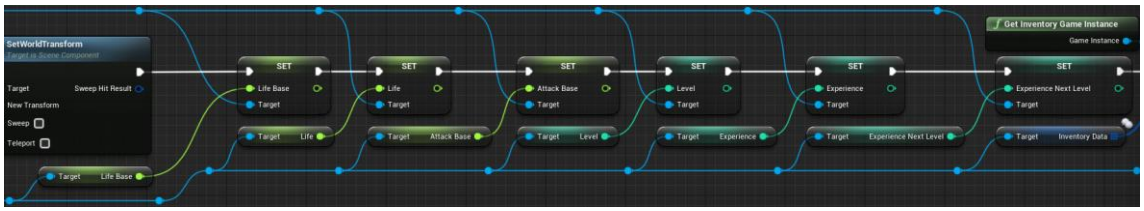

Figura 384. LoadGame - Part 3

Després, s'obté la informació de les runes, i s'assigna al *GI_Inventory*. Veure Figura 385.

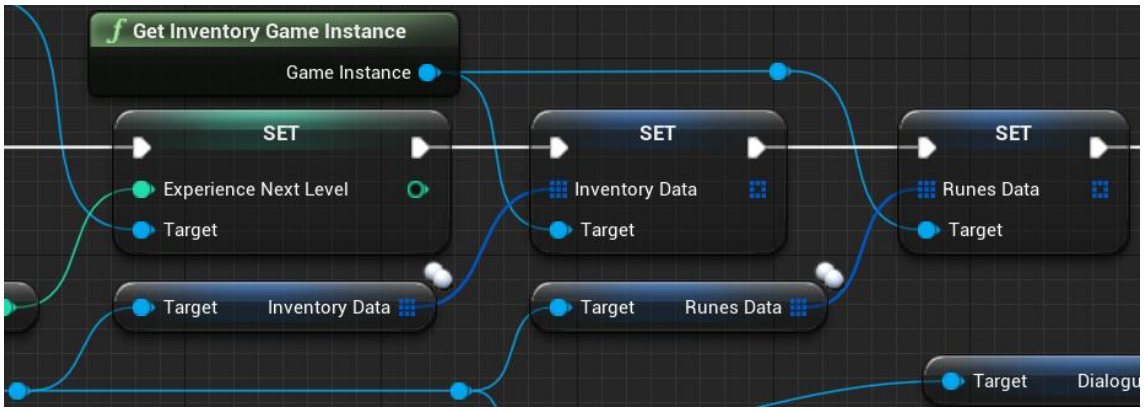

Figura 385. LoadGame - Part 4

Per acabar, s'obté la informació dels cofres corresponent, s'obtenen els cofres del mapa, i es va buscant un per un el cofre coincident. Amb cada cofre, si s'ha guardat que està obert, es crida la funció *SetOpened* del *Chest*. Veure Figura 386.

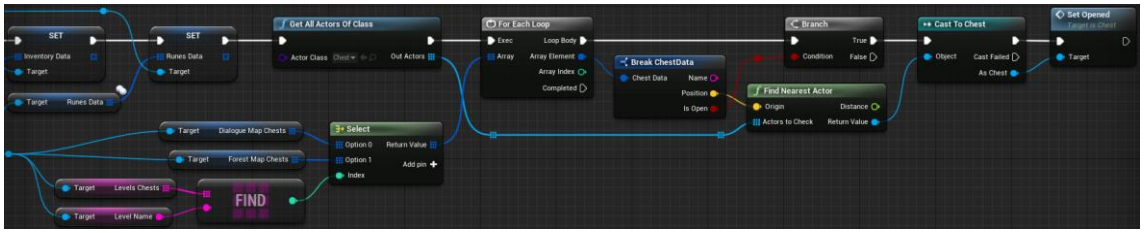

Figura 386. LoadGame - Part 5

6.15.2.3 Event BeginPlay

Aquest *Event* es crida a l'iniciar la partida. El què fa és mirar si existeix un objecte guardat del tipus *BP_SaveGame* amb el nom configurat. Si existeix, crida *LoadGame* per carregar la partida. En cas contrari, crea un objecte nou, el guarda amb el nom configurat, i crida *SaveGame* per guardar la informació inicial de la partida. Veure Figura 387.

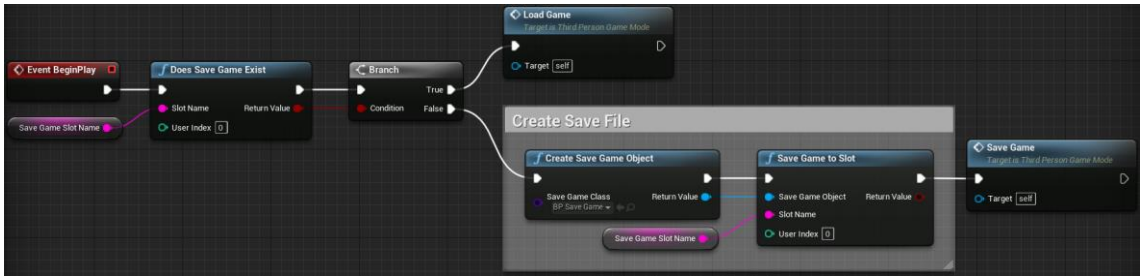

Figura 387. Event BeginPlay

6.15.2.4 ReloadGame

Aquest *Event* executa una comanda per reiniciar el nivell. Veure Figura 388.

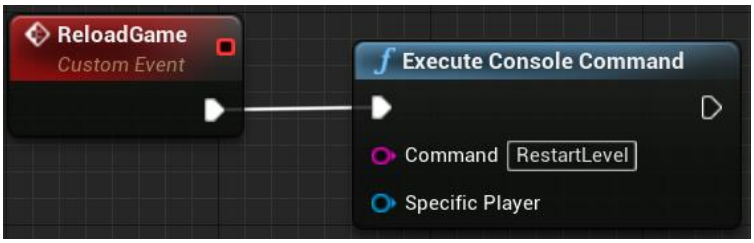

Figura 388. ReloadGame

6.15.2.5 DeleteSaveFile

Aquest *Event* elimina la partida guardada. Veure Figura 389.

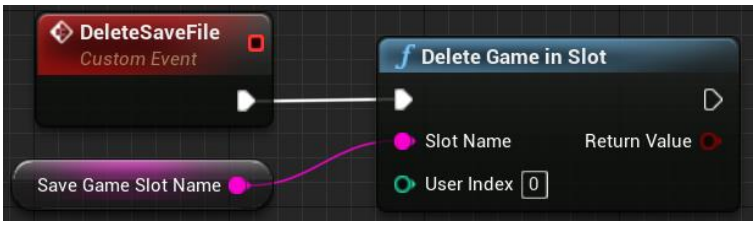

Figura 389. DeleteSaveFile

6.15.2.6 SaveMapAndTransform

Aquest *Event* sobreescriu el nom del mapa i la posició i rotació del personatge a la partida guardada amb els valors que rep. Veure Figura 390.

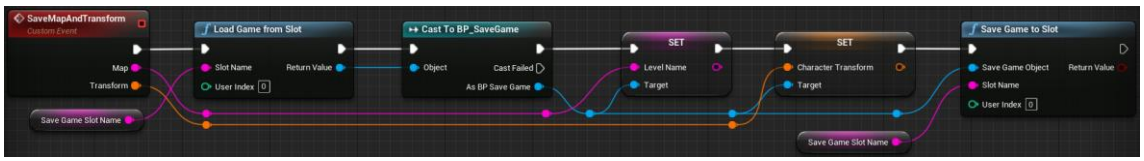

Figura 390. SaveMapAndTransform

6.15.2.7 ChangeMap

Aquest *Event* serveix per fer un canvi de mapa conservant la informació. El què fa és guardar la partida actual, sobreescriure el mapa i la posició i rotació del personatge cridant l'*Event SaveMapAndTransform* i carregar el mapa entrat. Veure Figura 391.

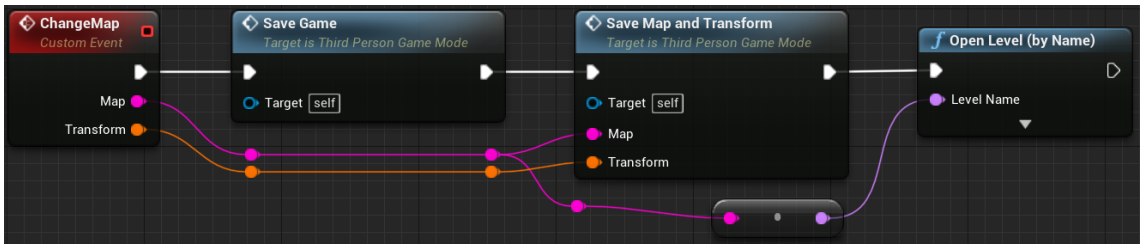

Figura 391. ChangeMap

6.16 Sistema de So

El sistema de so s'encarrega de tots els sons qui hi ha en el joc, tant de la música de fons, com dels sons que pugui fer el jugador.

6.16.1 Sons del Jugador

Dels sons del jugador ja s'ha fet referència, tant del so del *Dash* (Apartat 6.3.4), com de l'espasa (Apartat 6.6.7.1). A més d'aquests dos, també hi ha dos sons per a quan el jugador camina. Per a cada so del personatge, s'ha de tenir la configuració descrita a

continuació. El primer que cal fer és crear un *SoundCue*. Aquest és un component propi d'UE4. Rep un o diversos sons, i permet fer-ne configuracions i modificacions. Al final, cal que hi hagi una sortida.

Per als sons del personatge hi ha 4 *SoundCue*: un pel *Dash*, un pels cops d'espasa, i dos pels passos, un d'herba i un de vidre. En els sons del *Dash* i passos, simplement es posa un so i s'uneix directament a la sortida. En els dels passos cal fer que el so s'executi en bucle. Es pot veure la configuració a la Figura 392. Pel so del *Dash* no cal fer res especial.

Figura 392. GrassFootstepsSounds

El so de l'espasa és una mica diferent. Aquest rep 4 sons diferents. Per a fer que tinguin el volum igual entre ells, es posa un modulador a 3. Després, es fa que a la sortida hi arribi només un dels 4 amb un node que escull un de forma aleatòria. D'aquesta forma es poden tenir sons diferents de forma senzilla. Veure Figura 393.

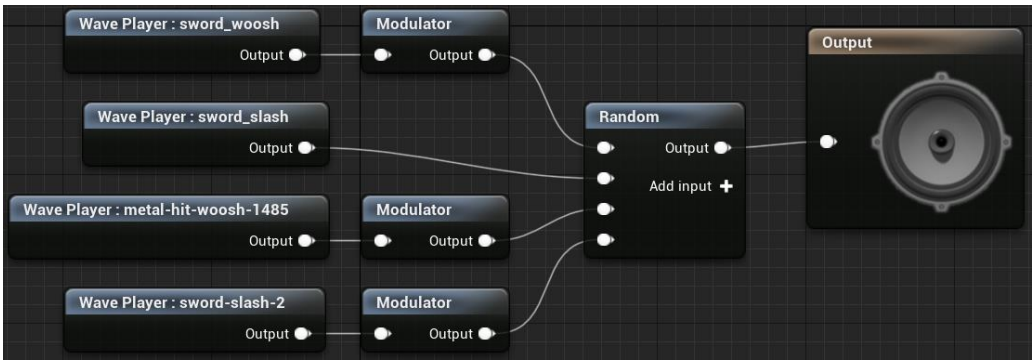

Figura 393. SwordSounds

Quan ja es tenen els components, s'ha d'afegir al personatge un *AudioComponent* per cadascun d'ells (dels passos només un), i configura'ls-hi els *SoundCue*. Aquest és un component que s'utilitza per executar un so i afegir-hi una mica de funcionalitat molt bàsica. Pels passos només en cal 1 perquè quan es carrega la partida, es configura segons el mapa. Amb els *AudioComponent* afegits i configurats, simplement cal cridar la funció Play per iniciar els sons.

Pels sons dels passos hi ha una mica més de complexitat, ja que aquests s'executen si s'està caminant. Per aquesta lògica, s'afegeix una funció *SetFootstepsSound* al *Tick* del personatge. En aquesta funció, el primer què es fa és comprovar que la velocitat del personatge no sigui 0, que estigui tocant el terra, i no estigui ni executant un *Dash* ni un atac. Veure Figura 394.

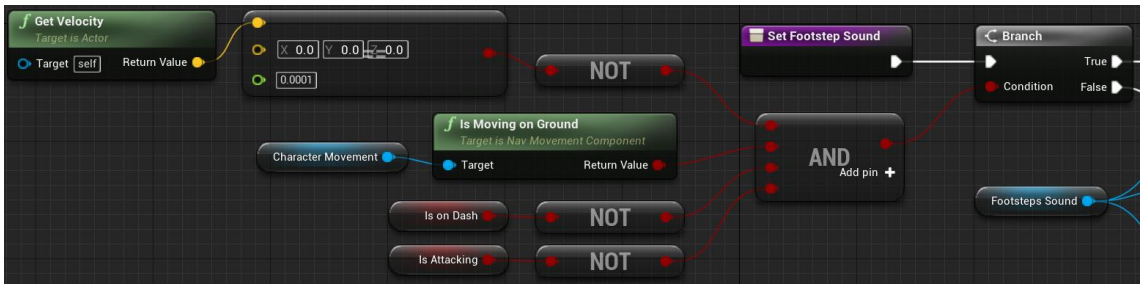

Figura 394. SetFootstepsSound - Part 1

En cas de no complir-se les condicions, s'atura el so. En cas contrari, es mira si ja estava actiu. Si no ho estava, es calcula (veure Figura 396) i s'assigna la velocitat del so per a que correspongui amb la velocitat de les passes, ja que aquesta pot canviar gràcies a les runes. Per acabar, s'inicia l'execució del so. Veure Figura 395.

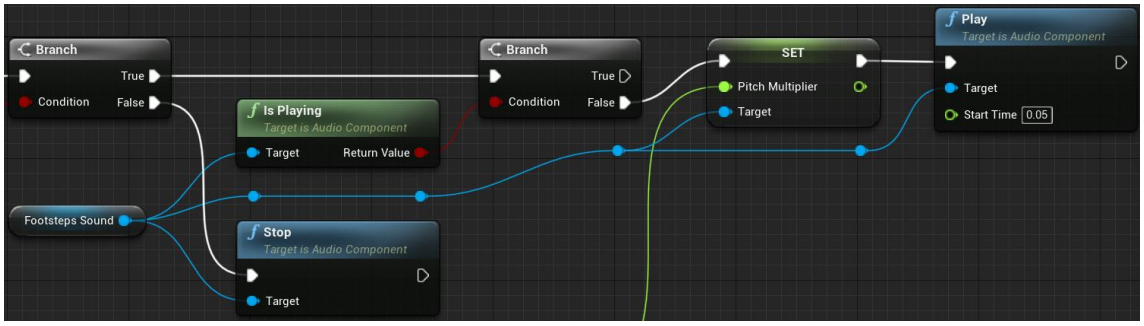

Figura 395. SetFootstepsSound - Part 2

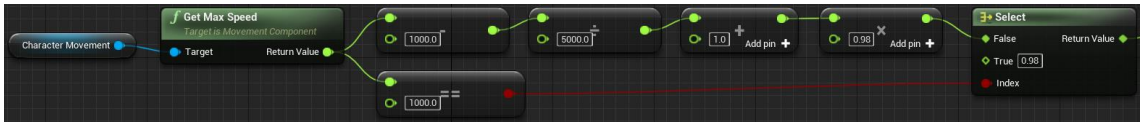

Figura 396. SetFootstepsSound - Part 3

6.16.2 Música de Fons

La música de fons és una mica diferent als sons del personatge. Aquesta és diferent en tots dos mapes. Pel mapa del bosc hi ha dues cançons, una per a l'exploració i una altra per al combat. El mapa del cel només té una. Per a executar la música, en tots dos casos s'utilitzen els *SoundCue*. A més, hi ha un controlador per la música de cada mapa.

6.16.2.1 ForestBackgroundMusicManager

ForestBackgroundMusicManager és el controlador de la música del mapa del bosc. Aquest té un *AudioComponent*, i la lògica per canviar la música entre exploració i combat.

6.16.2.1.1 Event BeginPlay

Aquest *Event* s'executa a l'iniciar la partida, i configura l'àudio per a que es pugui executar mentre el joc està en pausa i posa *initialTime* al temps actual. Veure Figura 397.

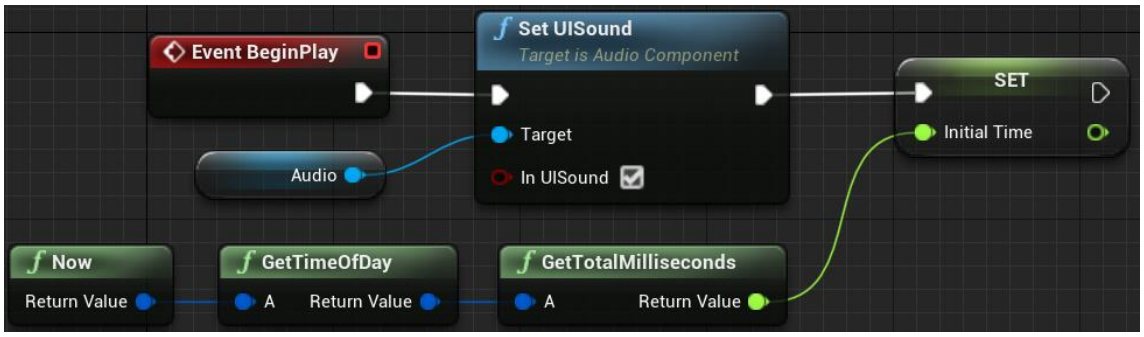

Figura 397. Event BeginPlay

6.16.2.1.2 SetCombatMusic

Aquest *Event* canvia la música cap a la de combat. Per a fer-ho, primer guarda quin temps ha estat executant-se la música d'exploració (si es passa, reinicia el comptador) a *playedTime*. Veure Figura 398.

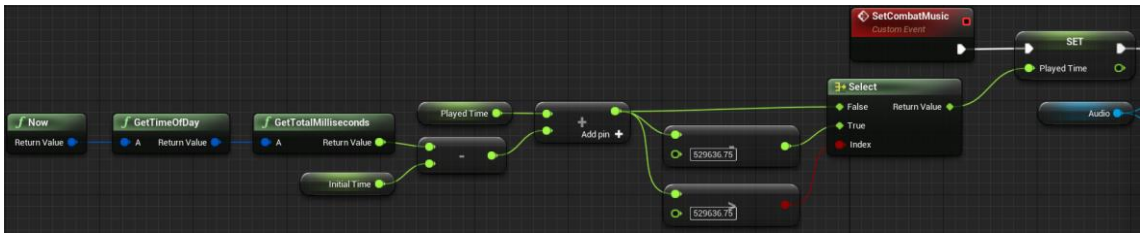

Figura 398. SetCombatMusic - Part 1

A continuació, fa un *FadeOut* amb la música d'exploració i un *FadeIn* amb la de combat. Veure Figura 399.

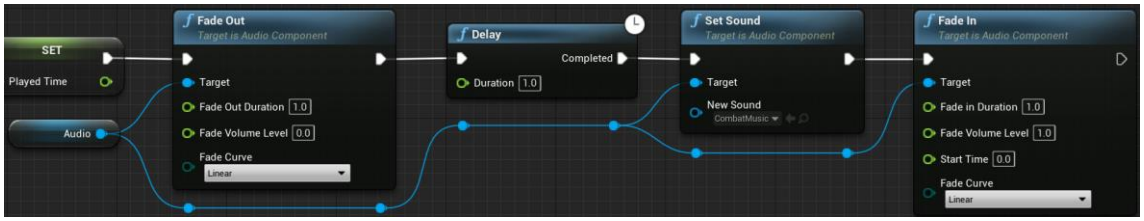

Figura 399. SetCombatMusic - Part 2

6.16.2.1.3 SetForestMusic

Aquest *Event* canvia la música cap a la d'exploració. Fa un *FadeOut* amb la música de combat i un *FadeIn* amb la d'exploració. La música d'exploració la inicia a partir del segon indicat al *playedTime*. A més, guarda el moment en què ha començat a sonar. Veure Figura 400.

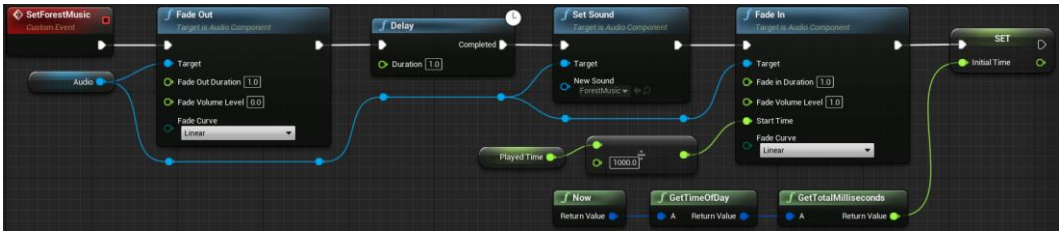

Figura 400. SetForestMusic

6.16.2.1.4 ChangeIsInCombat

Aquest *Event* s'executa quan es canvia d'exploració a combat o viceversa. Simplement posa la variable *isInCombat* indicant si s'està en combat i crida un dels dos canvis de música. Veure Figura 401.

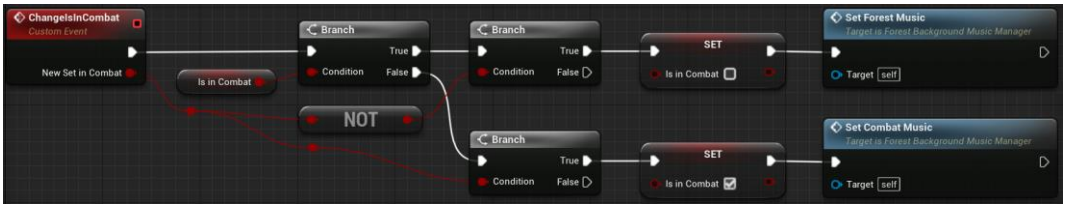

Figura 401. ChangeIsInCombat

6.16.2.1.5 ChangeInCombat

Aquest és un *Event* que es crida cada vegada que un enemic comença a atacar al jugador, o deixa de fer-ho, indicant si és un cas o l'altre amb *addMinion*. El què fa l'*Event* és actualitzar la quantitat d'enemics que estan en combat a la variable *inCombat*, i cridar *ChangeIsInCombat*, amb el paràmetre d'entrada a cert si *inCombat* és major a 0, i fals en cas contrari. Veure Figura 402.

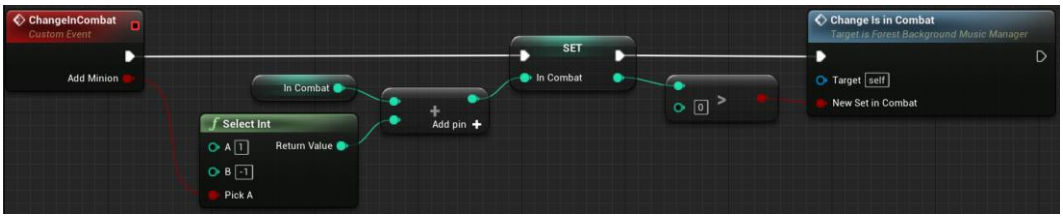

Figura 402. ChangeInCombat

6.16.2.2 HeavenBackgroundMusicManager

HeavenBackgroundMusicManager és el controlador de la música del mapa del cel. Aquest té un *AudioComponent* i la lògica per a reproduir el so adient.

6.16.2.2.1 Event BeginPlay

Aquest *Event* s'executa a l'iniciar la partida, i configura l'àudio per a que es pugui executar mentre el joc està en pausa. Veure Figura 403.

Figura 403. Event BeginPlay

6.16.2.2.2 PlayMusic

Aquest *Event* inicia la música des de zero. Veure Figura 404.

Figura 404. PlayMusic

6.17 Menús

6.17.1 MenuNavigationFunctions

A mesura que es va desenvolupar la navegació dels menús es va observar que tots els *Widgets* tenien algunes funcions en comú. Per aquest motiu s'ha creat aquesta interfície que permet simplificar el codi en algunes parts fent que els *Widgets* implementin aquestes funcions. Les funcions són les quatre mostrades a la Figura 405. *SelectActual* i *DeselectActual* s'utilitzen quan s'obre o es tanca un menú, per canviar l'estil del botó actual (en el cas de *SelectActual* acostumarà a ser el que es troba en la posició més superior). *NavigateUpDown* i *LeftRight* està pensat per canviar la selecció del botó dins el menú

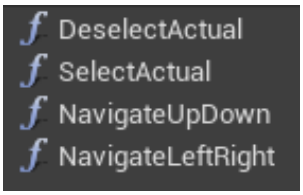

Figura 405. Funcions de la interfície

6.17.2 Navegació des del ThirdPersonCharacter

Pel cas del menú de pausa (i el d'opcions si s'accedeix des d'aquest), es criden les funcions de navegació des dels *Events UpDownGamepad* i *LeftRightGamepad*, esmentats als Apartats 6.1.7.2.8 i 6.1.7.2.9. En cas que l'inventari no estigui obert, s'arriba a la part de codi de la Figura 406. La navegació esquerra i dreta s'utilitza només al menú d'opcions, explicat a l'Apartat 6.17.5. En canvi, la navegació amunt i avall s'utilitza tant al menú d'opcions com al de pausa, desenvolupat a l'Apartat 6.17.4. Es dona prioritat a les opcions, però totes dues criden a la mateixa funció gràcies a la creació de la interfície *MenuNavigationFunctions*, esmentada a l'apartat anterior.

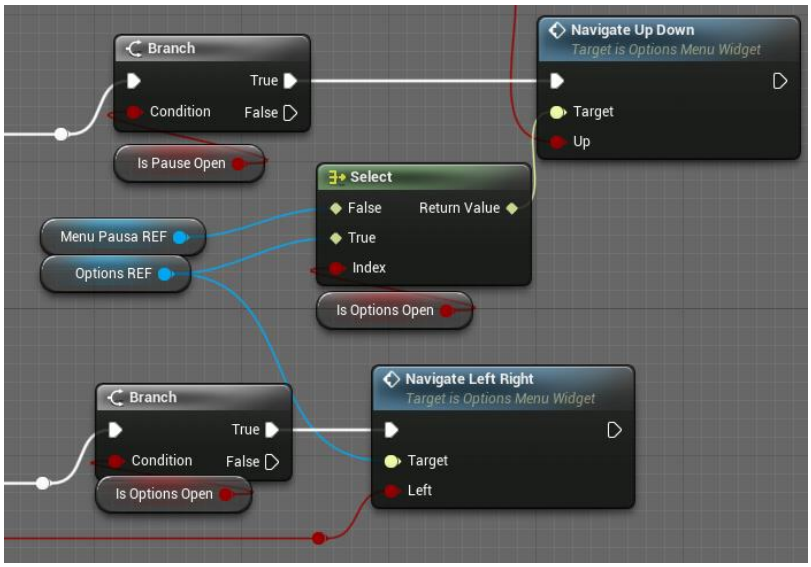

Figura 406. Navegació pels menús al ThirdPersonCharacter

6.17.3 Menú d'inici

El menú d'inici es troba dins d'un mapa buit *MainMenuLevel* que conté un *Widget* anomenat *Main Menu Widget* i un *Level Widget* anomenat *Main Menu Level Game Mode Base*, que permet carregar la configuració pertinent per iniciar el joc des del menú principal.

6.17.3.1 Main Menu Level Game Mode Base

El més important d'aquest *Level Widget* és la configuració de les classes que hi ha a la Figura 407. Concretament, el paràmetre més destacable és *Default Pawn Class*, que es troba a *None*. Això ha de ser així per tal que, al començar, no faci aparèixer a la protagonista i només aparegui el que ens interessa, que és el menú principal (Figura 29).

Figura 407. Informació de les classes del Level Widget "Main Menu Level Game Mode Base"

6.17.3.2 Main Menu Widget

Aquest *Widget* conté tota la informació i la lògica del funcionament del menú principal.

Figura 408. Elements del disseny del menú principal

Com es pot veure a la Figura 408 hi ha un total de sis elements que formen part d'aquest *Widget*: dues imatges (*Fondo* i *Titulo*) i quatre botons (*NewGame*, *LoadGame*, *OptionsMain* i *ExitGame*).

6.17.3.2.1 Imatges Main Menu Widget

Encara que la imatge *Titulo* només sigui una textura exportada des d'*Illustrator*, la imatge Fondo és en realitat un vídeo d'una animació. Per tal de poder afegir el vídeo s'han de fer un parell més de coses. El primer pas és importar dins la carpeta de *Content* el vídeo en format ".mp4" de l'animació. Un cop afegida, s'ha de crear un arxiu de tipus *File Media Source* i assignar la ruta d'on es troba aquest arxiu ".mp4". Veure Figura 409.

Figura 409. Ruta de l'arxiu ".mp4" dins de l'arxiu File Media Source

Un cop creat i assignat, s'ha de crear un arxiu del tipus *Media Player* (això també ens crearà automàticament un arxiu *MediaTexture*), on dins es pot seleccionar l'arxiu FMS que s'ha creat just abans. Veure Figura 410.

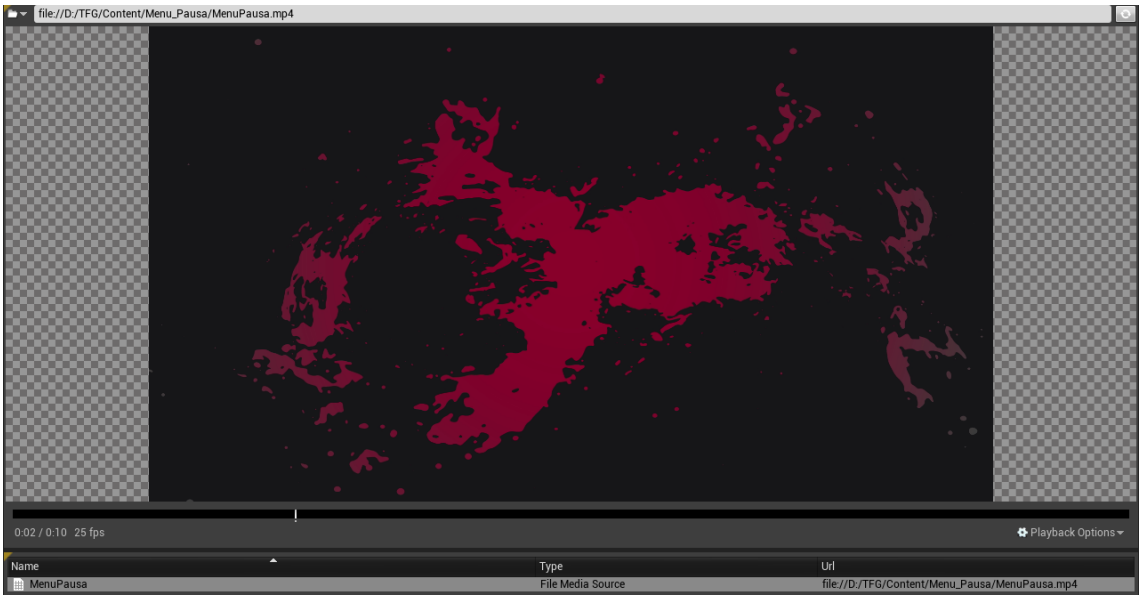

Figura 410. Arxiu "Media Player" amb l'arxiu FMS assignat

Quan l'arxiu FMS ja ha estat assignat al *Media Player*, automàticament s'actualitzarà el *MediaTexture*. El que s'ha de fer a continuació és crear un nou material partint d'aquest

arxiu, fent clic dret a sobre. Amb el nou material creat s'ha d'assignar a la imatge *Fondo* per tenir-ho tot preparat per poder reproduir l'animació. Veure Figura 411.

Figura 411. Material assignat a la imatge de fons per poder més endavant reproduir l'animació

6.17.3.2.2 Variables

Com es pot veure a la Figura 412, es té com a variables les imatges del *Widget* i els quatre botons que es poden clicar, a més d'una referència al *Widget* de menú d'opcions i un booleà que indica si el menú d'opcions és ja obert o no.

Figura 412. Variables del Main Menu Widget

6.17.3.2.3 Event ExitGame

Aquest *Event* s'executarà quan el jugador faci clic sobre el botó *Exit Game*, que cridarà a la funció *Quit Game* que tancarà el joc de manera automàtica, com es pot veure a la Figura 413.

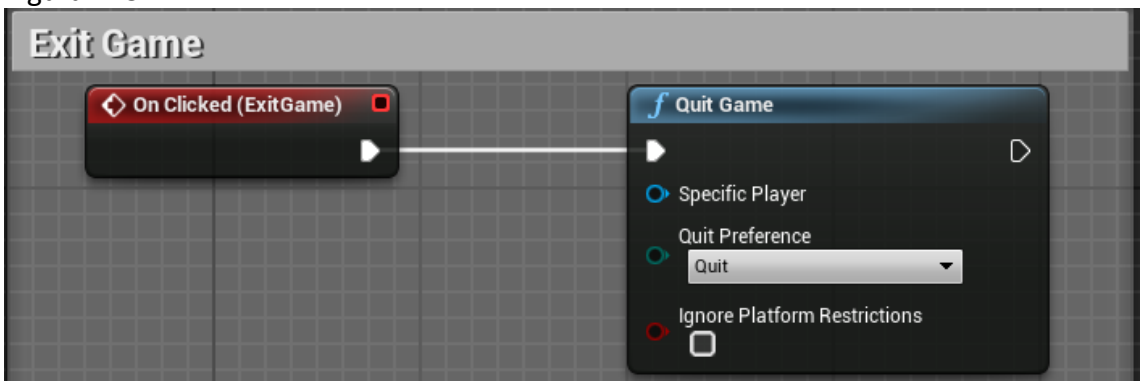

Figura 413. Execució quan es dispara l'event OnClicked del botó Exit Game al Main Menú Widget

6.17.3.2.4 Event Options

A la Figura 414 es pot veure l'execució per obrir el menú de pausa des del *Main Menu*.

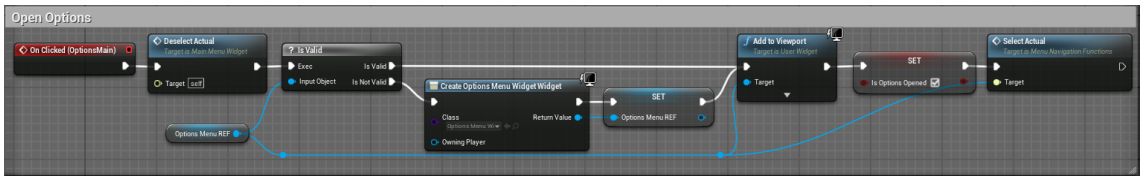

Figura 414. Event OnClicked del botó d'opcions del Main Menu Widget

El que es fa és una comprovació de si la referencia emmagatzemada del *Widget* del menú d'opcions és vàlida (si existeix o, si ja existia, si és correcta). En cas que no sigui vàlida, crearà aquest *Widget* i l'emmagatzemarà a la variable *OptionsMenuREF*. Un cop creada o validada, agafa aquesta referència i la fica sobre el *Viewport* deixant el *Main Menu* en segon pla ficant la variable *IsOptionsOpened* a cert.

6.17.3.2.5 Event NewGame

Aquest *Event* s'executarà quan el jugador faci clic al botó de *New Game*, esborrant qualsevol partida existent (si existeix) i carregant així el mapa *ForestMap* des de zero. Un cop carregat es treurà el *MainMenuWidget*. Veure Figura 415.

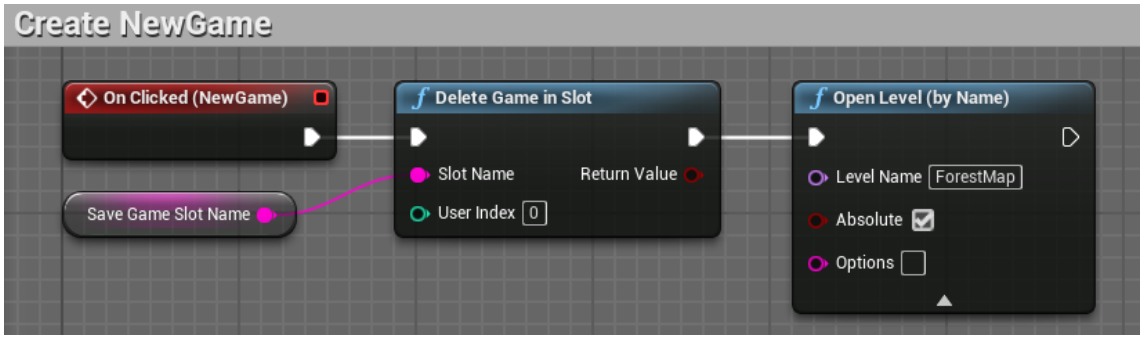

Figura 415. Event OnClicked del botó NewGame

6.17.3.2.6 Event LoadGame

Aquest *Event* s'executarà quan el jugador faci clic al botó de *Load Game*, carregant qualsevol partida existent (si existeix) i carregant així el mapa on es trobava el jugador al moment de guardar. Un cop carregat es traurà el *MainMenuWidget*. Veure Figura 416.

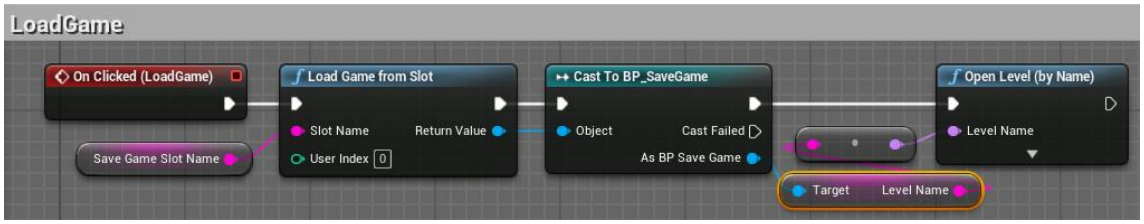

Figura 416. Event OnClicked del botó LoadGame

6.17.3.2.7 Construct

Quan es crea el *Widget*, es deixa l'índex a -1 per indicar que per defecte la navegació serà amb teclat i ratolí. A continuació s'emplenen les llistes d'*AllButtons*, *ButtonNormal* i *ButtonHover* amb els respectius valors, veure Figura 417.

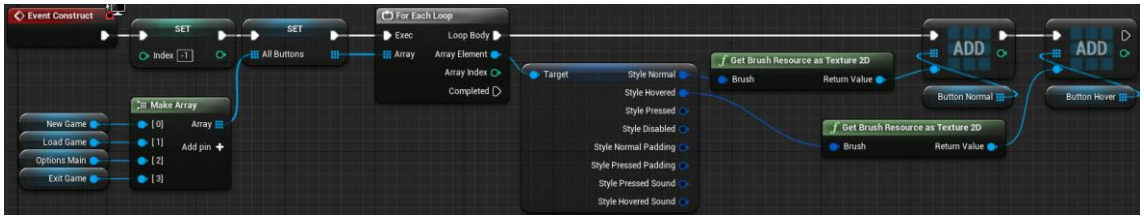

Figura 417. Construct al MainMenuWidget

6.17.3.2.8 Event NavigateUpDown

Primerament, cal saber que com el menú principal és un nivell diferent aquí no existeix el *ThirdPersonCharacter*. Per aquest motiu, tant aquest *Event* com l'explicat a la secció següent es criden des del *Blueprint* del nivell (explicat a l'Apartat 6.17.3.3), tal i com es pot apreciar a la Figura 418.

Figura 418. InputActions al MainMenuLevel Blueprint

Tenint en compte això, tota la navegació es gestiona des del propi *Widget*. Per aquest motiu es comprova si les opcions estan obertes per cridar a l'*Event NavigateUpDown*, esmentat a l'Apartat 6.17.5.6. Si no és el cas, comprova el valor de l'índex. Si val -1, es selecciona el primer botó amb *SelectActual*, veure Apartat 6.17.3.2.10. Si no, selecciona el següent valor tenint en compte si s'ha clicat la fletxa amunt o avall. Veure Figura 419.

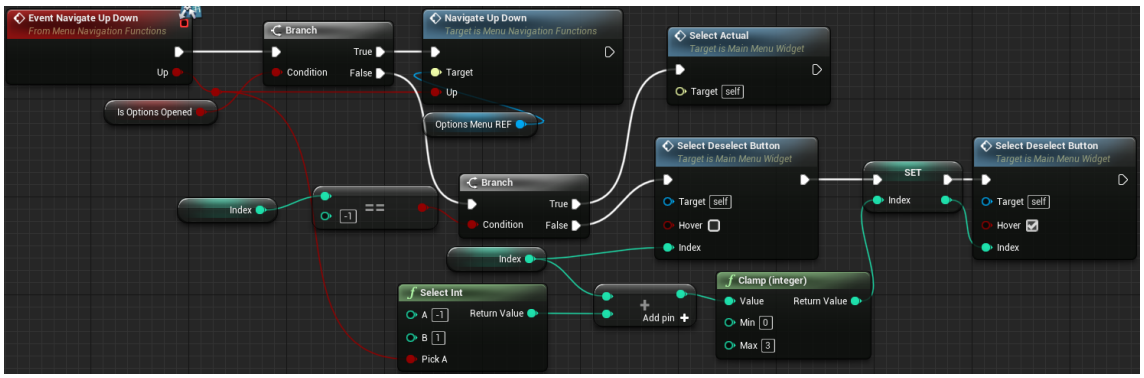

Figura 419.Event NavigateUpDown al MainMenuWidget

6.17.3.2.9 Event NavigateLeftRight

Ja que el menú principal té una llista de botons d'amunt a avall, només gestiona aquest *Event* si les opcions estan obertes. Per fer-ho, crida a l'*Event NavigateLeftRight*, explicat a l'Apartat 6.17.5.7. Veure Figura 420.

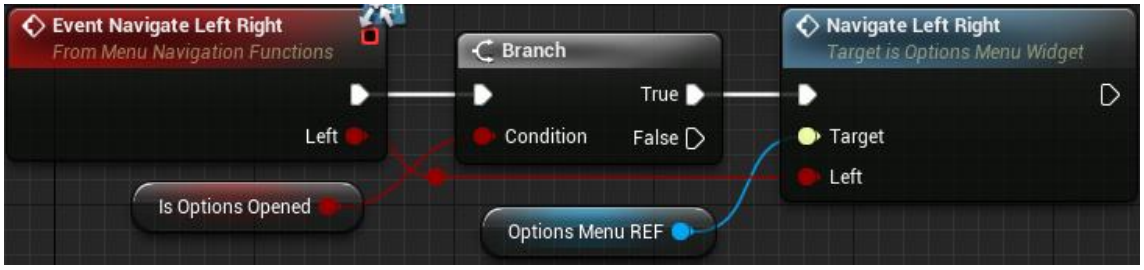

Figura 420. Event NavigateLeftRight al MainMenuWidget

6.17.3.2.10 SelectActual

En aquest cas, l'*Event* en qüestió deixa l'índex a zero i selecciona el primer botó amb *SelectDeselectButton*, desenvolupada a l'Apartat 6.17.3.2.13. Veure Figura 421.

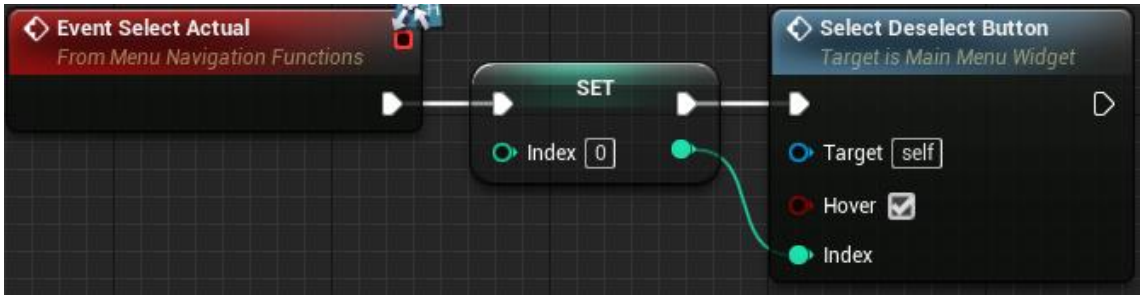

Figura 421. Event SelectActual al MainMenuWidget

6.17.3.2.11 DeselectActual

Quan es crida *DeselectActual*, es comprova el valor de l'índex per evitar accessos fora de rang. Si el seu valor és diferent de -1, és a dir, la navegació està en mode controlador, es crida a la funció *SelectDeselectButton*, esmentada a l'Apartat 6.17.3.2.13. Veure Figura 422.

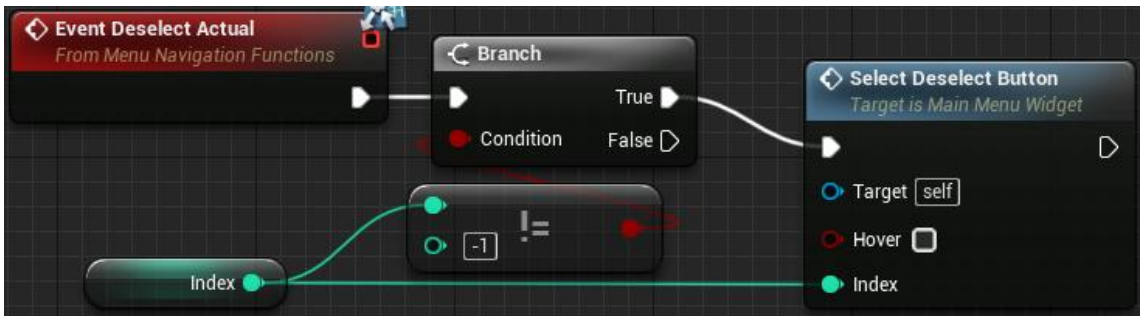

Figura 422. Event DeselectActual al MainMenuWidget

6.17.3.2.12 OptionsClosed

L'*Event OptionsClosed* es crida quan es tanca el menú d'opcions. Primerament es posa el booleà *IsOptionsOpened* a fals. De mateixa manera que amb l'apartat anterior, a continuació es comprova el valor de l'índex per saber si està dins del rang. Si és així, desselecciona el botó d'opcions amb *DeselectActual*, esmentat a l'Apartat 6.17.5.9 i selecciona el botó amb *SelectActual*, explicat a l'Apartat 6.17.3.2.10. Veure Figura 423.

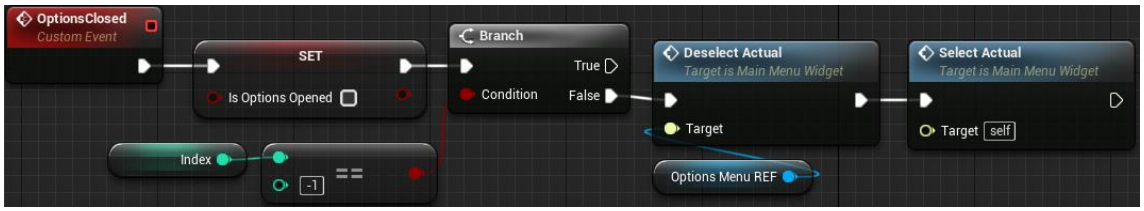

Figura 423. OptionsClosed

6.17.3.2.13 SelectDeselectButton

Aquesta funció s'utilitza a la navegació amb controlador i, rebent un índex i un booleà com a paràmetre, selecciona un botó i canvia l'estil a *Hover* o a Normal segons si *Hover* es cert o fals, respectivament. Veure Figura 424.

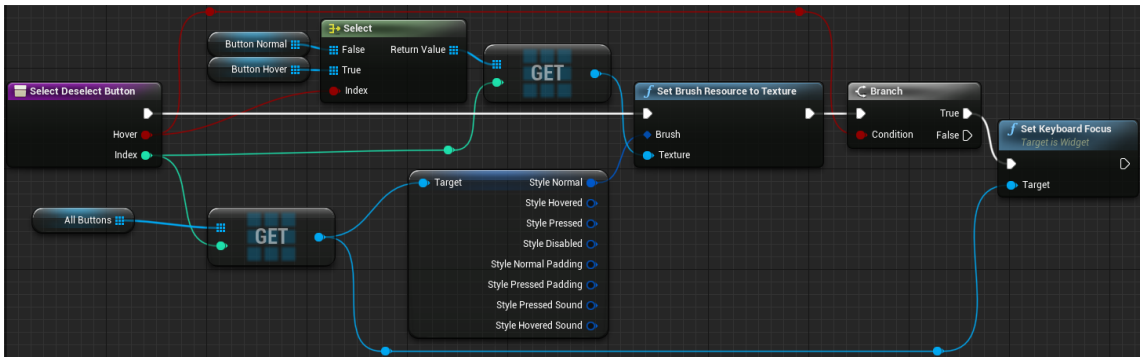

Figura 424. SelectDeselectButton

6.17.3.3 MainMenuLevel Blueprint

Aquest *Blueprint* s'executa a l'iniciar aquest nivell del joc i permet carregar el menú d'inici a la pantalla permetent així crear una partida, carregar una partida ja existent o modificar les opcions de joc.

6.17.3.3.1 Variables

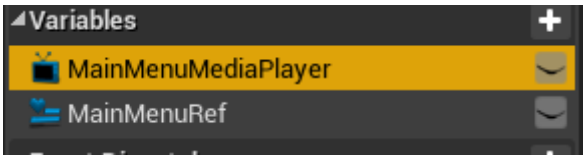

Figura 425. Variables del MainMenuLevel Blueprint

Com es pot veure a la Figura 425 hi ha dues variables, una variable *MediaPlayerObject* que conté el *MediaPlayer* creat a l'Apartat 6.17.3.2.1, i una referència al *Main Menu Widget* (Apartat 6.17.3.2).

6.17.3.3.2 Event BeginPlay

Aquest *Event* s'executarà quan el nivell es carrega per primer cop.

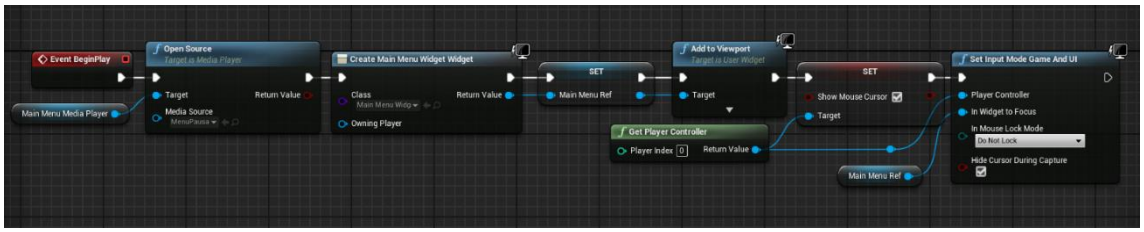

Figura 426. Event BeginPlay del MainMenuLevel Blueprint

Quan s'inicia el joc per primer cop, el que farà és, mitjançant la funció *Open Source*, obrirà l'arxiu FMS creat anteriorment (Apartat 6.17.3.2.1) per tal de poder reproduir l'animació, crearà el *Main Menu Widget*, l'afegirà al *Viewport* i ficarà el joc en mode *Game And UI* per poder navegar pels menús.

6.17.4 Menú de pausa

El menú de pausa té un únic component anomenat *MenuPausaWidget* que permet pausar l'execució del joc, modificar les opcions del joc i sortir del joc (Figura 31).

6.17.4.1 Variables

Com es pot veure a la Figura 427, tenim dues variables d'imatge (*Espadas*, que son les espases de la protagonista (Figura 28) ficades de manera únicament decorativa, i la imatge de fons *FondoMP* que es el fons animat, fet amb el mateix material que hem creat al punt 6.17.3.2.1), tres botons (*ExitGame*, *Options* i *ResumeGame*) i un objecte de tipus *MediaPlayerObject* que es diu *MediaPlayer* per poder controlar la reproducció de l'animació de fons.

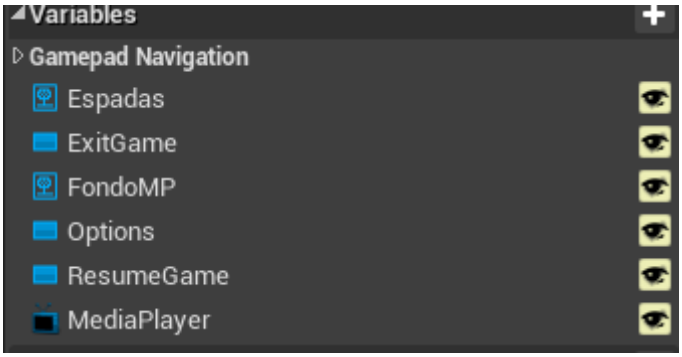

Figura 427. Variables del MenuPausaWidget

6.17.4.2 Construct

L'*Event Construct* es crida quan es crea per primer cop i, de la mateixa manera que a l'Apartat 6.17.3.3.2, es crida a la funció *Open Source* per obrir l'arxiu FMS de l'Apartat 6.17.3.2.1, i després crida a la funció *Initialize Arrays* explicat a l'Apartat 6.17.4.8. Veure Figura 428.

Figura 428. Event Construct del MenuPausaWidget

6.17.4.3 Open

Aquest *Event* té una única funció senzilla i és que cada cop que s'obre el menú de pausa tornar a reproduir des de l'inici l'animació del fons. Veure Figura 429.

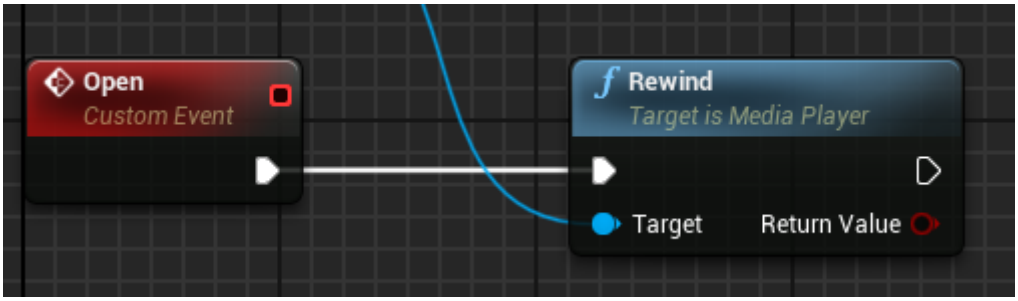

Figura 429. Event Open del MenuPausaWidget

6.17.4.4 Event ExitGame

A l'igual que a l'Apartat 6.17.3.2.3, quan s'executa aquest *Event* fent clic al botó de *Exit Game*, cridarà a la funció *Quit Game* que automàticament acabarà amb l'execució del joc. Veure Figura 430.

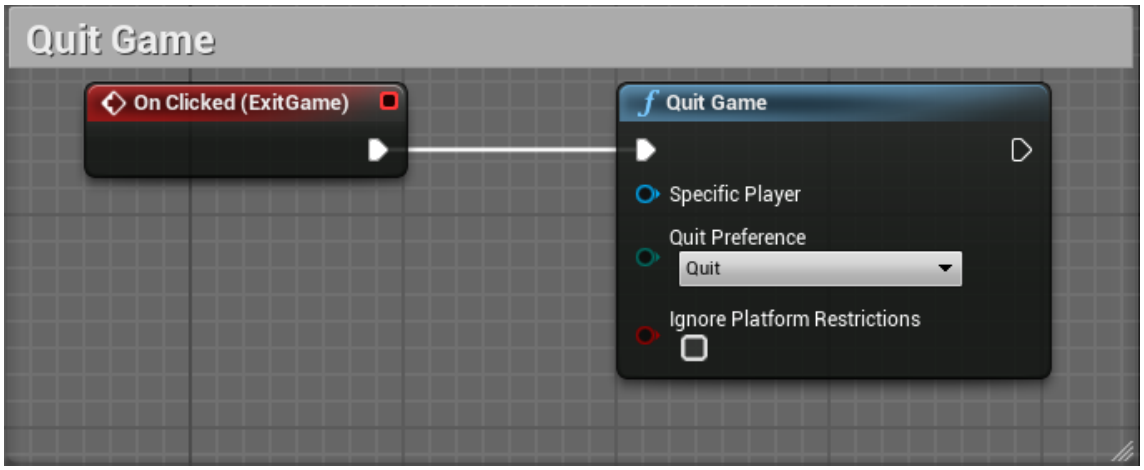

Figura 430. Execució de l'event OnClicked del botó "Exit Game" al MenuPausaWidget

6.17.4.5 Event ResumeGame

Aquest *Event* s'executa quan el jugador faci clic al botó *Resume Game*, cridant a la funció *Open or Close Menu* del *ThirdPersonCharacter* per tancar el menú de pausa i reprendre l'execució del joc. Veure Figura 431.

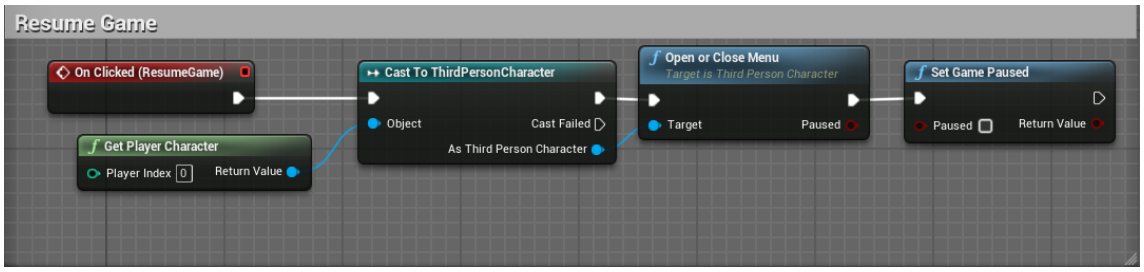

Figura 431. Execució de l'event OnClicked del botó "Resume Game" al MenuPausaWidget

6.17.4.6 Open or Close Menu del ThirdPersonCharacter

La funció *Open or Close Menu* és una funció associada al personatge protagonista que s'encarrega en qualsevol moment tant d'obrir com de tancar el menú de pausa.

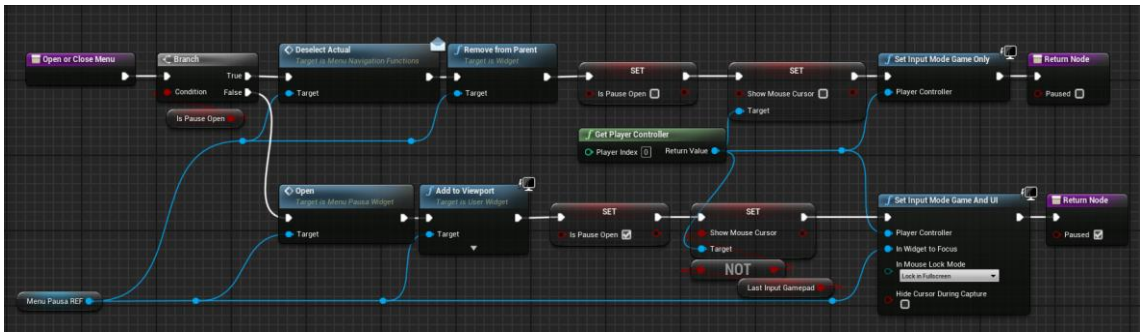

Figura 432. Execució de la funció "Open or Close Menu"

Com es pot veure a la Figura 432, hi ha dos fils d'execució a seguir, un per si el menú està ja obert i un altre per si el menú està tancat.

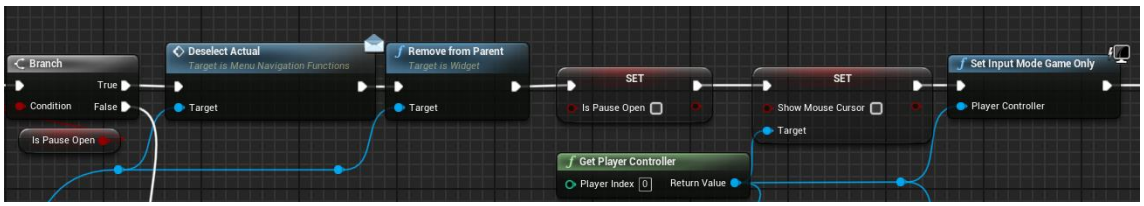

Figura 433. Execució de "Open or Close Menu" en cas de que el menú estigui ja obert

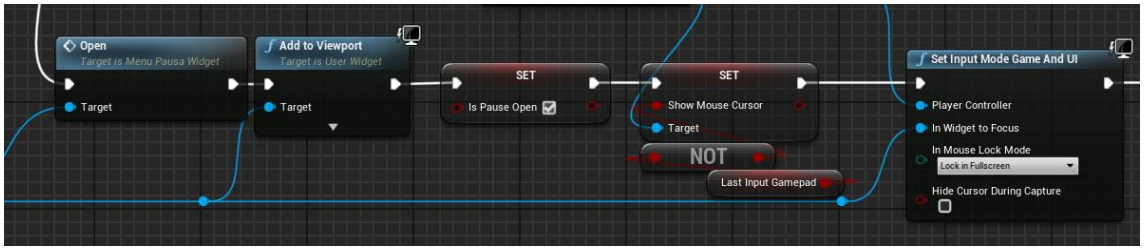

Figura 434. Execució de "Open or Close Menu" en cas de que el menú no estigui ja obert

En el cas en que el menú de pausa ja estigui obert (Figura 433) el que es farà és treure el menú de la pantalla, amagar el cursor, ficar a fals el booleà *Is Pause Open* i ficar els inputs en mode *Game only* per poder continuar amb el joc.

Per altra banda, si el menú no estava ja obert (Figura 434) el que es farà és cridar activar l'*Event Open* del *MenuPausaWidget* (Apartat 6.17.4.3), afegir a la pantalla el *MenuPausaWidget*, fer visible el cursor, ficar a cert el booleà *Is Pause Open* i els inputs en mode *Game and UI* per poder interactuar amb el menú.

6.17.4.7 Event Options

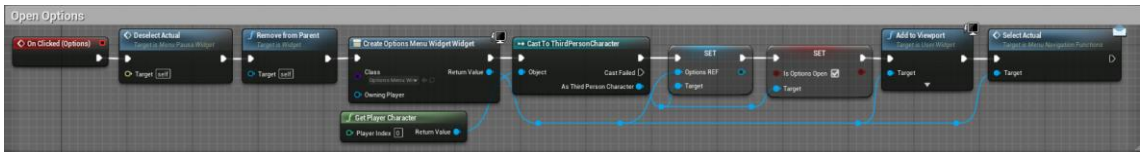

Figura 435. Execució de l'Event OnCLicked del botó "Options" al MenuPausaWidget

A la Figura 435 es pot veure com quan es clica al botó d'opcions es treu del *Viewport* el menú actual de pausa i es crea un nou *MenuOptionsWidget* i es fica al *Viewport*, canviant així de menú. Figura 435.Figura 435. Execució de l'Event OnCLicked del botó "Options" al MenuPausaWidget

6.17.4.8 InitializeArrays

Aquesta funció neteja totes les llistes, emplena la llista *AllButtons* i en fa un recorregut per obtenir la textura normal i *Hovered* per guardar-les a *NormalTextures* i *HoverTextures*, respectivament. Aquestes llistes s'utilitzen per seleccionar els botons a la navegació amb controlador. Veure Figura 436 i Figura 437.

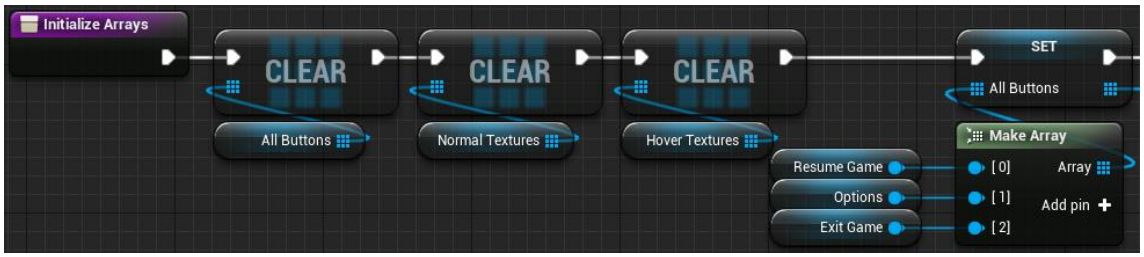

Figura 436. InitializeArrays - Part 1

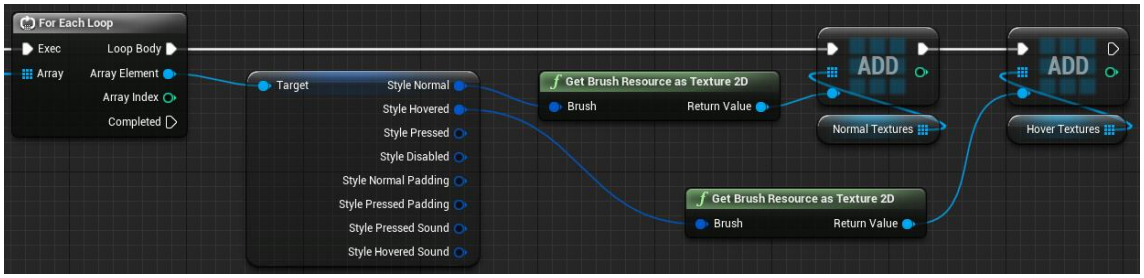

Figura 437. InitializeArrays - Part 2

6.17.4.9 NavigateUpDown

Aquest *Event* és una implementació de *MenuNavigationFunctions*, esmentada a l'Apartat 6.17.1. La seva funció és desseleccionar el botó amb *ChangeHoverStyle*, explicada a l'Apartat 6.17.4.12, incrementar o decrementar l'índex del botó seleccionat dins del límit i seleccionar-lo amb *SetKeyboardFocus* i *ChangeHoverStyle*. Veure Figura 438.

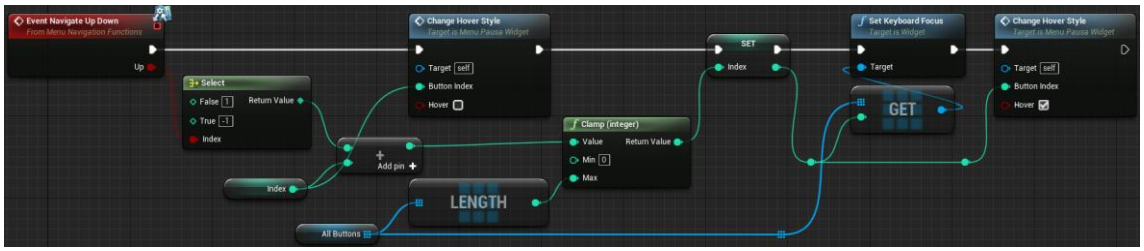

Figura 438. NavigateUpDown al menú de pausa

6.17.4.10 SelectActual

Aquest *Event* també és una implementació de *MenuNavigationFunctions*, desenvolupada a l'Apartat 6.17.1. Canvia l'índex a zero, per seleccionar el primer botó, i el selecciona amb *SetKeyboardFocus* i *ChangeHoverStyle*, esmentada a l'Apartat 6.17.4.12. Veure Figura 439.

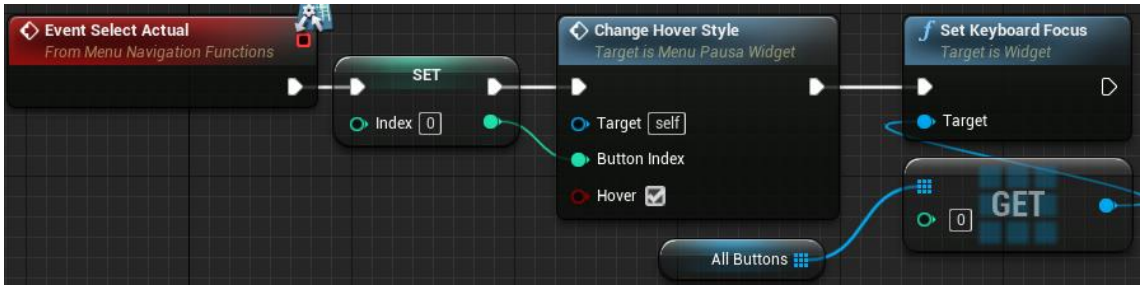

Figura 439. SelectActual

6.17.4.11 DeselectActual

DeselectActual forma part de les funcions de *MenuNavigationFunctions*, explicada a l'Apartat 6.17.1. S'utilitza quan es surt del menú de pausa i crida a *ChangeHoverStyle*, desenvolupada a l'Apartat 6.17.4.12, enviant l'índex actual. Veure Figura 440.

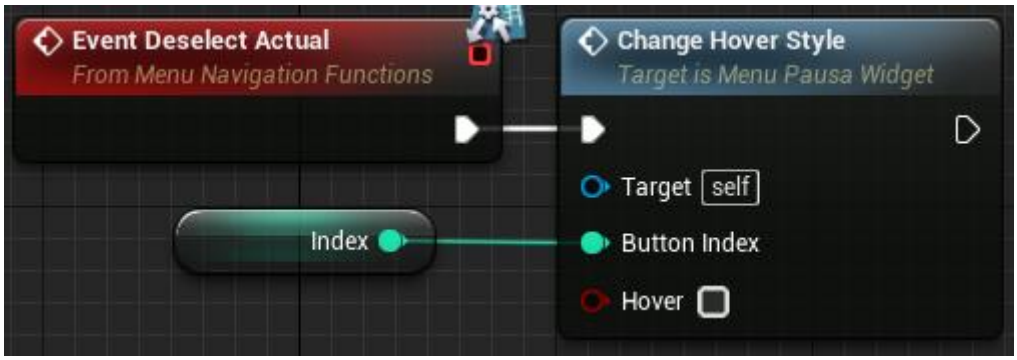

Figura 440. DeselectActual

6.17.4.12 ChangeHoverStyle

Aquesta funció rep com a paràmetre l'índex del botó per seleccionar-lo de la llista *AllButtons* i un booleà per saber si s'ha d'assignar la textura normal o *Hover*. Veure Figura 441.

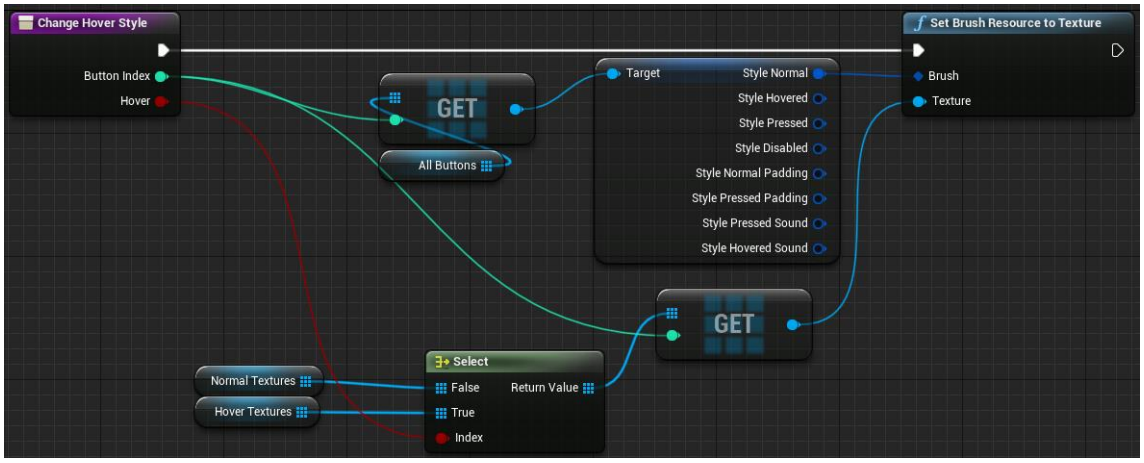

Figura 441. ChangeHoverStyle

6.17.5 Menú d'opcions

El menú d'opcions, d'igual manera que el menú de pausa de l'Apartat 6.17.4, està format per un únic *Widget* que permeten modificar la resolució i el volum general del joc. Veure Figura 30.

6.17.5.1 Construct

Aquesta funció inicialitza posant l'índex de les resolucions a zero i emplenant les llistes de *Resolutions*, *NormalTextures* i *HoverTextures* amb les dades corresponents. Veure Figura 442.

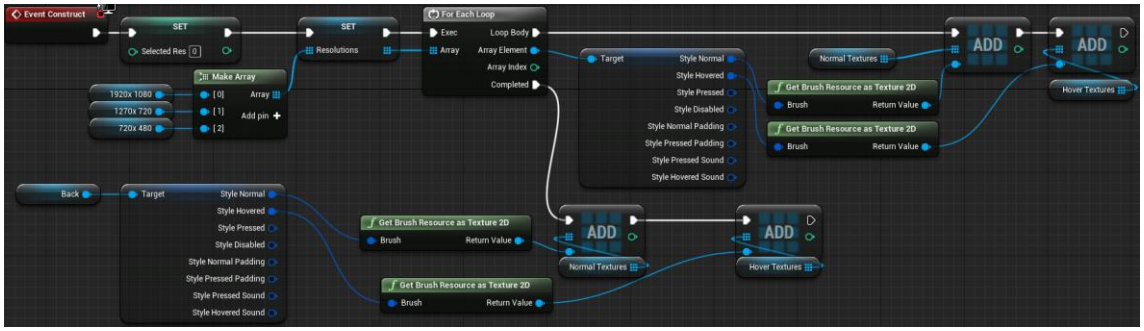

Figura 442. Construct

6.17.5.2 Canvi de resolució

Quan el jugador faci clic a sobre d'una de les opcions possibles de resolució (1920x1080, 1270x720 o 720x480), el joc automàticament s'ajustarà a la última resolució clicada mitjançant una comanda "r:setres" i la resolució que sigui. Veure Figura 443.

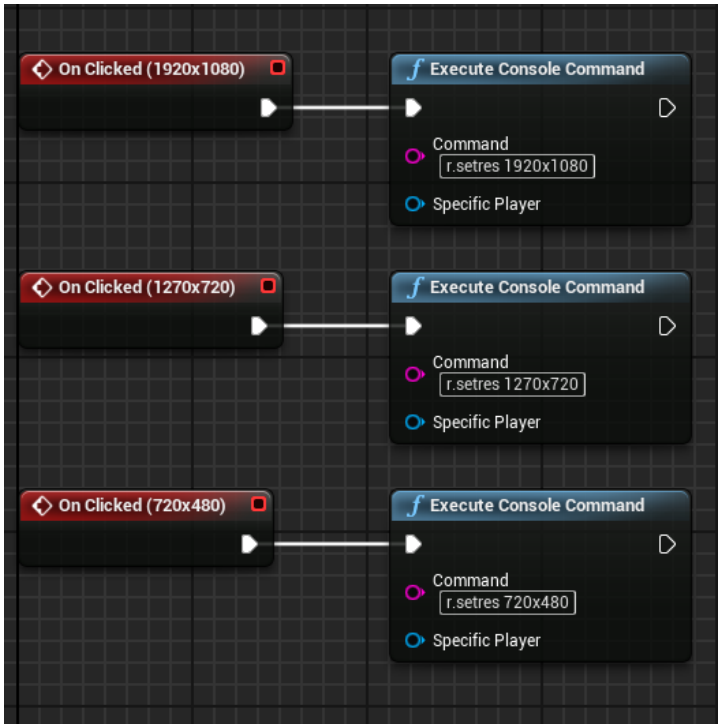

Figura 443. Com canvia mitjançant la comanda "r:setres " la resolució de pantalla

6.17.5.3 Canvi volum general

Quan es mou el *Slider* que hi ha com a opció reguladora de volum, automàticament es dispara l'*Event On Value Changed* que s'encarrega de regular el volum general del joc. Veure Figura 444.

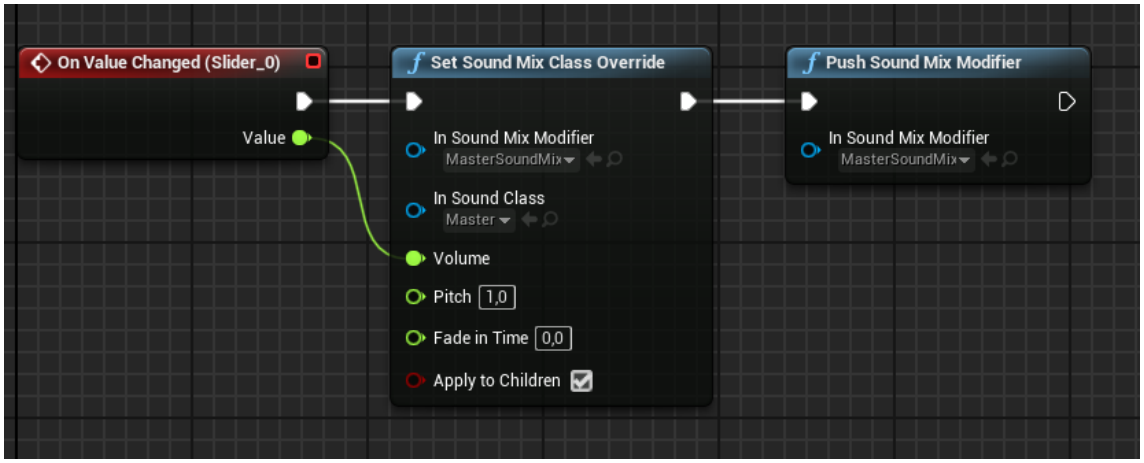

Figura 444. Quan es mou de posició el slider, el volum general varia

6.17.5.4 Event Back

Quan es fa clic a sobre del botó *Back* del menú, tanquem el menú d'opcions i aquí existeixen dues opcions, o bé es torna al joc, o es torna al menú principal, sempre en funció de des d'on s'ha obert el menú d'opcions.

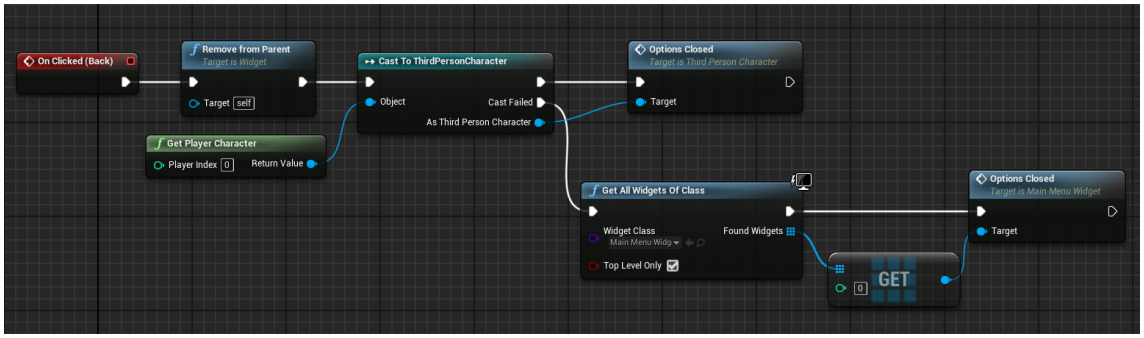

Figura 445. Execució de l'Event Back

Com es pot veure a la Figura 445, es fa un *cast* al *ThirdPersonCharacter* just després de treure el menú d'opcions, si aquest cas funciona correctament, es crida a l'*Event Options Closed* del *ThirdPersonCharacter*, si falla es crida al mateix *Event* però del menú d'inici.

6.17.5.5 Options Closed

Aquest *Event* es dispara just quan s'ha tancat el menú d'opcions i hi ha dos possibles execucions, una si s'ha obert des del menú de pausa, i l'altra des del menú d'inici.

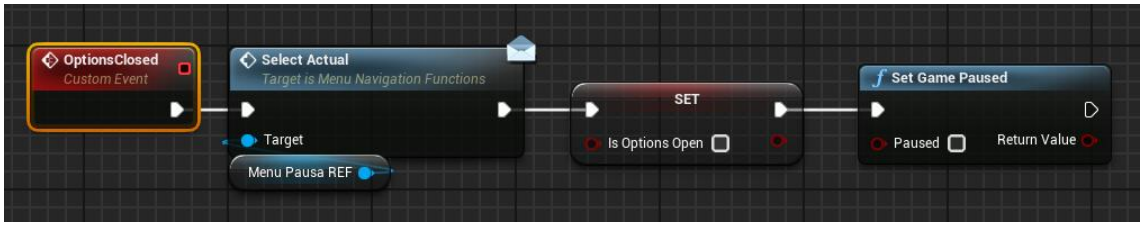

Figura 446. Execució de l'Event Options Closed des del ThirdPersonCharacter

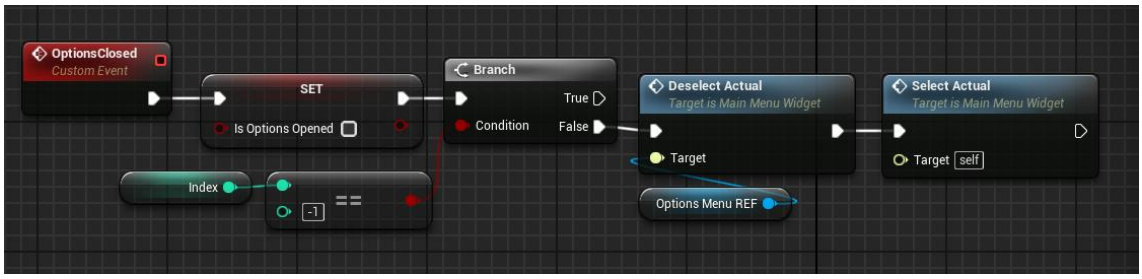

Figura 447. Execució de l'Event Options Closed des del menú d'inici.

La major diferencia que trobem és que a la Figura 446 un cop tret el menú d'opcions es reprèn l'execució normal del joc, mentre que a la Figura 447 es prepara la navegació per el canvi de menú.

6.17.5.6 NavigateUpDown

Aquest *Event* implementa una de les funcions de *MenuNavigationFunctions*, explicada a l'Apartat 6.17.1. Deselecciona l'objecte actual amb *SelectDeselectObject*, desenvolupada a l'Apartat 6.17.5.10, després canvia el valor d'índex i selecciona el nou objecte amb *SelectDeselectObject*. Veure Figura 448.

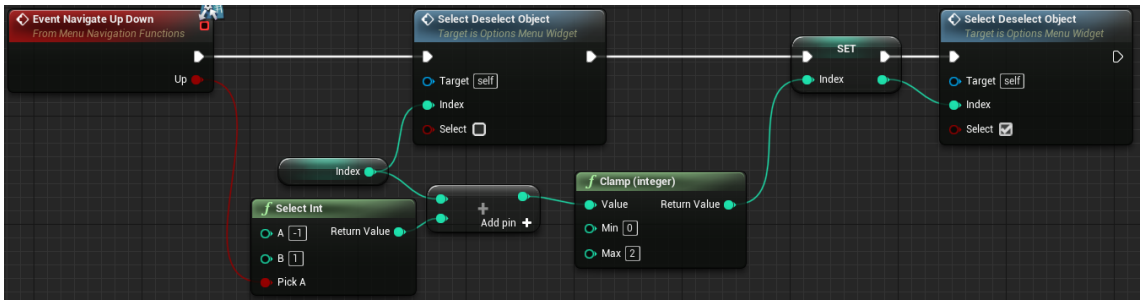

Figura 448. NavigateUpDown

6.17.5.7 NavigateLeftRight

L'*Event NavigateLeftRight* implementa una funció de *MenuNavigationFunctions*, esmentada a l'Apartat 6.17.1. Utilitza un *Switch* per determinar si es troba sobre els botons de resolucions o la barra de so. En el cas del so, canvia el valor del *Slider* amb *SetValue*. En l'altre cas, realitza la mateixa tasca que l'apartat anterior, però utilitzant l'índex de les resolucions i cridant a la funció *ChangeButtonStyle*, explicada a l'Apartat 6.17.5.11. Veure Figura 449.

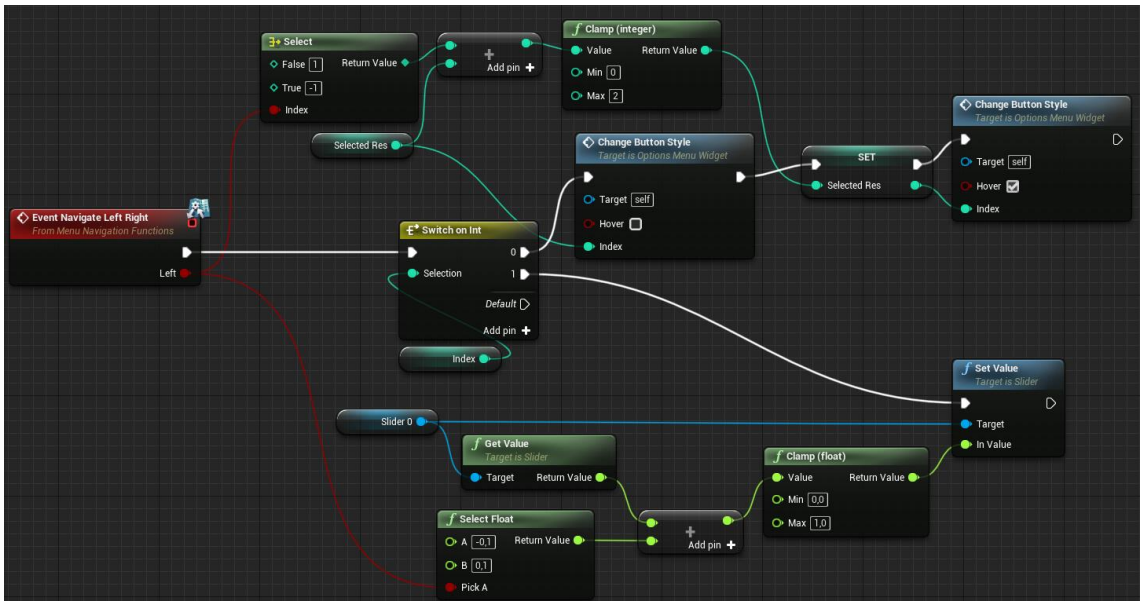

Figura 449. NavigateLeftRight

6.17.5.8 SelectActual

Aquest *Event* també implementa *MenuNavigationFunctions*, desenvolupada a l'Apartat 6.17.1. Posa l'índex a zero i selecciona l'objecte corresponent amb *SelectDeselectObject*, explicada a l'Apartat 6.17.5.10. Veure Figura 450.

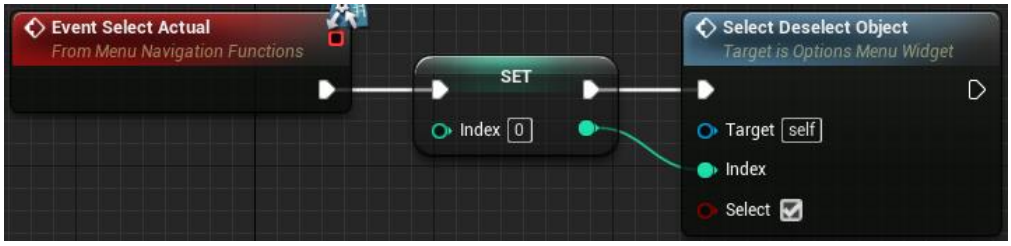

Figura 450. SelectActual

6.17.5.9 DeselectActual

DeselectActual és la implementació d'una de les funcions de *MenuNavigationFunctions*, esmentada a l'Apartat 6.17.1. En aquest cas es crida la funció *SelectDeselectObject*, explicada al següent apartat, amb l'índex actual per desseleccionar l'objecte. Veure Figura 451.

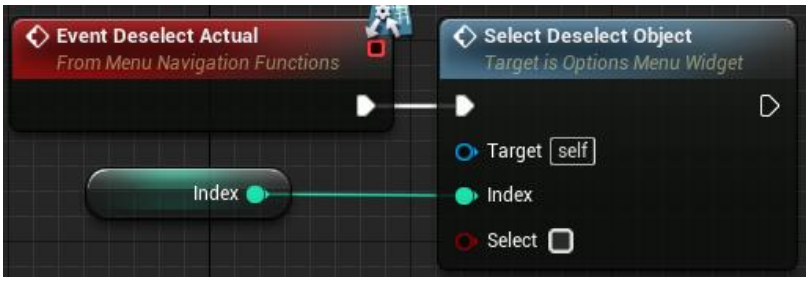

Figura 451. DeselectActual

6.17.5.10 SelectDeselectObject

Aquesta funció rep com a paràmetre un índex i un booleà. Utilitza un *Switch* per saber si s'han de seleccionar els botons de resolució, la barra de so o el botó de *Back*. Pels botons crida a la funció *ChangeButtonStyle*, explicada al següent apartat, i per la barra utilitza *ChangeBarStyle*, esmentada a l'Apartat 6.17.5.12. Veure Figura 452.

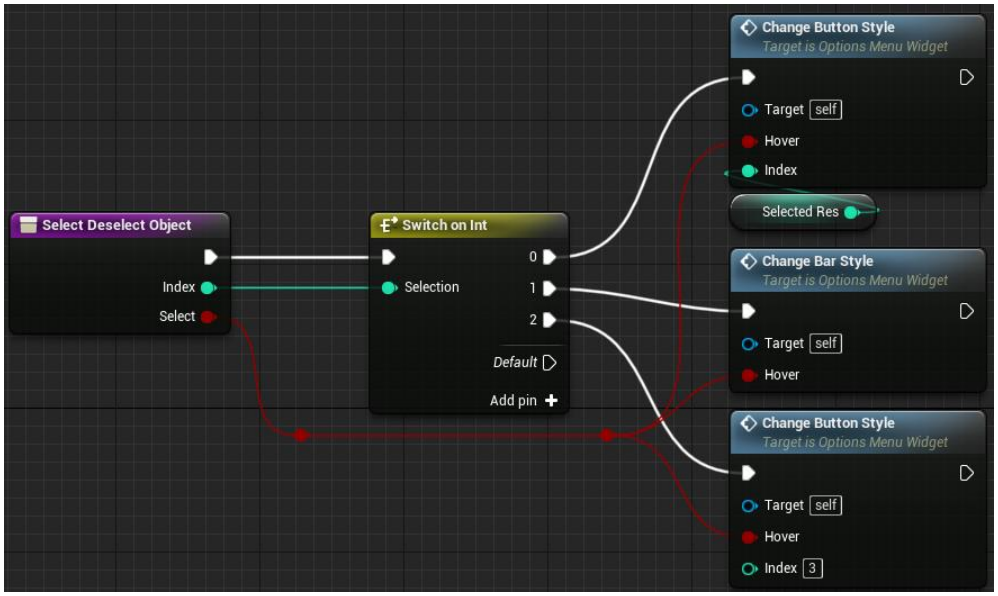

Figura 452. SelectDeselectObject

6.17.5.11 ChangeButtonStyle

Aquesta funció rep un índex com a paràmetre per saber quin botó s'ha de seleccionar i un booleà per determinar si s'ha d'utilitzar l'estil normal o *Hover*. Actualitza l'estil i, si *Hover* és cert, posa el focus amb *SetKeyboardFocus*. Figura 453.

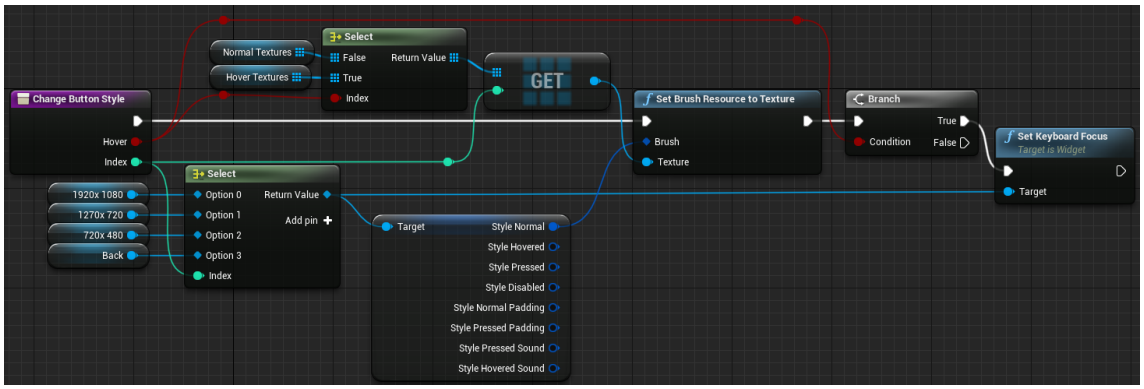

Figura 453. ChangeButtonStyle

6.17.5.12 ChangeBarStyle

La funció *ChangeBarStyle* rep com a paràmetre un booleà per decidir el color de la barra. Obté l'estil de la barra de so i en canvia el color, rosat pel cert i vermell pel fals. A més, posa el focus en el *Widget* del menú d'opcions. Veure Figura 454.

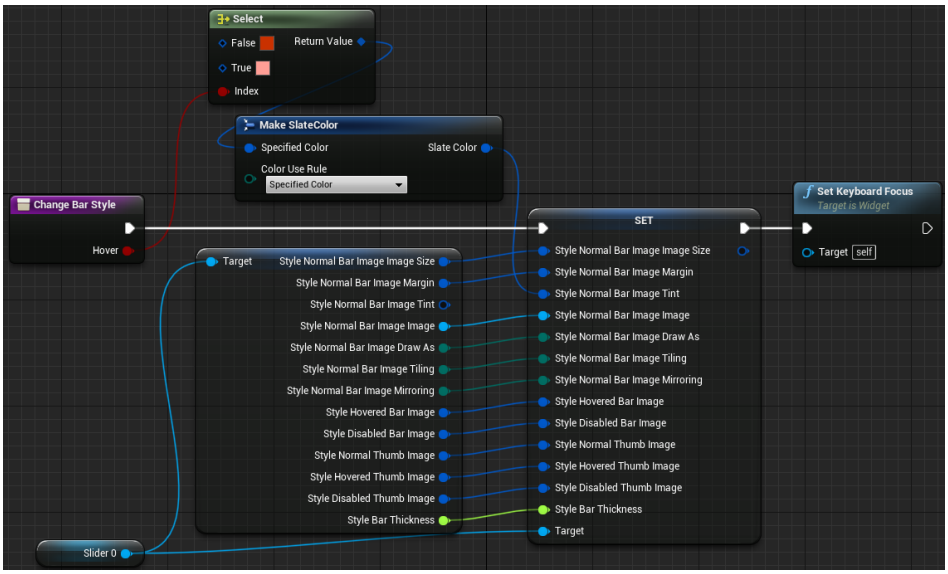

Figura 454. ChangeBarStyle

6.17.6 Mort del Jugador (Interfície)

Quan el jugador mor durant la partida, es mostrarà la interfície de la Figura 47 just abans de tornar al menú d'inici. Aquesta interfície es troba dins un *Widget* anomenat *DiedWidget*.

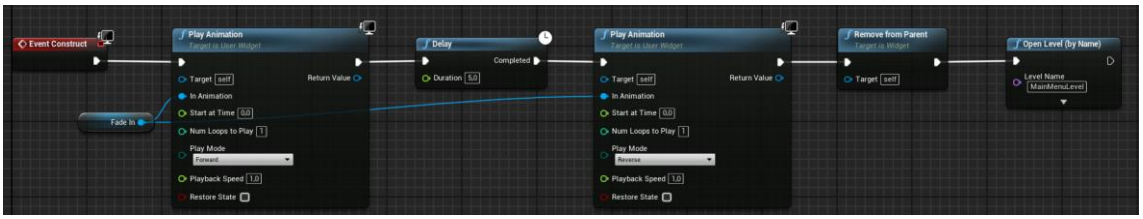

Figura 455. Execució del Widget DiedWidget

Com es pot veure a la Figura 455, a l'hora de construir aquest *Widget* s'executa una animació *Fade In* i després es queda en espera durant 5 segons, abans de tornar a executar aquesta animació però a la inversa i treure aquest W*idget* de la pantalla i tornar al menú principal.

6.17.7 Crèdits finals

Quan el jugador acaba amb l'enemic final, es mostrarà per pantalla la interfície de la Figura 48 abans de tornar al menú d'inici. Aquesta interfície es troba dins un *Widget* anomenat *CreditsWidget.*

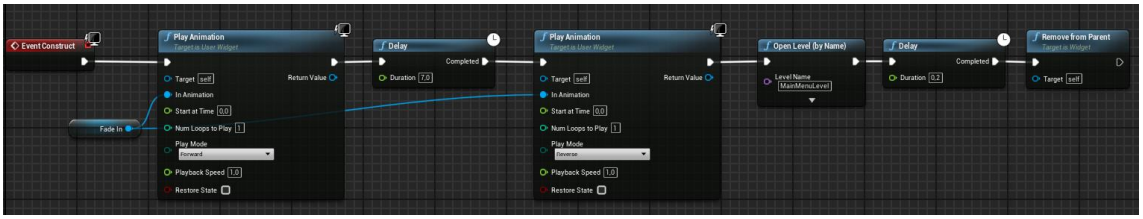

Figura 456. Execució del Widget CreditsWidget

Com es pot veure a la Figura 456, la manera d'implementar-ho és quasi idèntica a l'Apartat 6.17.6 amb la petita diferència que l'espera és més gran per poder llegir millor el text.

7 Resultats

7.1 Legislació i normativa vigent

El projecte presentat no presenta cap tipus de requeriment legal. El joc no guarda cap informació personal del jugador, ni interactua amb la informació dels seus equips. Per tant, no aplica la Llei Orgànica de Protecció de Dades (LOPD). Tampoc té cap objectiu de caràcter econòmic, de forma que tampoc aplica la Llei de Serveis de la Societat de la Informació i Comerç Electrònic (LSSICE). A més a més, no infringeix cap llei de drets d'autor, ja que qualsevol recurs extret d'Internet s'ha cerciorat que tingués drets de Copyright. Això és així en el cas de no comercialitzar el projecte. En cas contrari, s'hauria de pagar una llicència per la música de fons del mapa del Bosc, del creador *BrunuhVille*. Per acabar, si el projecte comercialitzat arribés a uns beneficis d'un milió de dòlars, s'haurien de pagar uns *royalties* a *Epic Games* del 5% dels beneficis, per l'ús del motor de videojocs, UE4.

7.2 Classificació PEGI

Pel què fa a la classificació per edats estàndard PEGI de videojocs cal dir que no és obligatòria. Tot i així, la classificació del projecte presentat seria PEGI 16. Els motius d'aquesta classificació és la presencia de violència i llenguatge incorrecte com a constant, però sense arribar als límits que marquen la línia del PEGI 18. A més, caldria afegir les etiquetes de "Violència", "Llenguatge groller" i "Discriminació". Les dues primeres ja s'ha comentat el perquè, i l'etiqueta de discriminació vindria associada al rebuig que hi ha entre les espècies. Tot i posar l'etiqueta de PEGI 16 en el projecte presentat, si aquest seguís endavant, la quantitat de violència i la seva ferocitat augmentarien, així que passaria a ser PEGI 18.

7.3 Resultat Final

A continuació es poden observar algunes captures del joc, que s'han extret del vídeo del següent enllaç: https://youtu.be/AB-2393vQ1w

Figura 457. Menú principal

Figura 458. Menú de pausa

Figura 459. Menú d'opcions

Figura 460. Aria a l'inici del joc

Figura 461. Aria interactuant amb l'estàtua

Figura 462. Escenari del poble

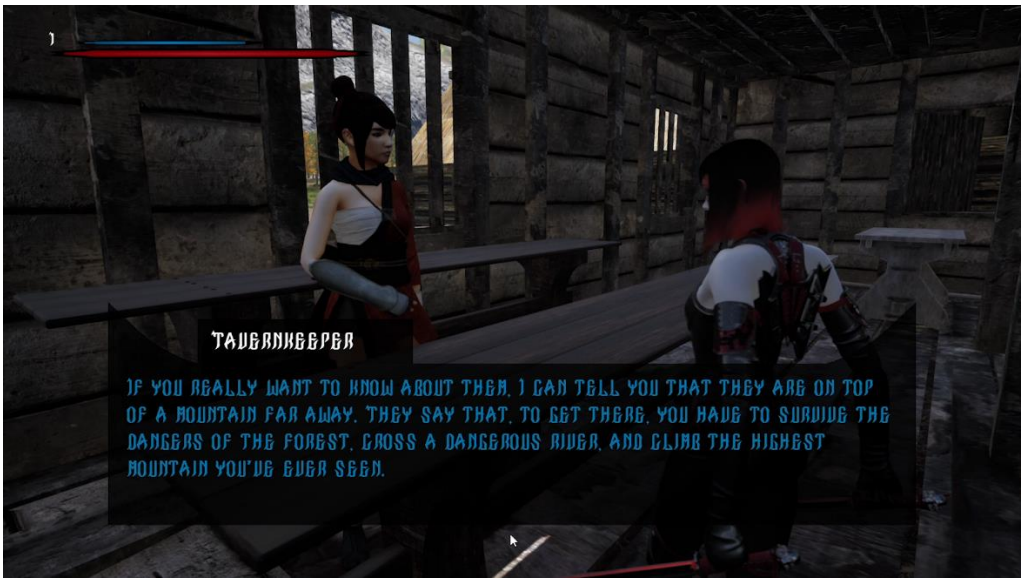

Figura 463. Aria interactuant amb la tavernera

Figura 464. Aria parlant amb l'àngel

Figura 465. Aria obtenint una runa d'un cofre

Figura 466. Animació d'atac contra un minion

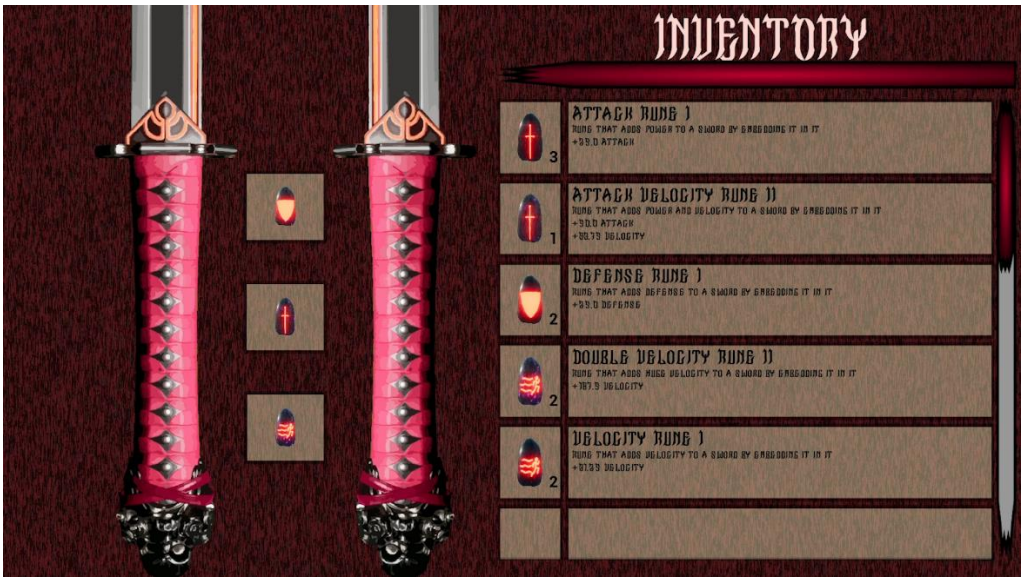

Figura 467. Inventari amb runes incrustades

Figura 468. Entrada al santuari

Figura 469. Minijoc plataformes

Figura 470. Zona del mapa propera al final

Figura 471. Accés a la lluita final

Figura 472. Lluita contra l'enemic final (atac cos a cos)

Figura 473. Lluita contra l'enemic final (atac a distància)

Figura 474. Pantalla de derrota

Figura 475. Pantalla de victòria

8 Conclusions

En general, estem molt satisfets del resultat que hem obtingut amb la implementació d'aquest prototipus. Hem estat capaços de crear un producte assolint la majoria dels objectius proposats i, a més, creiem que és un joc molt vistós gràficament.

Aquest projecte ens ha estat de gran ajuda, ja que hem pogut treballar amb aspectes del motor que no havíem treballat mai, com un sistema de diàlegs, gestió d'inventari o la creació de la intel·ligència artificial d'un enemic final per fases. A més, hem estat capaços de treballar els tres conjuntament utilitzant l'eina de *Github* sense tenir grans problemes, més enllà d'algun conflicte que vam poder arreglar fàcilment. També estem molt orgullosos del disseny del mapa, ja que considerem que, si el jugador té ganes de gaudir de l'experiència del joc al complet, el mapa actual convida a explorar i superar els reptes (com per exemple la zona amb núvols a mode de plataforma) per aconseguir runes de millor qualitat.

El producte té uns gràfics atractius i unes animacions que quan es veuen per primer cop creiem que poden impressionar i captivar al jugador. Si bé és així, també cal esmentar que el sistema de combat no és tan refinat com ens agradaria i li faria falta més varietat d'enemics per afegir un nivell extra de complexitat. Tampoc ha estat possible crear un sistema de fortaleses i debilitats amb els atacs elementals, que donarien un punt d'estratègia interessant al combat. A més, no es va implementar el sistema de missions i els mapes tenen menys zones de les que es parlen al disseny del joc. Si bé aquest últim fet va ser una decisió que es va prendre des d'un bon inici, altres aspectes negatius que s'han esmentat com el combat o les missions no han estat possibles de realitzar, ja que tots tres integrants del grup hem estat l'últim any treballant a mitja jornada, a part dels estudis. A més, durant l'estiu hem passat a treballar a jornada completa. Aquest fet ha estat el principal motiu pel qual no hem pogut invertir tantes hores com ens hagués agradat al projecte.

9 Treball Futur

Com a possibles treballs futurs a desenvolupar, hi ha una gran part en les possibilitats de millora i afegits secundaris per al joc.

Per començar, es podria implementar els danys elementals. També es podria afegir un sistema de vulnerabilitats, on el jugador pugui fer més mal als enemics si se'ls hi ataca amb aquell element a què són vulnerables.

A més, com a afegit al sistema de diàlegs, es podria afegir un sistema de missions. Aquestes s'obtindrien a través d'una extensió dels diàlegs ja implementats, i es podria crear un nou sistema per a indicar al jugador quin objectiu té, o quines missions té disponibles.

Pel que fa al mapa, aquest es podria ampliar, o modificar per afegir molts més detalls. També es podria afegir algun tipus de mapa per guiar al jugador, ja fos a través del ja existent, o els nous.

Per part dels enemics, es podrien afegir enemics diferents. Uns de molt necessaris serien els enemics voladors, per a poder donar més valor al combat a l'aire.

Un sistema de comerç tampoc li aniria malament al joc. Aquesta era una idea inicial que no ha estat possible plasmar a la *Demo*, i que seria un afegit important. A causa d'aquest sistema, es podrien afegir molts més objectes útils. Cal afegir, que a conseqüència d'aquests nous objectes, s'hauria de canviar l'inventari, o fer un de nou complementari al ja existent.

Les idees donades són algunes de les moltes possibilitats de millora o extensió del joc, ja que, tot i considerar que té moltes mecàniques diferents implementades, el joc podria créixer sense problema en moltes direccions diferents.

10 Bibliografia

Adobe Systems, Inc. (s.d.). *Mixamo*. https://www.mixamo.com

Beardgames. (s.d.). *Canal de YouTube "Beardgames"*. https://www.youtube.com/@Beardgames

BibliaOn. (s.d.). *12 noms de Déu i que signifiquen*. https://www.bibliaon.com/es/nombres_de_dios_significado/

Cronza. (s.d.). *Canal de YouTube "Cronza"*. https://www.youtube.com/@Cronza

Cynthia. (s.d.). *Canal de YouTube "cynshin08"*. https://www.youtube.com/@cynshin088

Epic Games, Inc. (s.d.). *Documentació oficial d'Unreal Engine 4.27*. https://docs.unrealengine.com/4.27/en-US/

Epic Games, Inc. (s.d.). *Unreal Engine Forum*. https://forums.unrealengine.com/categories?tag=unreal-engine

Laley, R. (s.d.). *Canal de Youtube "Ryan Laley"*. https://www.youtube.com/@RyanLaley

Meza, H. (s.d.). *Canal de YouTube "HoracioMeza"*. https://www.youtube.com/@HoracioMeza

Microsoft. (s.d.). *Coneix el teu controlador sense fils Xbox One*. https://support.xbox.com/es-ES/help/hardware-network/controller/xbox-one-wireless-controller

Reids. (s.d.). *Canal de YouTube "ReidsChannel"*. https://www.youtube.com/@ReidsChannel

Seredias. (s.d.). *Canal de YouTube "Seredias"*. https://www.youtube.com/@Seredias.

Shutterstock. (s.d.). *TurboSquid - Runestones 3D model.* https://www.turbosquid.com/3d-models/runestones-baking-3d-model-1474826

Wastein, M. (s.d.). *Canal de YouTube "MathewWadsteinTutorials"*. https://www.youtube.com/@MathewWadsteinTutorials

WeMakeTheGame. (s.d.). *Portal d'animacions de "WeMakeTheGame"*. https://www.wemakethegame.com/

11 Annexos

11.1 Projecte

Per tal d'accedir al codi del prototip, es pot utilitzar l'adreça del repositori que s'ha utilitzat durant el desenvolupament del projecte. Cal descarregar el codi en un fitxer *.zip* des del botó *Code* de la pàgina, descomprimir l'arxiu i obrir el projecte amb la versió d'UE 4.27. Adreça del projecte: https://github.com/DanielVazSam/TFG

11.2 Diàlegs

A continuació es mostren els diàlegs implementats dins el projecte. Primer hi ha el nom de qui parla i, després, el diàleg. On hi posi *Reply*, significa que s'entra en una de les possibles branques del diàleg.

11.2.1 Statue

- [Statue] MY CHILD.
- [Statue] THERE ARE FEW WHO DARE APPROACH ME.
- [Statue] I DON'T KNOW IF I'M MORE SURPRISED BY YOUR COURAGE OR YOUR AUDACITY.
- [Statue] YOUR HEART SCREAMS FOR VENGEANCE.
- [Statue] I WILL HELP YOU WITH YOUR MISSION BY HEALING YOUR PAIN.
- [Statue] MAKE YOUR GOD PROUD OF YOU.

11.2.2 Blacksmith

- [ARIA] Hi.
- [Blacksmith] Hello. How can I help you?
- [ARIA] (Reply 1) I don't need anything.
- [ARIA] (Reply 2) I'm looking for information on how to get to the gates of heaven.
    - [ARIA] Do you know anything about it?
    - [Blacksmith] HAHAHAHA. Reach the gates of heaven, you say? That's a dangerous undertaking, my friend. I don't have much idea, but you can ask at the tavern.
    - [ARIA] Thank you.

11.2.3 Villager

- [ARIA] Excuse me, sir. Could you tell me a little bit about this town?
- [Villager] Ah, yes. This is a humble place, my friend. We are simple people, without luxuries or riches.
- [ARIA] I see the roads are pretty empty. Aren't there many people living here?
- [Villager] You see, many people have died in recent years. Diseases have taken many lives. But those of us who remain manage as best we can.
- [ARIA] I understand. How do you survive in a place like this?
- [Villager] Some of us are farmers, others work in the nearby mines. But don't expect to find riches here. Most of us barely survive.
- [ARIA] Is there any danger around here?

- [Villager] Well, there are bandits and outlaws on the roads, and wolves are also a problem. Also, sometimes demons and other evil creatures appear in these places.
- [ARIA] Demons, you say. Have you seen any?
- [Villager] Not personally, thank God. But I have heard stories of people who have had encounters with them. They say they come from hell, looking for souls to take with them.
- [ARIA] Interesting... By the way, I'm looking for the way to heaven, do you know anything about it?
- [Villager] Heaven, huh. Well, I don't think it exists, but I've heard it's to the north. But be careful, the road is dangerous. There are creatures that will attack you on the way, and the so-called angels are not usually very friendly to mortals.
- [ARIA] I will keep this in mind. Thank you very much for the information.

11.2.4 Tavernkeeper

- [Tavernkeeper] What can I do for you?
- [ARIA] (Reply 1) A beer, please.
    - [Tavernkeeper] Of course. Here you go.
    - Something else?
    - [ARIA] (Reply 1) No, thank you.
        - (Exits the conversation)
    - [ARIA] (Reply 2) Yes.
        - [Tavernkeeper] Tell me.
        - *(Restart the conversation)*
- [ARIA] (Reply 2) I am looking for information.
    - [ARIA] Do you know anything about the gates of heaven?
    - [Tavernkeeper] The gates of heaven! That's something people talk about when they're drunk.
    - [Tavernkeeper] If you really want to know about them, I can tell you that they are on top of a mountain far away. They say that, to get there, you have to survive the dangers of the forest, cross a dangerous river, and climb the highest mountain you've ever seen.
    - [Tavernkeeper] I don't know how much of this is true...
    - [Tavernkeeper] But be careful, my friend. Many have tried and few have returned.
    - [ARIA] Thank you, I will take it into account.
    - [Tavernkeeper] Something else?
    - [ARIA] (Reply 1) No, thank you.
        - *(Exits the conversation)*
    - [ARIA] (Reply 2) Yes.
        - [Tavernkeeper] Tell me.
        - *(Restart the conversation)*

11.2.5 Runes Expert

- [Runes Expert] I am the runes expert of this little village. Do you want to know something?
- [ARIA] (Reply 1) What is a rune?
    - [Runes Expert] A rune is an object that you can embed in your sword.
    - [Runes Expert] Once embedded, it will grant you different types of attributes, to a greater or lesser degree, depending on its type and rarity.
    - [Runes Expert] A maximum number of runes can be embedded in each sword. Your sword, for example, can be embedded with up to 3 runes, which is the maximum possible.
    - [Runes Expert] But it is not only the capacity of the sword that matters, but also that of the wielder. As you train, you will be able to carry more embedded runes.
    - *(Restart the conversation)*
- [ARIA] (Reply 2) Are there different types of runes?
    - [Runes Expert] Yes, that's right.
    - [Runes Expert] There are three different categories: Common, Epic and Legendary.
    - [Runes Expert] In addition to these 3 categories, there are power-ups for 3 types of attributes: Attack, Defense and Velocity.
    - [Runes Expert] The Common category is the simplest one. It boosts a single attribute, and does so in a weak form.
    - [Runes Expert] The Epic category can boost one or two different attributes. Each attribute is boosted with the same strength. If only one attribute is boosted, the improvement will be double.
    - [Runes Expert] The Legendary category is the best without any doubt. It always boosts 2 attributes in a very powerful way. The main attribute will be boosted to a higher rank than the secondary one. In addition, they have a unique ability: they inflict elemental damage. The elements can be Fire, Electric and Critical.
    - [Runes Expert] This Legendary Runes are unique, and you only can have one of each. There exist 6 of them. I highly recommend you to look for them. They will be of great help to you.
    - [Runes Expert] Right behind me, I have left one Common Rune of each type. Just for having listened to me, I'll give them to you. Pick them up if you haven't already. Use them wisely.
    - *(Restart the conversation)*
- [ARIA] (Reply 3) No, thank you.
    - [Runes Expert] Come anytime you need my help.

11.2.6 Angel

- [ARIA] What are you doing here, you dirty angel?
- [Angel] How did you discover me?

- [ARIA] You think you're the only one capable of camouflaging yourself among mortals? I've already seen many of your kind. What brings you to this hole?
- [Angel] Maybe it is a hole, but it's where I live.
- [ARIA] What? What are you talking about?
- [Angel] What I just said: I live here.
- [ARIA] Impossible.
- [Angel] It is not, and I am not the only one. Humans, angels and demons live here. Even beasts, some believe.
- [Angel] Everyone has their reasons for hiding here. Mine is that, despite being an angel, I never accepted the abominable rejection of you demons. Losing the Great War was no reason to banish you or change your name: You are as much angels as I am.
- [Angel] My ideas did not please the other angels, and they repeatedly tried to kill me. In the end, I had to leave too. Not for safety, I could have changed my words, but for my ideals. I would never live in a place with such sad ideas...

11.2.7 Monk

- [Monk] Welcome to my sanctuary.
- [Monk] This is a sacred place since immemorial times.
- [Monk] Here lie power and wisdom.
- [Monk] This is a place of calm. Self-control. Self-awareness.
- [Monk] This is a place where wholeness, and mental and physical strength have always been worked on.
- [Monk] I see that you have a hard assignment.
- [Monk] Since you have come this far, I will give you the Rune of Wisdom, better known as Flame Determination.
- [Monk] I have left it right behind me. Pick it up if you haven't already.
- [Monk] Best of luck on your journey.

11.2.8 Traveller

- [Traveller] WOW!
- [Traveller] How did you get here?
- [Traveller] I thought I was the only person capable of getting here. I see that's not the case.
- [Traveller] I am a traveller. I love discovering every little corner of the world. I see that you love to travel too. Otherwise, what would have brought you here?
- [ARIA] Yes, well... Sort of...
- [ARIA] Anyway.
- [ARIA] I'm looking for some magic runes. Do you know where there are any?
- [Traveller] Yes. I already have one.
- [ARIA] Wha-
- [Traveller] But I don't want it at all. Mmm... I'll give it to you.
- [ARIA] Seriously?!
- [Traveller] Yes, I am. The truth is, I really like you. I wish there were more explorers.

- [ARIA] Well... Thank you very much.
- [Traveller] I left it in that chest in the back. Take it if you haven't already.

11.3 Controls

Quan es treballa amb controls de teclat resulta molt senzill entendre què s'ha de posar ja que cada lletra es refereix a cada tecla. A més, totes les tecles especials tenen un nom únic i general. Pel cas del controlador és una mica diferent, ja que n'hi ha de molts tipus i cadascú fa servir diferents noms. Per aquest motiu cada motor utilitza noms propis, que s'intenta que siguin genèrics. A la Figura 476 es poden veure els noms que s'utilitzen pel controlador, agafant com a exemple un controlador d'*Xbox*.

Figura 476. Mapping dels botons per un controlador d'Xbox a UE4

12 Manual d'Usuari i d'Instal·lació

12.1 Manual d'Usuari

Per jugar hi ha dos maneres: teclat amb ratolí o controlador. A continuació hi ha una llista amb totes les accions que es poden realitzar i el botó corresponent per teclat i controlador, respectivament (en aquest cas s'ha agafat com a exemple el controlador d'una *Xbox One*):

- Moure: WASD, *Joystick* esquerre
- Càmera: ratolí, *Joystick* dret
- Saltar: espai, B
- *Dash*: SHIFT, RT
- Atac fluix: clic esquerre, X
- Atac fort: clic dret, Y
- Inventari: I, botó Menú
- Menú: ESC, botó Vista
- Marcar enemic: T, botó del *Joystick* dret
- Interactuar: E, A

12.2 Manual d'Instal·lació

Per tal de jugar a la *demo* cal accedir al següent *link*: https://sergiochinchilla.itch.io/arias-ascent. A continuació s'ha de clicar al botó de *Download Now*. Finalment només cal descomprimir el *.zip* i obrir l'executable anomenat "Aria's Ascent". En cas que el botó no funcioni, es pot descarregar a través de l'enllaç a *Google Drive* proporcionat a la mateixa pàgina.