Izrada trodimenzionalne igre u programskom alatu Unreal Engine

Romić, Borna

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Borna Romić

IZRADA TRODIMENZIONALNE IGRE U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Borna Romić

Matični broj: 45857/07–R

Studij: Informacijski sustavi

IZRADA TRODIMENZIONALNE IGRE U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, svibanj 2021.

Borna Romić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Završni rad pokriva proces izrade 3D videoigre koja se igra iz trećeg lica u programskom alatu Unreal Engine 4 korištenjem nacrt *(eng. Blueprint)* načina programiranja.

Unreal Engine 4 korištenjem nacrta *(eng. Blueprint)* omogućava brz, učinkovit i kvalitetan način programiranja videoigre te je idealan odabir za kreiranje osnovne videoigre za početnike, a i za naprednije programere.

Cilj završnog rada je prikazati proces i tok razmišljanja programiranja videoigre iz gledišta početničkog game developera.

Žanr videoigre je Hack and Slash s elementima žanra Metroidvanije u smislu težine te interaktivne i intrigantno postavljene mape u kojoj se igrač kreče.

Cilj videoigre je pobijediti sve protivnike kako bi se došlo do konačnog neprijatelja čijom pobjedom igrica je uspješno završena.

Ključne riječi: Unreal Engine 4, nacrt *(eng. Blueprint)*, videoigra, animacija, funkcija, model *(eng. Mesh)*, widget

Sadržaj

1. Uvod

Videoigra je posebna vrsta softvera koji radi na hardveru, računalu ili konzoli za videoigre. Ta hardverska platforma zahtijeva barem nešto memorije (koja može biti u nekoliko oblika), neki procesorski kapacitet i načine interakcije sa zaslonom te neku metodu s kojom igrač može kontrolirati igru.

Industrija videoigara 2020. godine imala je veću zaradu od industrije filmova i Sjeverno Američkog sporta zajedno [1], činjenica da je industrija videoigara jedna od brže rastućih industrija, ali i osobnog užitka i brojnih videoigara koje sam odigrao su samo jedni od brojnih faktora zašto sam uzeo temu za završni rad upravo izradu videoigre.

Glavni cilj ovog završnog rada je prikazati postupak, tijek i način razmišljanja tijekom programiranja videoigre iz stava početnika.

2. Metode i tehnike rada

Kako bi proveli postupak kreiranja videoigre potrebno je odabrati pogon igre (*eng. Game engine*) u kojem ćemo programirati i realizirati ideju zamišljene videoigre.

Pogon igre koji sam odabrao za korištenje u ovom završnom radu je Unreal Engine 4. Unreal Engine je pogon igre, koji je razvio Epic Games Inc, prvi put predstavljen u pucačkoj igri iz 1998. godine po nazivu Unreal. Otada se koristi u raznim žanrovima trodimenzionalnih (3D) igara, Unreal Engine napisan je na jeziku C++, odlikuje se visokim stupnjem prenosivosti, podržavajući širok raspon platformi za stolna računala, mobilne uređaje, konzole i virtualne stvarnosti. Unreal Engine se koristi u puno modernih AAA videoigara kao što je Epic-ova vlastita battle royale pucačina Fortnite ili jedna od vizualno najimpresivnijih igara 2017. godine Hellblade: Senua's Sacrifice. [2]

Programiranje u Unreal Engine 4 može se odvijati korištenjem C++ programskog jezika za izradu vlastitih skripti koje se pokreću u samom pogonu igre. Alternativa standardnom načinu programiranje je korištenje nacrta (*eng. blueprint)*, koji su vrlo moćni gotovi blokovi koda koji se dodaju objektima za interakciju. Ujedno, igrica napravljena u ovom završnom radu izrađena je u potpunosti koristeći Unreal Engine 4 i njegove nacrte, odnosno, Blueprint Visual Scripting sustav.

„Blueprint Visual Scripting sustav u Unreal Engineu je cjelovit sustav skriptiranja igranja zasnovan na konceptu upotrebe sučelja temeljenog na čvorovima za stvaranje elemenata igranja iz Unreal Editora. Kao i kod mnogih uobičajenih skriptnih jezika, koristi se za definiranje objektno orijentiranih (OO) klasa ili objekata u stroju.“ [3]

Ovaj sustav je izuzetno fleksibilan i moćan jer pruža mogućnost dizajnerima da koriste gotovo čitav niz koncepata i alata koji su općenito dostupni samo programerima te je svojim jednostavnim i intuitivnim dizajnom idealan odabir za početnike koji žele programirati svoju prvu videoigru.

Pri izradi videoigre veliki utjecaj i pomoć oko učenja svih potrebnih materijala za izradu igre imao je tečaj s Teachable-a [4] te brojni Youtube videi.

3. Razrada teme

Kako bi započeli sa izradom trodimenzionalne igre u programskom alatu Unreal Engine 4 potrebno je imati takozvanu prototip ideju kakvu videoigru želimo napraviti. Prvobitne ideje su sklone mnogim promjenama tijekom programiranja videoigre, ali je krucijalno imati neku viziju videoigre koju želimo napraviti.

Budući da su videoigre oblikovale veliki dio mojeg života, kada sam se uputio u programiranje svoje prve videoigre, imao sam već čvrsto definiran cilj kakvu videoigru želim napraviti.

3.1. Inspiracija za videoigru

Od svih žanrova igara, oni koji su me najčvršće držali i koje sam najviše igrao bili su „Souls-like“ video igre, odnosno, videoigre napravljene nalik popularnim Souls igrama, koje je napravio From Software.

Souls igre:

- Demon's Souls
- Dark Souls
- Dark Souls 2
- Dark Souls 3
- Bloodborne
- Sekiro

Glavne značajke Souls igara je velika međusobno povezana karta svijeta koju igrač može istražiti, pristup dijelovima svijeta bio bi često ograničen vratima ili kojekakvim drugim preprekama, koje se mogu proći tek kad igrač u igri nabavi potrebne predmete za prelaženje tih prepreka. Stjecanje novih predmeta pomaže igraču u pobjedi težih neprijatelja i pronalaženju prečaca i tajnih područja. S ovakvom vizijom videoigre From Software je napravio čvrstu integraciju dizajna priče i razine te fluidnu kontrolu likova kako bi se potaknulo istraživanje i eksperimentiranje tako da igrač više investira u svoj karakter igrača.

Jedna od više poznatih značajka Souls igara je izuzetno zahtjevna težina igre, mali broj ljudi koji započne igrati Souls igre uspije ih završiti, ali to je samo jedan od čari ovih videoigara.

Inspiriran From Software-ovom vizijom videoigara bio sam čvrsto odlučan u kojem smjeru želim programirati svoju prvu videoigru, a to je štoviše nalik Souls igrama u smislu glavnih funkcionalnosti i načinu igranja videoigre.

3.2. Vizija i cilj videoigre

Pri prvom ulasku u videoigru glavni lik će biti bez oružja, štita i šljema. Kako bi se prešlo igricu potrebno je doći do konačnog neprijatelja i pobijediti ga, no put do glavnog neprijatelja je zaključan iza sporednih neprijatelja, odnosno trollova koje glavni lik mora ubiti kako bi mu se otvorio put do glavnog neprijatelja.

Videoigra je teška te zahtjeva strpljenje i pažnju od igrača kako bi se uspješno prešla, igrač mora uvijek biti opremljen mačem ako se planira boriti protiv neprijatelja, osim mača također je idealno imati štit zbog sposobnosti zaustavljanja neprijateljskih napada i šljem radi dodatnih životnih bodova.

Mačevi, štitovi i šljemovi su raspoređeni po karti u ograničenoj količini te svaki predmet koji igrač pokupi te zatim umre gubi se iz njegovog imanja i mora otići pronaći nove predmete za borbu protiv trollova, ukoliko igrač umre s zadnjim mačem u svojem imanju videoigra završava s „Game Over“.

Ukoliko igrač uspije pobijediti desetog trolla dolazi mu obavijest kako može pristupiti glavnom neprijatelju i pokušati ga pobijediti kako bi prešao igricu. Nakon pobjede nad glavnim neprijateljem igrač dobiva obavijest da je uspješno pobijedio videoigru te se time završava videoigra.

3.3. Funkcionalnosti videoigre

Funkcionalnosti videoigre su jako inspirirane funkcionalnostima koje koriste sve Souls videoigre, a to su:

1. Odabir modela glavnog lika/neprijatelja i ostalih predmeta
2. Kontrola glavnog lika
3. Uzimanje predmeta s poda i opremanje glavnog lika
4. Uzrokovanje štete i statistika glavnog lika
5. Osposobljavanje glavnih mehanika
6. Osposobljavanje neprijatelja i njegove umjetne inteligencije
7. Kreiranje vanjskog svijeta za videoigru
8. „Targeting system“
9. Interakcija igrača i neprijatelja sa svijetom
10. Glavni izbornik

Kako bi moja vizija videoigre mogla se uopće smatrati videoigrom, vrlo je važno da se realiziraju sve navedene funkcionalnosti videoigre.

3.3.1. Odabir modela glavnog lika/neprijatelja i ostalih predmeta

Modeli i animacije su srž svake videoigre, bez dobrih modela i animacija videoigra nema težine i gubi bilo kakav učinak koji je htjela postići, stoga je vrlo bitno za videoigru da ima korektne modele i animacije s obzirom na žanr i što igrica želi postići.

Kako bi pronašli modele i animacije koje želimo koristiti u svojoj videoigri Unreal Engine nam omogućuje pronalaženje raznoraznih modela i animacija preko Unreal Engine Marketplace stranice. Svi modeli koje koristim u videoigri su besplatni ili su preuzeti za vrijeme dok su se dijelili za besplatno.

3.3.1.1. Model glavnog lika

Idealan model glavnog lika biti će Paragon Greystone iz Epic Games-ove igrice Paragon.

Slika 1: Paragon Greystone

Model lika Greystone postao je dostupan sa svim animacijama i predmetima zajedno sa svim likovima i nivoima iz videoigre Paragon na Unreal Engine Marketplaceu za besplatno nakon što je videoigra Paragon ugašena te je Epic Games Inc svu imovinu vezanu za videoigru objavio na Unreal Engine Marketplace. [5]

Modeli svih likova i nivoa su izuzetno kvalitetni, no za završni rad iz videoigre Paragon koristit ćemo samo model Greystonea.

Jedan nedostatak korištenja Paragonovih modela je što je Greystone zajedno sa svim svojim predmetima već spojen u model (eng. Mesh) Greystonea, no za to rješenje nalazimo u programu Blender.

Blender je besplatan profesionalni alat otvorenog koda za 3D računalnu grafiku koji se koristi za izradu animiranih filmova, vizualnih efekata, modela za 3D printere, interaktivnog sadržaja i sličnog na operativnim sustavima. [6]

Slika 2: Paragon Greystone u Blenderu

Pomoću Blendera odvajamo model (eng. Mesh) lika na 3 dijela: „Greystone_Tough“, „Shield“ i „Sword“ te ih tak odvojene uvozimo u naš Unreal Engine 4 projekt tako da imamo odvojen model Greystonea, njegovog mača i štita, a šljem je već odvojen sam po sebi.

Slika 3: Blender odvajanje modela

U projektu Unreal Engine 4 kreiramo novi nacrt koji ćemo nazvati „BP_HeroCharacter“ te unutar tog nacrta uvozimo model (eng. Mesh) Greystonea kojeg smo pomoću blendera odvojili.

Slika 4: BP_HeroCharacter

U nacrtu „BP_HeroCharacter“ dalje ćemo programirati sve potrebne funkcionalnosti i mehanike pomoću kojih ćemo upravljati našeg glavnog lika.

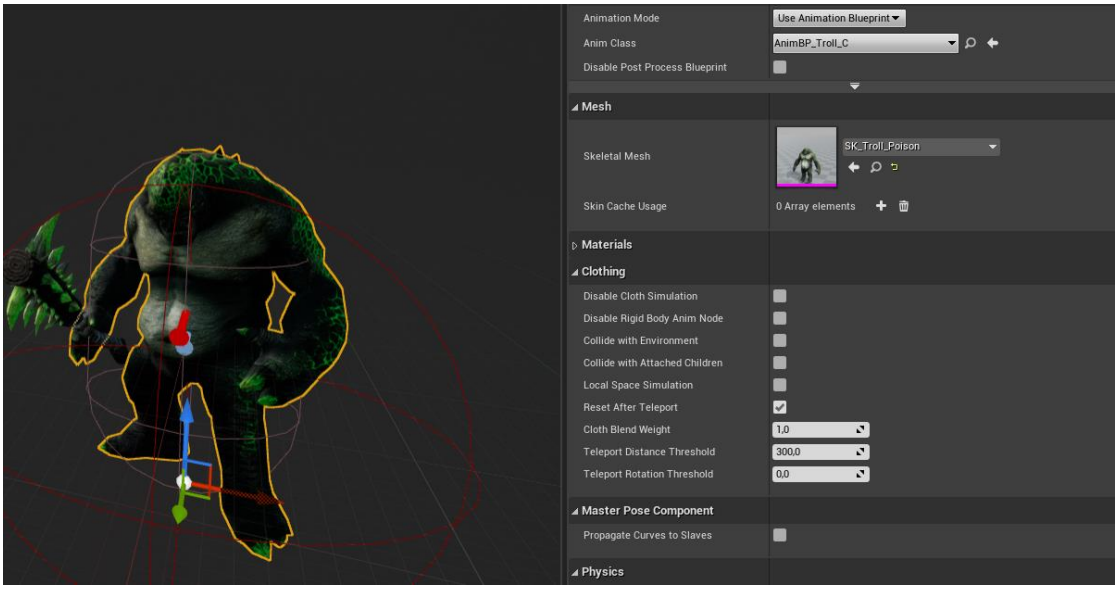
3.3.1.2. Model Greystone-ovih predmeta

Kao što smo prije naveli, naš glavni lik koristi mač, štit i šljem tijekom igre i zato što želimo napraviti da ih igrač može pokupiti moramo te predmete tretirat kao odvojene modele (eng. Mesh) i nacrt sa svojim funkcionalnostima.

Slika 5: Mapa „Pickups“

3.3.1.3. Model trolla

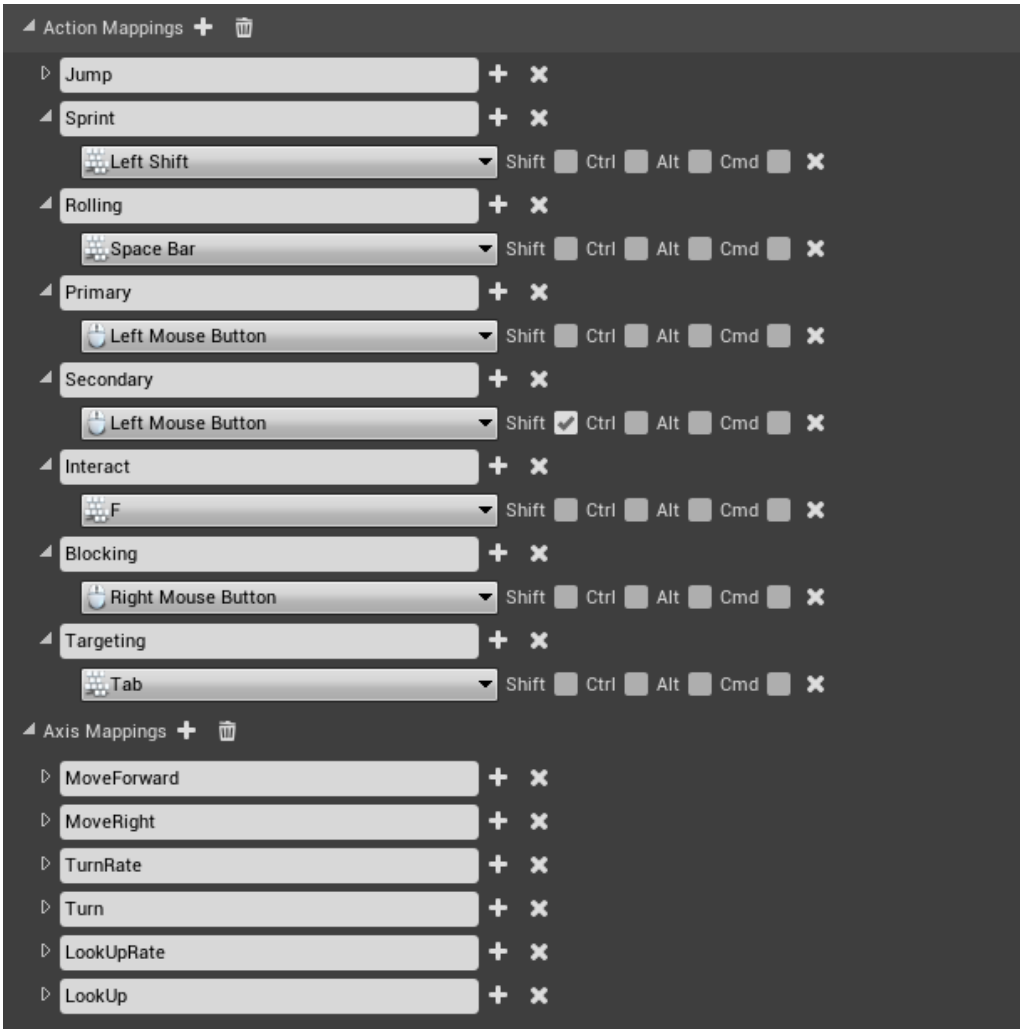
Model trolla preuzimamo također s Unreal Engine Marketplace-a ovaj put od „Infinity Blade: Adversaries“ [7], također jedne Epic Games Inc videoigre za koju su odlučili sve modele staviti na Unreal Engine Marketplace. Iz „Infinity Blade: Adversaries“ za model našeg trolla uzimamo model i animacije iz mape „Enemy_Troll“ te kreiramo u svojoj Troll mapi „Enemy_Troll“ nacrt u kojem ćemo dalje programirati sve funkcionalnosti za trolla.



Slika 6: Enemy_Troll

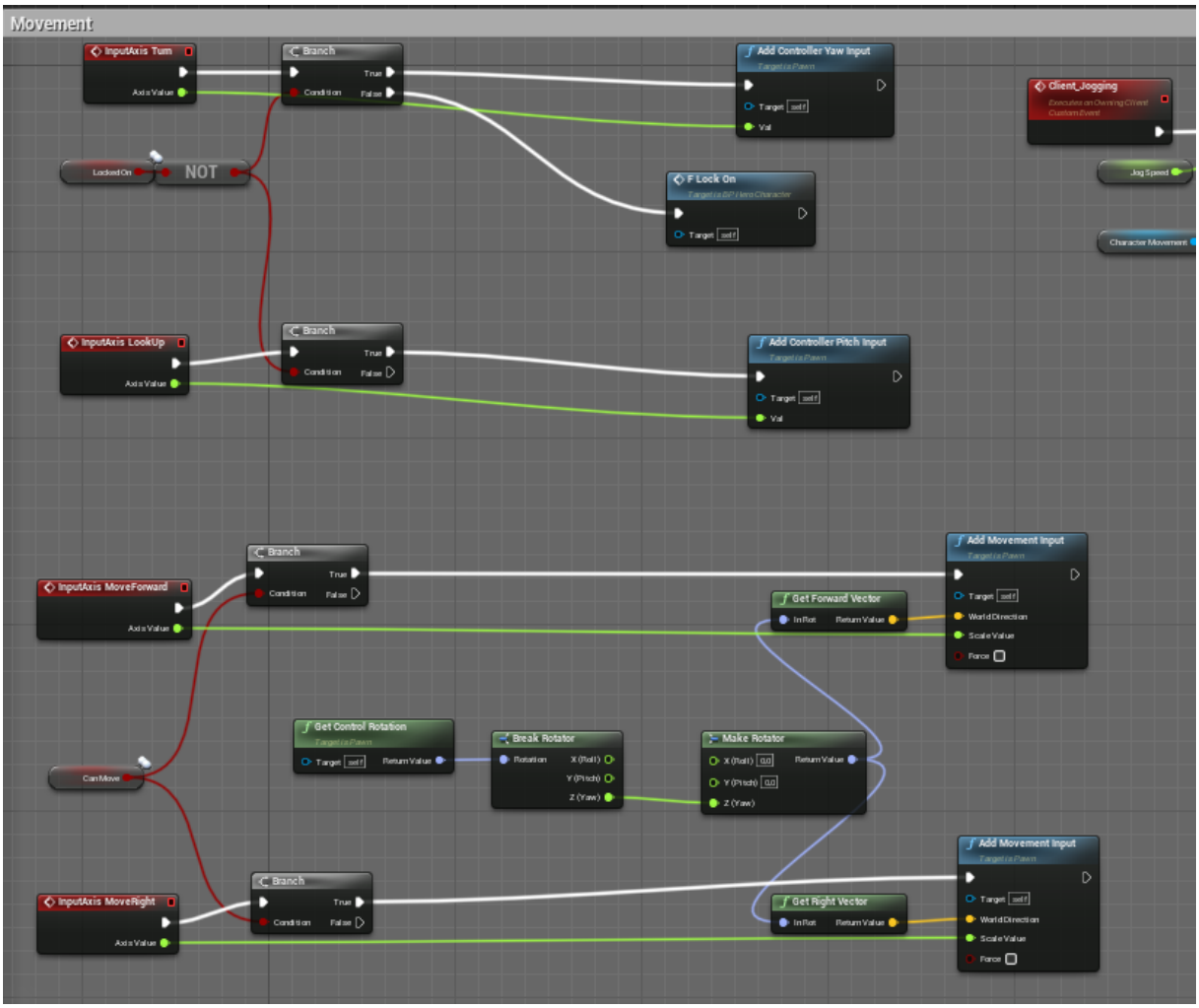
3.3.2. Kontrola glavnog lika

Prvo što želimo je omogućiti kretanje glavnog lika po nivou putem tipkovnice te razgledavanje okolo pomoću miša.



Slika 7: Ulazi *(eng. Inputs)* za kontrolu glavnog lika

Za kontrolu glavnog lika potrebno je definirat ulaze *(eng. Input)* kako bi u događajima *(eng. Event)* unutar nacrta mogli definirati što se dešava po kliku na tipku ili pokret miša.



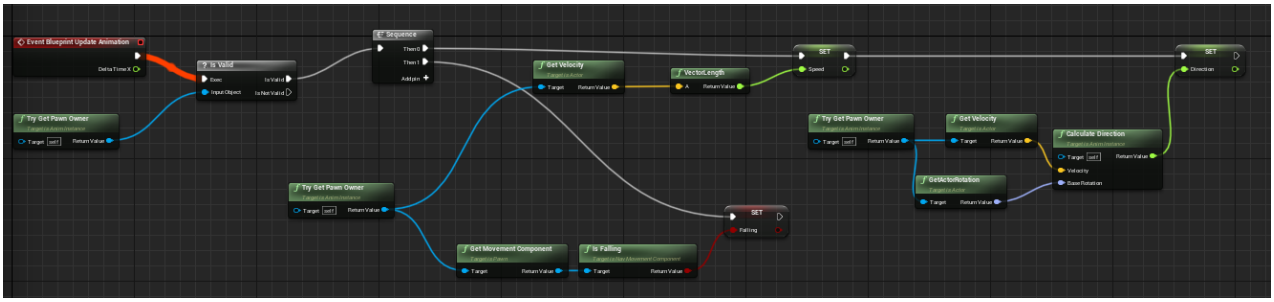
Slika 8: Kretanje glavnog lika i rotacija kamere

U nacrtu „BP_HeroCharacter“ dodajemo 2 događaja za rotaciju kamere, a to su „InputAxis turn“ za okretanje lijevo/desno i „InputAxis LookUp“ za okretanje kamere gore/dolje.

Nadalje za kretanje lika kreiramo događaje „InputAxis MoveForward“ i „InputAxisMoveRight“ te razdvajamo „Control Rotation“ preko „Breat Rotator“ zato što želimo da nam se glavni lik kreće samo po Z osi, te tu Z os šaljemo u „Get Forward/Right Vector“ kako bi omogućili kretanje glavnog lika po Z osi po svijetu.

3.3.2.1. Animacije kretanja za glavnog lika

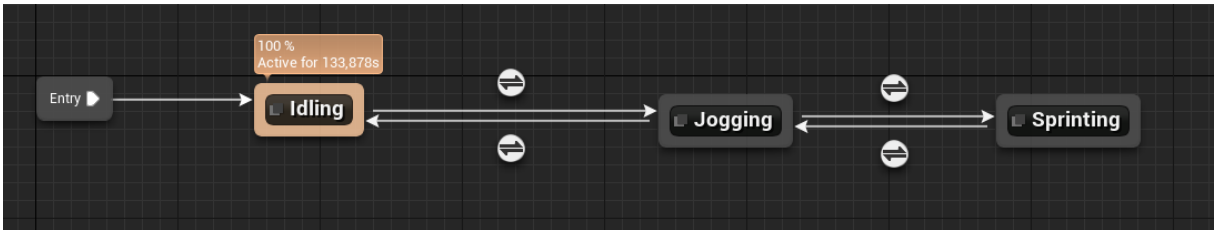
Kako bi osposobili animacije pri kretanju glavnog lika potrebno je prvo kreirati nacrt za animacije, a njega cemo nazvati „AnimBP_Greystone“.



Slika 9: Kretanje glavnog lika i rotacija kamere

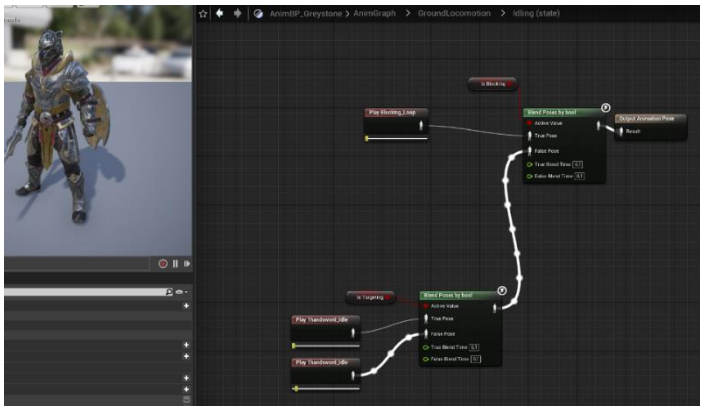
Za početak, unutar našeg nacrta za animaciju glavnog lika želimo dobiti float varijablu „speed“ preko koje ćemo definirati kojom brzinom se naš glavni lik kreče.

Nadalje unutar „GroundLocomotion“ definirati ćemo određena stanja *(eng. State)* u kojem se naš glavni lik nalazi s obzirom na vrijednost varijable „speed“.



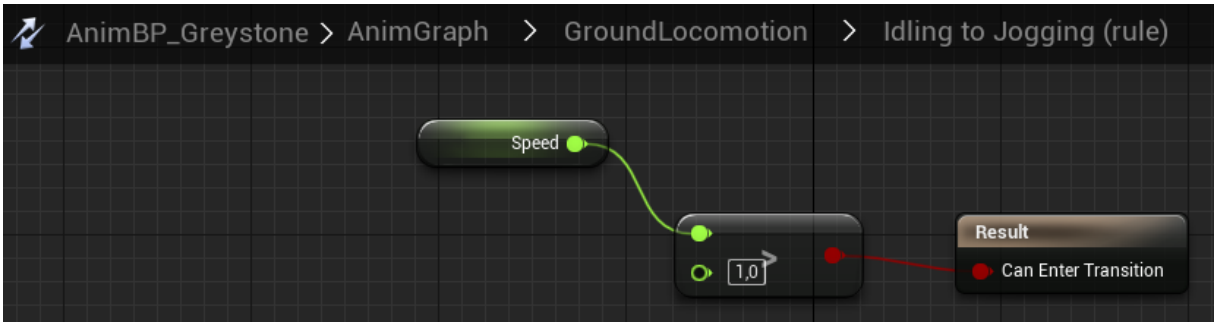
Slika 10: „GroundLocomotion“

Unutar svakog određenog stanja dalje definiramo animacije koje model našeg glavnog lika koristi kao što su mirovanje *(eng. Idling*, trčanje *(eng. Jogging)* i sprint *(eng. Sprinting).*



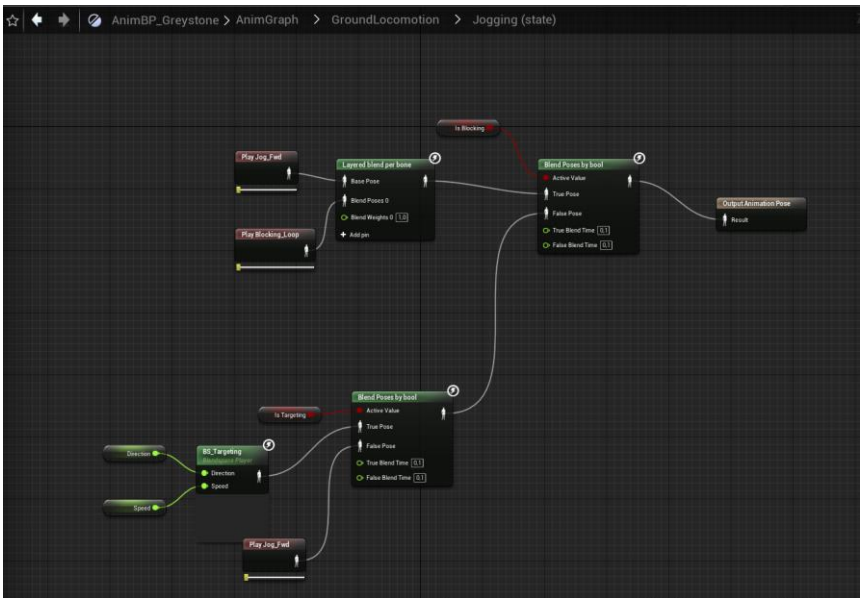
Slika 11: Stanje mirovanja *(eng. Idling state)*

Kako bi naš glavni lik prešao iz stanja mirovanja u stanje trčanja moramo provjeriti varijablu „speed“ te napraviti tu tranziciju između stanja mirovanja i trčanja.



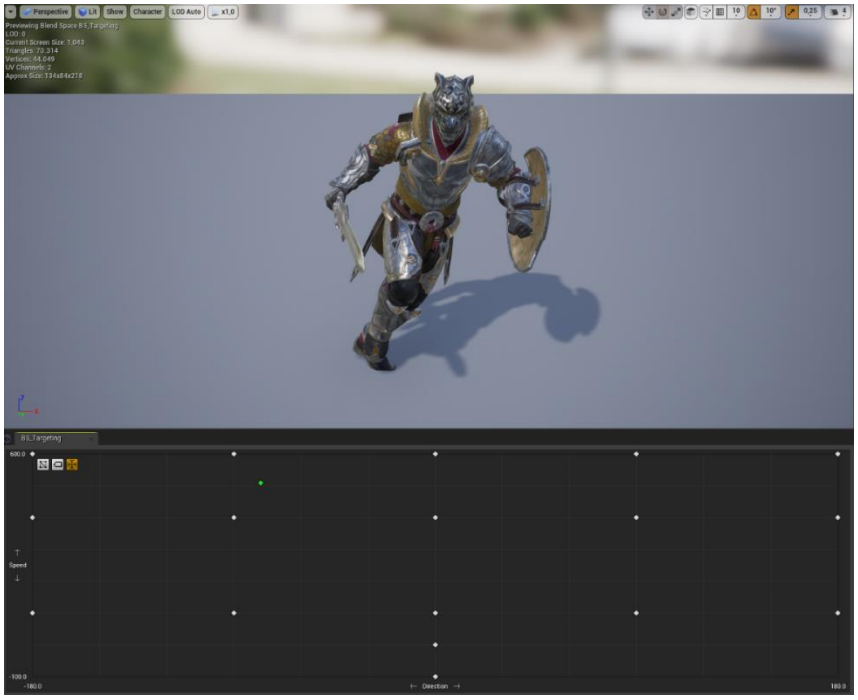
Slika 12: Tranzicija iz stanja mirovanja u stanje trčanja

Ako je varijabla „speed“ veća od 1 onda nam lik ulazi u tranziciju u stanje trčanja.



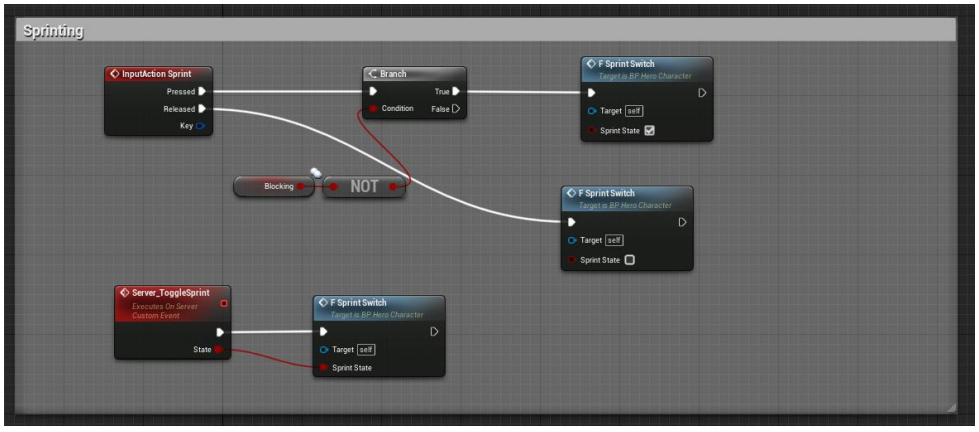
Slika 13: Stanje trčanja

Unutar stanja trčanja imamo definiran „BS_Targeting“ (BlendSpace Targeting) koji nam služi za određivanja animacije i smjera u kojemu se animacije kreću s obzirom na kretanje našeg lika i s obzirom na njegovu brzinu.



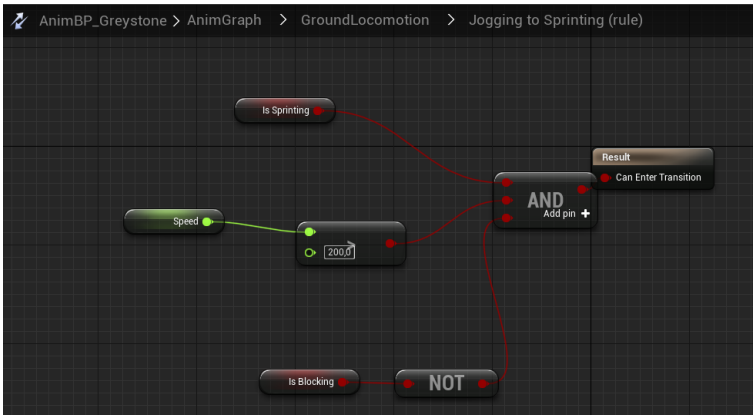
Slika 14: BlendSpace Targeting

Kao što prikazuje Slika 14. glavni lik nam je u animaciji „Jog_Left“ kada je smjer *(eng. Direction)* -90, a brzina *(eng. Speed)* 400, ako nam se brzina poveća događa se tranzicija u stanje sprinta *(eng. Sprinting state)*, ukoliko se smanji na 0 glavni lik nam je opet u stanju mirovanja *(eng. Idling state).*



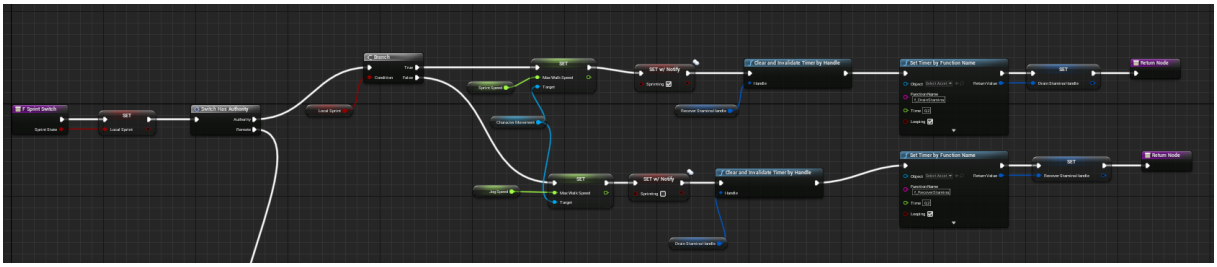
Slika 15: InputAction Sprint u BP_HeroCharacter nacrtu

Unutar nacrta „BP_HeroCharacter“ moramo definirati događaj *(eng. Event)* kojim ćemo postaviti brzinu kretanja glavnog lika u brzinu koja nam je potrebna kako bi glavni lik ušao u stanje sprinta i time započeo animaciju sprinta tranzicijom iz stanja trčanja u stanje sprinta.



Slika 16: Tranzicija stanja trčanja u stanje sprinta

Kako bi postavili brzinu kretanja glavnog lika najurednije je napraviti zasebnu funkciju koja će obavljati točno to, a funkciju smo nazvali „f_SprintSwitch“



Slika 17: funkcija f_SprintSwitch

Funkcijom „f_SprintSwitch“ postavljamo float varijablu „Max Walk Speed“ na vrijednost float varijable „Sprint Speed“ koja je postavljena na 750 po zadanoj vrijednosti *(eng. Default value)* te postavljamo bool varijablu „Sprinting“ na true. Pritiskom gumba „shift“ na tipkovnici brzina odnosno varijabla „Speed“ postavljena je na 750 i kada pokrenemo glavnog lika u neki smjer započeti će stanje sprinta.

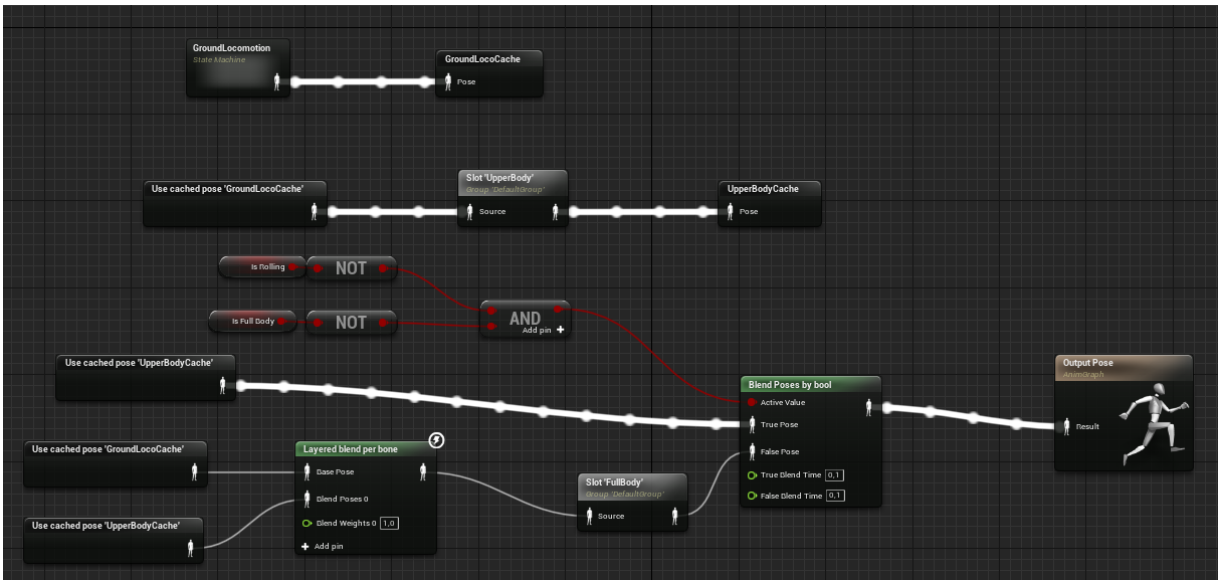
Isto tako, prestankom držanja gumba „shift“ ponovo se pokreće događaj *(eng. Event)* preko „Released“ poziva i započinje funkciju „f_SprintSwitch“ koja ovaj put postavlja varijablu „Max Walk Speed“ na vrijednost varijable „Jog Speed“, odnosno na „Max Walk Speed“ od 400 te se bool varijabla „Sprinting“ postavlja na „false“.

3.3.2.2. Animacija kotrljanja glavnog lika

Budući da za lika Greystone sve animacije koje imamo su slijed animacije *(eng. Animation sequence)* potrebno je pretvoriti te animacije u montažu animacije *(eng. Animation montage)* kako bi mogli koristiti te animacije na našem liku po pozivu neke funkcije ili događaja.

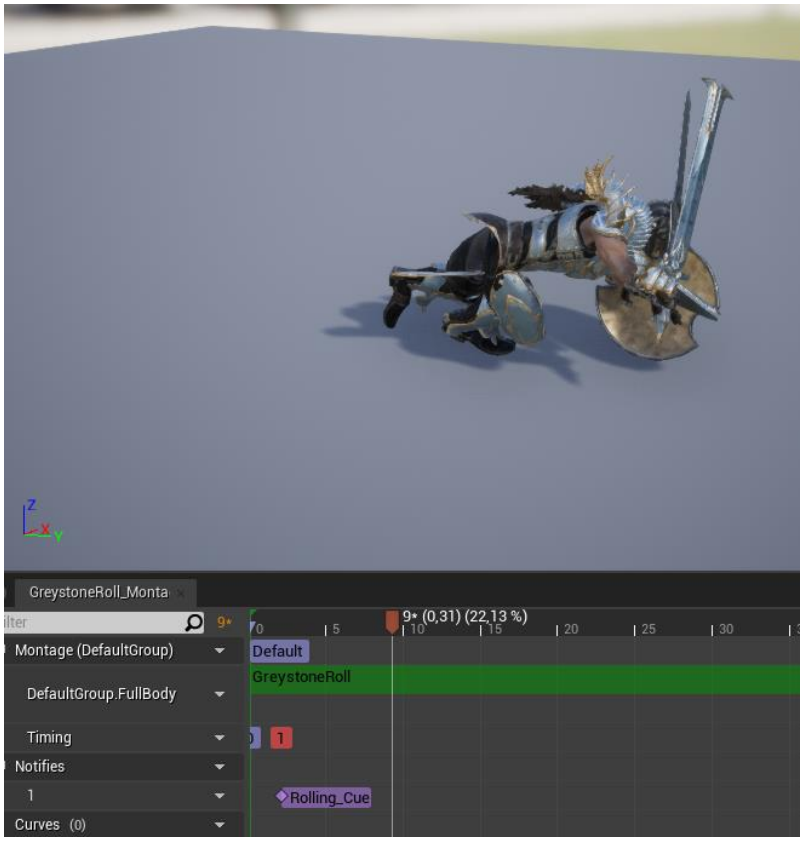
Sljedeća funkcija koju želimo napraviti je funkcija za kotrljanje te nam je za to potrebna animacija kotrljanja, no prije nego što slijed animacije za kotrljanje pretvorimo u montažu animacije bitno je unutar slijeda animacije postaviti „EnableRootMotion“ na „true“, bez RootMotion-a glavni lik nakon izvođenja animacije bi nam se vratio na prvobitno mjesto gdje je i započeo animaciju.

Nakon kreiranja montaže animacije *(eng. Animation montage)* koju ćemo nazvati „GreystoneRoll_Montage“, bitno je da definiramo kojem utoru *(eng. Slot)* pripada montaža unutar našeg „AnimBP_Greystone“ animacijskog nacrta. Kako nam animacija kotrljanja kontrolira cijeli dio našeg glavnog lika potrebno je definirati „Slot 'FullBody'“ kojim će animacija koristiti cijelo tijelo glavnog lika, kasnije uvođenjem animacije držanja štita morati ćemo odvojeno gledati gornji i donji dio tijela glavnog lika.



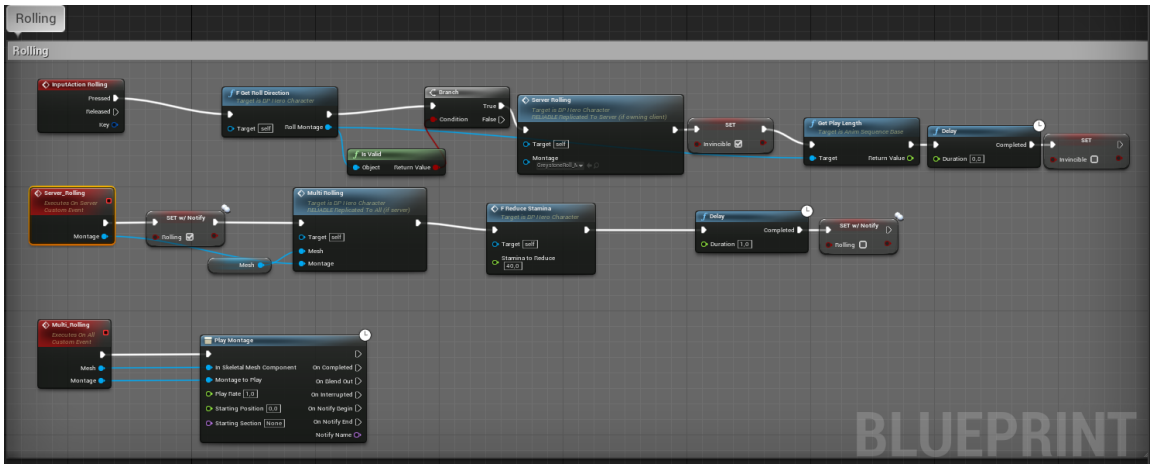
Slika 18: Slot 'FullBody'

Nadalje, unutar „GreystoneRoll_Montage“ postavljamo „Montage (DefaultGroup)“ na „DefaultGroup.FullBody“ kako bi model *(eng. Mesh)* Greystonea koristio svoje cijelo tijelo za izvođenje animacije.



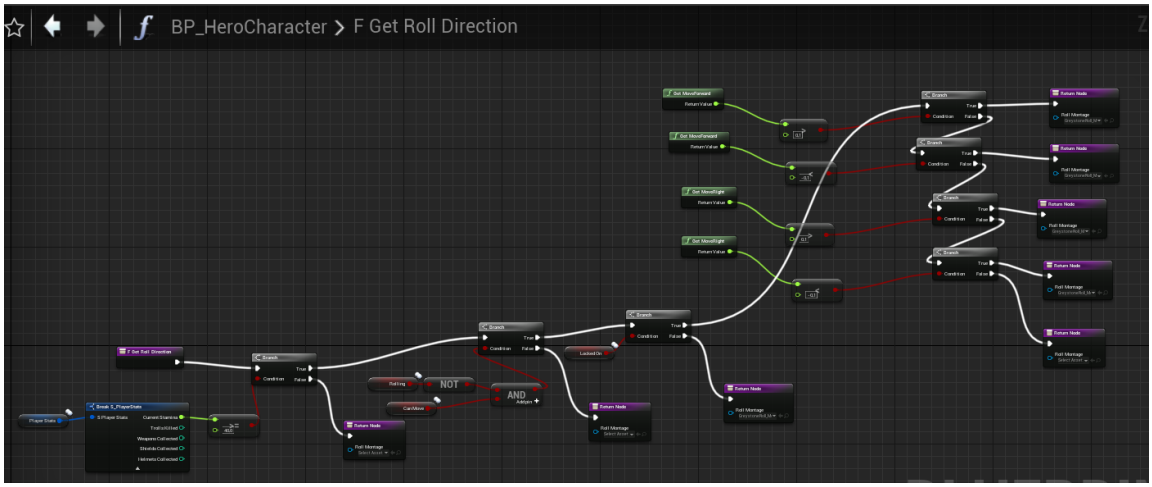
Slika 19: GreystoneRoll_Montage

Kako bi osposobili glavnog lika da može započeti montažu kotrljanja potrebno je unutar nacrta „BP_HeroCharacter“ napraviti događaj *(eng. Event)* koji se poziva pri kliku tipke „spacebar“



Slika 20: InputAction Rolling

Nakon što započne događaj *(eng. Event)* „InputAction Rolling“ prvo se poziva funkcija „F Get Roll Direction“ pomoću koje spremamo animaciju, no najbitnije je da preko te funkcije omogućujemo kotrljanje u svim smjerovima kretanja (naprijed, nazad, lijevo, desno).



Slika 21: F Get Roll Direction

„F Get Roll Direction“ prvo provjerava ima li igrač dovoljno izdržljivosti *(eng. Stamina)* čije ćemo osposobljavanje i funkcionalnost objasniti u daljnjim naslovima. Nakon toga, funkcija provjerava ukoliko igrač već je usred animacije kotrljanja te ako nije onda može ići dalje, ujedno tako provjerava smije li se igrač kretati te ako su oba uvjeta zadovoljena funkcija nastavlja dalje.

Pri samom kraju funkcije funkcija provjerava u kojem smjeru se igrač kreće i to je obavljeno preko „Get MoveForward“ i „Get MoveRight“ funkcija, ako je vrijednost veća od 0.1 znači da se igrač kreće i da ide u smjeru unaprijed *(eng. Forward)* ili udesno *(eng. Right)*, ako je na 0 igrač miruje, a ako je vrijednost manja od -0.1 znači da se igrač kreće unazad *(eng. Backward)* ili ulijevo *(eng. Left)* te s obzirom na zadovoljeni uvjet vraća vrijednost pomoću povratnog čvora *(eng. Return node)* koja sadrži animaciju kotrljanja „GreystoneRoll_Montage“.

Nakon izvršavanja funkcije dalje provjeravamo je li ispravan objekt koji smo vratili preko povratnog čvora *(eng. Return node)* te pokrećemo događaj *(eng. Event)* „Server Rolling“ iz Slike 20 koji sadrži animaciju kotrljanja, postavlja stanje igrača „Rolling“ na istinito *(eng. True)*, započinje događaj *(eng. Event)* „Multi_Rolling“, smanjuje izdržljivost *(eng. Stamina)* za 40 te čeka 1 sekundu da animacija završi kako bi postavio stanje igrača „Rolling“ na neistinito *(eng. False)*.

Događaj *(eng. Event)* započinje animaciju preko „Play Montage“ funkcije kojoj prosljeđujemo model *(eng. Mesh)* lika i animaciju koja se treba izvesti.

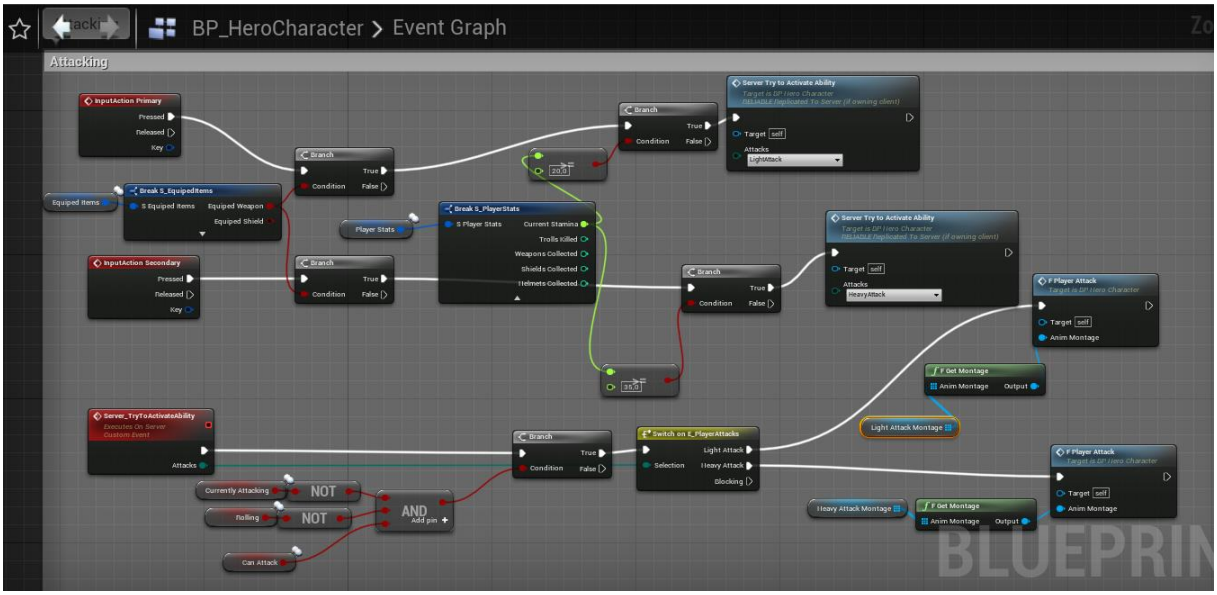
3.3.2.3. Animacija napada glavnog lika

Igraču omogućujemo dvije vrste napada, a to su „Light Attack“ i „Heavy Attack“ te budući da omogućujemo igraču odabir između ta dva napada stvaramo nabrajanja *(eng. Enumeration)* „E_PlayerAttacks“ unutar kojeg imamo deklariran „Light Attack“, „Heavy Attack“ i „Blocking“.

Za „Light Attack“ koristimo 2 animacije, a to su „LightAttackA_Montage“ i „LightAttackB_Montage“, koje su dobivene sa samim likom Greystonea, no u obliku sekvence animacija koje također moramo promijeniti u montažu animacija.

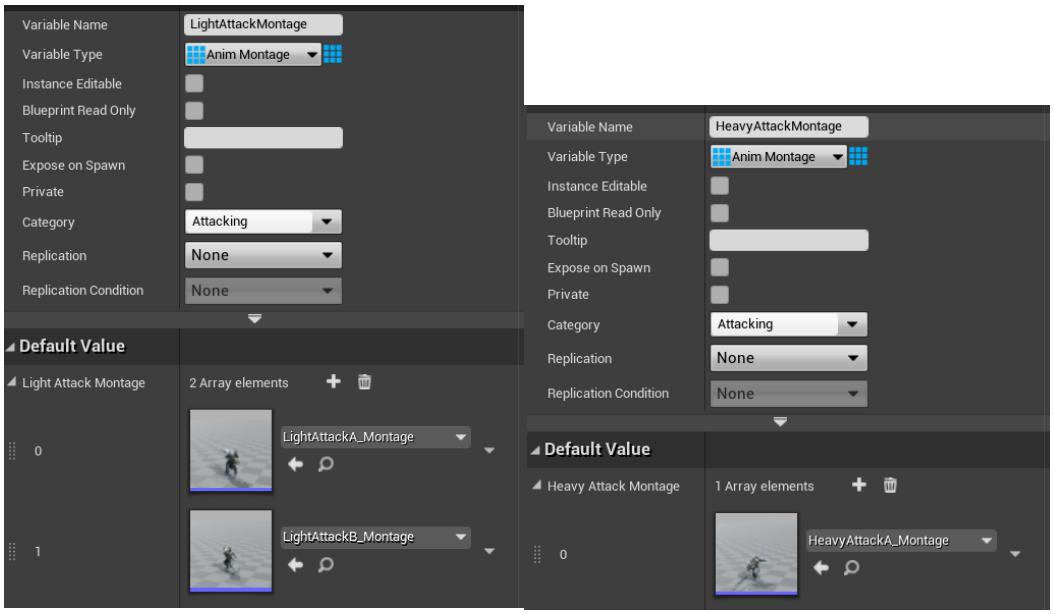
Za „Heavy Attack“ koristimo samo 1 animaciju, a to je „HeavyAttackA_Montage“ te nju također pretvaramo u montažu animacija kako bi ju mogli koristiti preko funkcije „Play Montage“.

Isto kao i sa kretanjem i kotrljanjem, da bi započeli animaciju potrebno je napraviti događaj *(eng. Event)* na pritisak tipke u ovom slučaju lijevog klika miša za „Light Attack“ i kombinacije tipaka shift + desni klik miša za „Heavy Attack“.



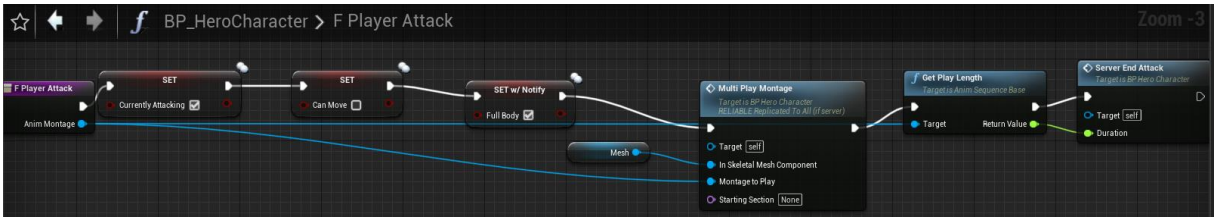
Slika 22: InputAction Primary („Light Attack“) i InputAction Secondary („Heavy Attack“)

Oba događaja *(eng. Events)* provjeravaju je li igrač opremljen mačem te ako je ima li dovoljno izdržljivosti *(eng. Stamina)* za napad te pomoću događaja *(eng. Event)* „Server Try_To_Activate_Ability“ dalje provjerava ukoliko igrač već napada, kotrlja se i je li u mogućnosti izvesti napad. Ako su uvjeti zadovoljeni ulazimo u nabrajanje *(eng. Enumeration)* koji se dalje grana s obzirom na igračev odabrani ulaz *(eng. Input)* te prosljeđuje nizove *(eng. Array)* koji sadrže animacije koje želimo izvesti.



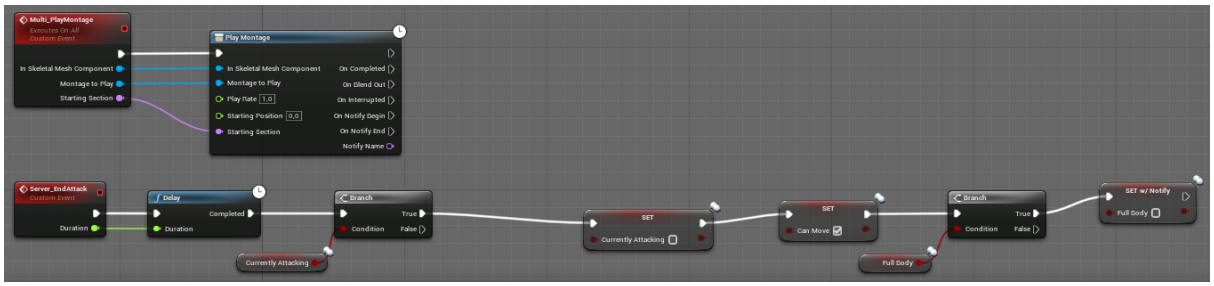
Slika 23: „LightAttackMontage“ niz i „HeavyAttackMontage“ niz

Niz animacija dalje prosljeđujemo preko funkcije „F Get Montage“ u funkciju „F Player Attack“.



Slika 24: funkcija „F Player Attack“

Funkcija „F Player Attack“ postavlja igrača u stanje napadanja te stavlja neistina *(eng. False)* na stanje „Can Move“ kako bi igrač morao ostati na mjestu tijekom izvođenja animacije te stanje „Full Body“ na istinu *(eng. True)* kako bi značilo da se koristi cijelo tijelo lika za animaciju. Montaža se prosljeđuje zajedno s modelom *(eng. Mesh)* u događaj *(eng. Event)* „Multi Play Montage“, dobiva se duljina izvođenja animacije preko „Get Play Length“ funkcije te se započinje događaj *(eng. Event)* „Server End Attack“.



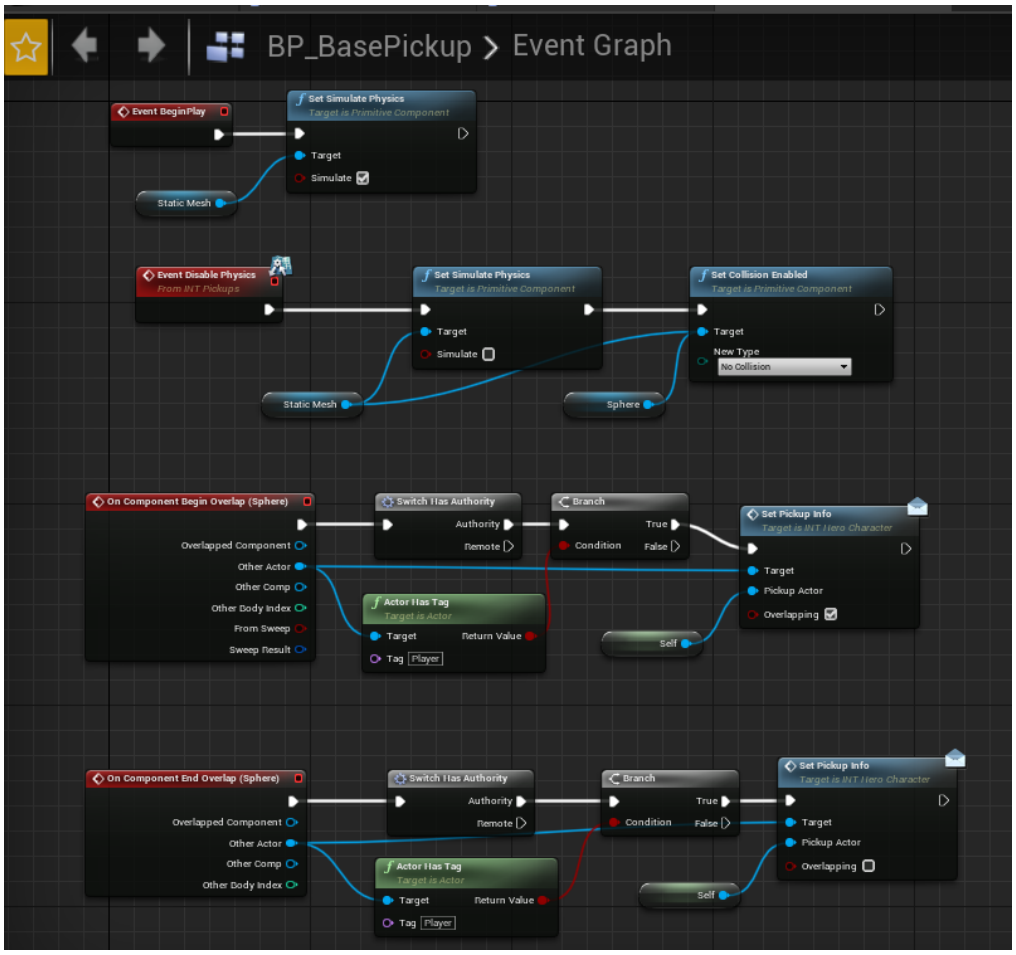
Slika 25: Događaji „Multi_PlayMontage“ i „Server_EndAttack“

„Multi_PlayMontage“ započinje animaciju na modelu lika koji su proslijeđeni u funkciju „Play Montage“, a „Server_EndAttack“ s obzirom na duljinu animacije nakon završetka vrača igrača u prvobitna stanja prije izvođenja animacije.

3.3.3. Uzimanje predmeta s poda i opremanje glavnog lika

Prije nego što bi osposobili događaje *(eng. Events)* za uzimanje predmeta s poda, potrebno je napraviti nacrt „BP_BasePickup“ unutar kojeg ćemo razviti logiku za same predmete.

Nacrt sa slike 26 poziva se na samom početku igre i on simulira fiziku za naše modele (eng. Mesh) predmeta preko Unreal Engine 4 funkcije „Set Simulate Physics“, a događaj (eng. Event) „Event Disable Physics“ koristimo kako predmeti ne mogu imati koliziju s drugim Sphere modelima kao što je naš igrač.

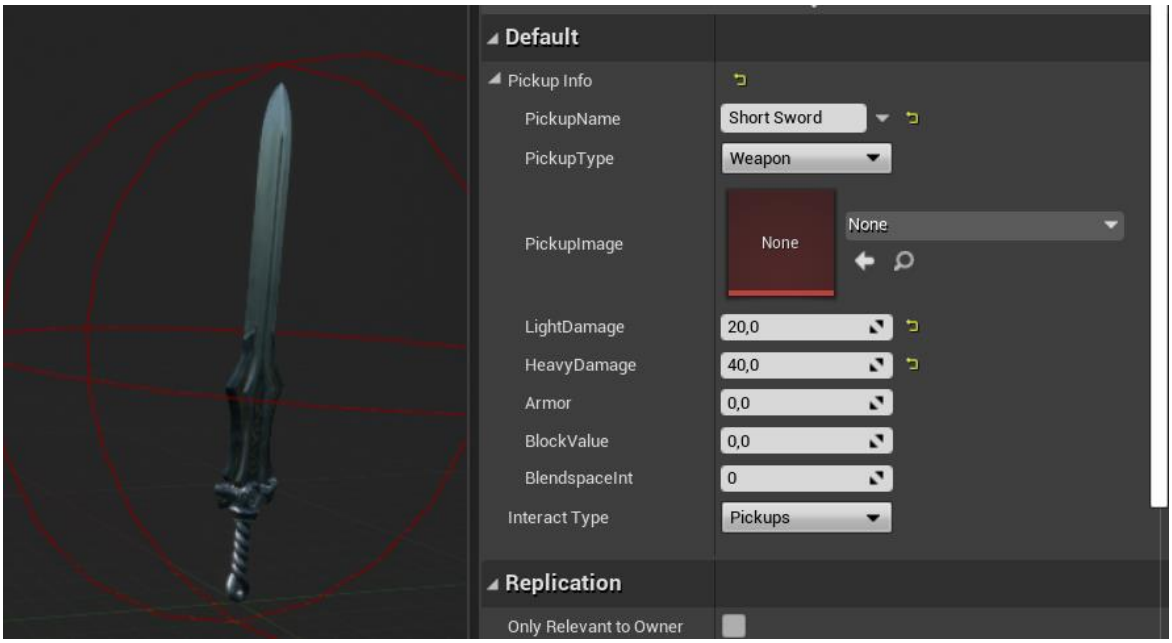


Slika 26: Nacrt „BP_BasePickup“

Događaji *(eng. Events)* „On Component Begin Overlap (Sphere)“ i „On Component end Overlap (Sphere)“ pozivaju se kada modeli drugih likova, odnosno njihove sfere, dođu u kontakt sa sferom predmeta *(eng. Pickup)*. Ti događaji prosljeđuju informaciju našem liku s kojim sferama, odnosno predmetima, ima koliziju.

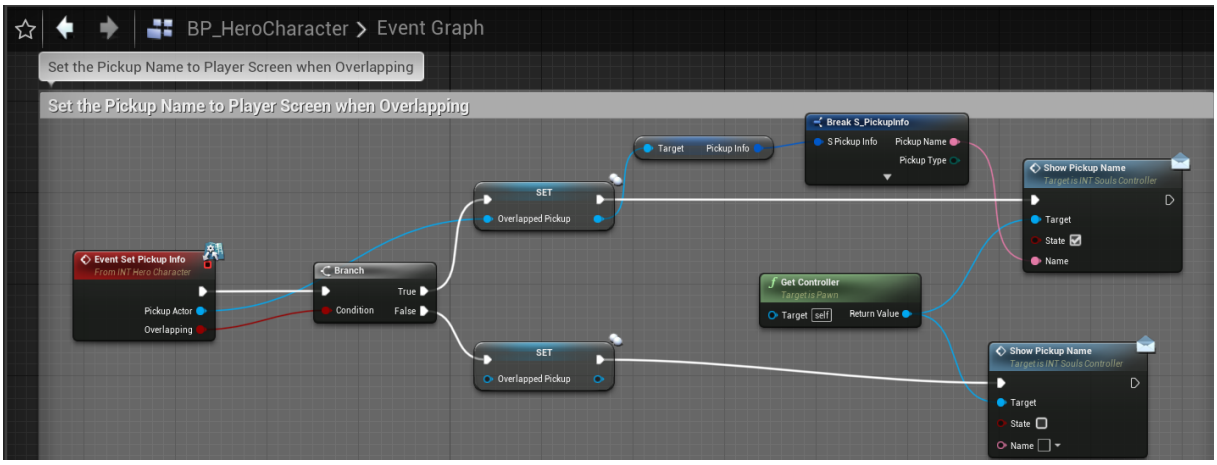
Nakon toga potrebno je stvoriti nacrt *(eng. Blueprint)* za svaki predmet koji želimo koristiti.

Unutar nacrta *(eng. Blueprint)* umećemo model *(eng. Mesh)* za predmet te određujemo u ovom slučaju za mač njegov „PickupName“, „PickupType“, „Light Damage“, „Heavy Damage“, „Armor“ i „BlockValue“ od kojih nas trenutno samo zanimaju prva četiri.



Slika 27: Nacrt „Pickup_ShortSword“

Sad kada se pozove događaj *(eng. Event)* „Set Pickup Info“ sa slike 28, prosljeđuju se informacije dobivene iz nacrta predmeta.



Slika 28: Događaj „Event Set Pickup Info“

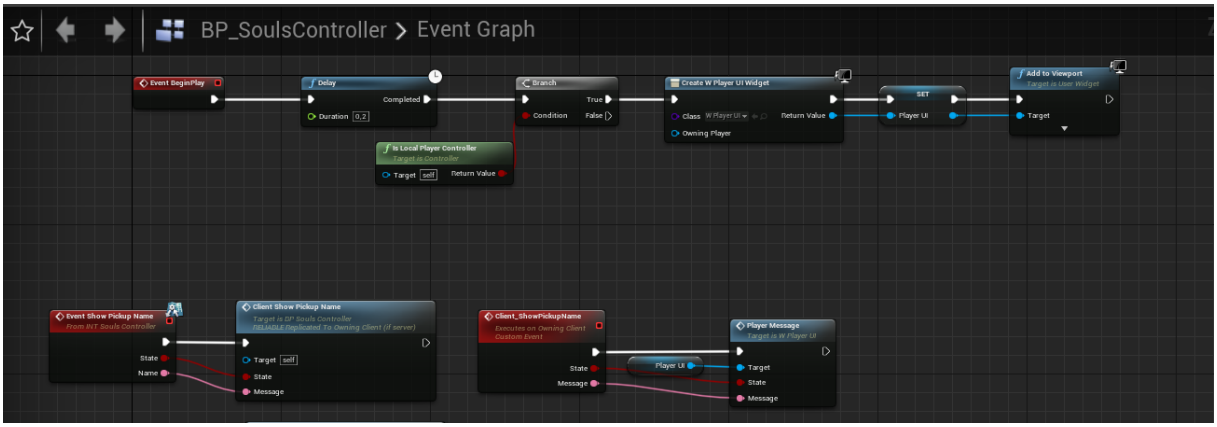
Događajem „Event Set Pickup Info“ prosljeđujemo ime predmeta s kojim se naš lik preklapa te započinjemo događaj *(eng. Event)* „Show Pickup Name“ kojim ćemo pokazivati korisničko sučelje *(eng. User interface)* odnosno kroz naš kreiran widget izbacivati poruku na ekran igrača.

Događaj *(eng. Event)* „Event Show Pickup Name“ deklariramo unutar nacrta „BP_SoulsController“ koji pripada „PlayerController“ klasi te preko tog nacrta koristimo widgete koje kreiramo.

Prije nego što pozovemo widget, potrebno ga je kreirati, stoga stvaramo novi widget nacrt koji ćemo nazvati „W_PlayerUI“.

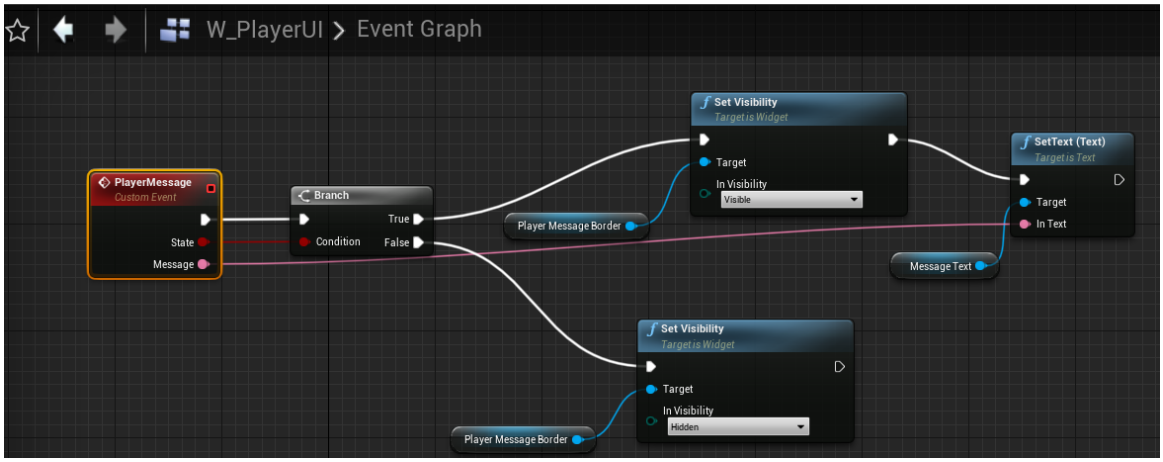
Slika 29: „W_PlayerUI“

Za naše predmete i izbacivanje poruke s kojim predmetom imamo koliziju koristimo samo „MessageText“.



Slika 30: „BP_SoulsController“

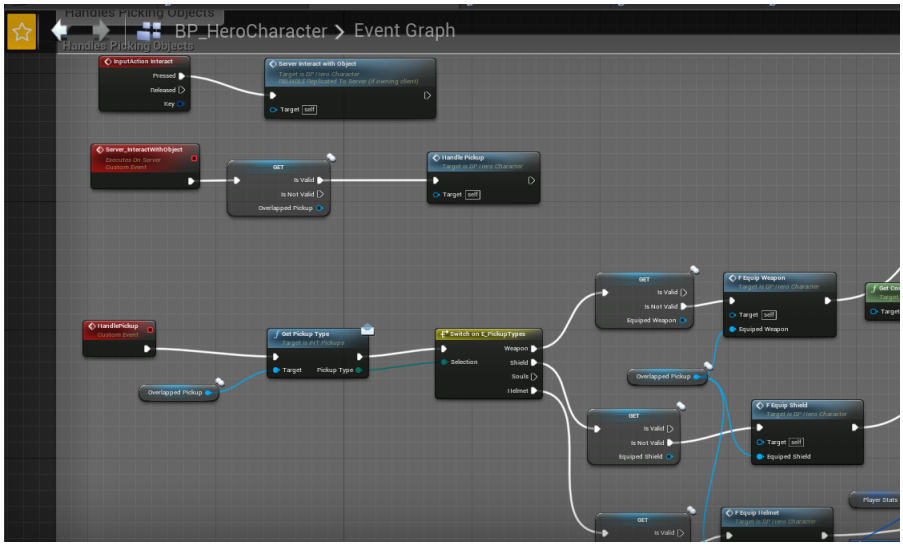
Pri početku igre, stvaramo widget preko „Create W Player UI Widget“ te ga dodajemo na ekran igrača preko „Add to Viewport“ funkcije. „Show Pickup Name“ sa slike 28 pozivamo ovdje i pokrećemo funkciju „Player Message“.



Slika 31: „PlayerMessage“ funkcija

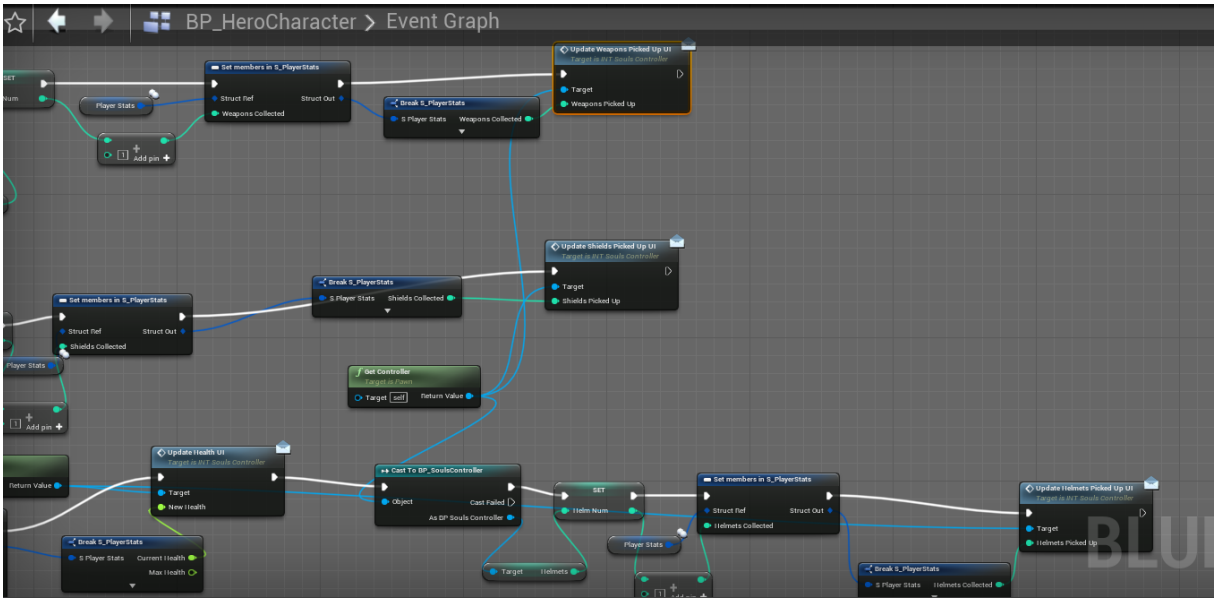
Koja postavlja „Message Text“ kao vidljiv *(eng. Visible)* ukoliko se dešava kolizija s modelom igrača te na igračev ekran postavlja tekst „Message Text“ kao naziv predmeta s kojim igračeva sfera ima koliziju.

Ukoliko igrač ima koliziju s predmetom može pokrenuti događaj „InputAction Interact“ koji se poziva klikom na tipku „F“.



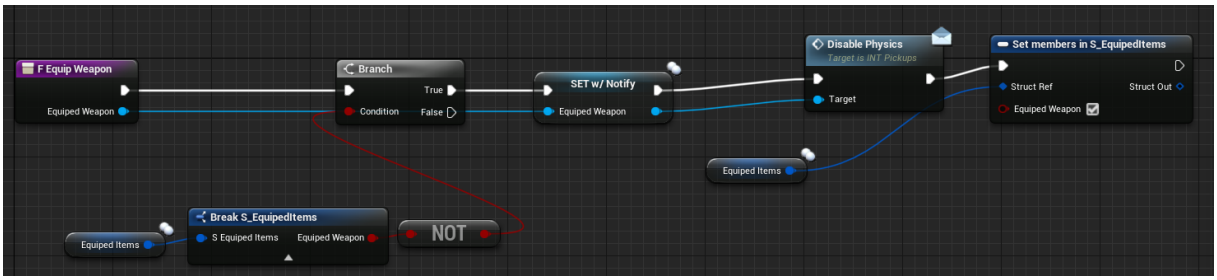
Slika 32: „InputAction Interact“

Događajem „InputAction Interact“ provjeravamo valjanost te pozivamo događaj „Handle Pickup“ koji s obzirom na vrstu predmeta izvodi funkcije „F Equip Weapon“, „F Equip Shield“ ili „F Equip Helmet“. Svaka od navedenih 3 funkcija povećava broj u tekstu widgeta sa slike 29. kao što su „Weapons/Shields/Helmets picked up“ preko „Update Weapons/Shields/Helmets Picked Up UI“ funkcije.



Slika 33: „Update Weapons/Shields/Helmets Picked Up UI“ funkcije

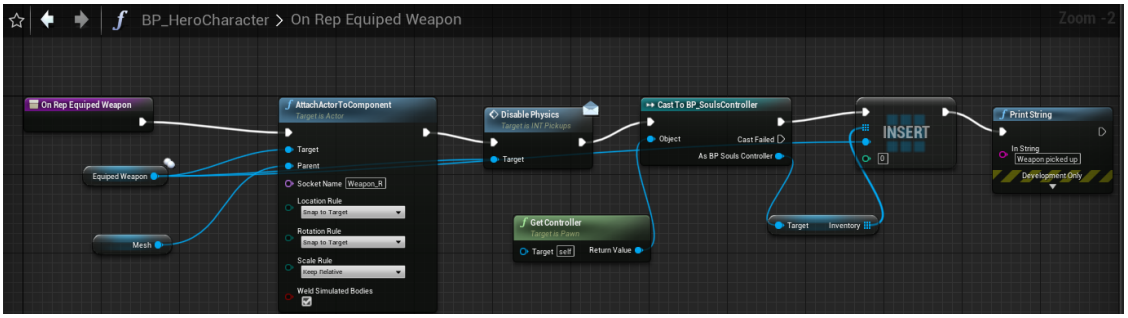
Funkcije „F Equip Weapon“, „F Equip Shield“ ili „F Equip Helmet“ sa slike 32 nam dalje izvode logiku iza stavljanja predmeta na model našeg lika.



Slika 34: funkcija „F Equip Weapon“

Unutar ove funkcije provjeravamo da igrač nema već uzeto oružje, mičemo fiziku s predmeta te postavljamo to oružje kao obučeno u strukturu „S_EquipedItems“.

Kako bi nam se oružje vidjelo kao model *(eng. Mesh)* dodan na model našeg lika Greystone radimo ponavljajuću funkciju „OnRep_EquipedWeapon“.



Slika 35: funkcija „OnRep_EquipedWeapon“

„OnRep_EquipedWeapon“ nam pomoću „AttachActorToComponent“ funkcije stavlja proslijeđeni model na model našeg lika u „Socket Name“ koji smo mu zadali, u ovom slučaju za oružje to je „Weapon_R“.

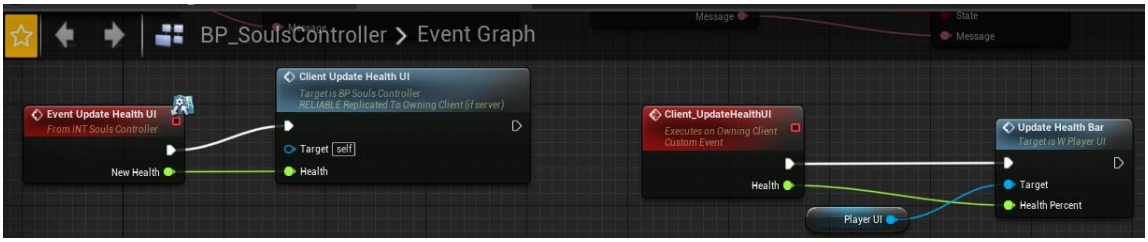
Ista logika s stavljanjem modela predmeta *(eng. Pickup)* na model našeg lika Greystone izvodi se i za štit i šljem.

3.3.4. Uzrokovanje štete i statistika glavnog lika

3.3.4.1. Životni bodovi i izdržljivost glavnog lika

Iz slike 29 možemo vidjeti u gornjem lijevom kutu dvije trake za napredak *(eng. Progress bar)*, gornju traku označenu crvenom bojom koristit ćemo za prikaz životnih bodova *(eng. Health points)* i donju traku označenu zelenom bojom za prikaz izdržljivosti *(eng. Stamina)*.

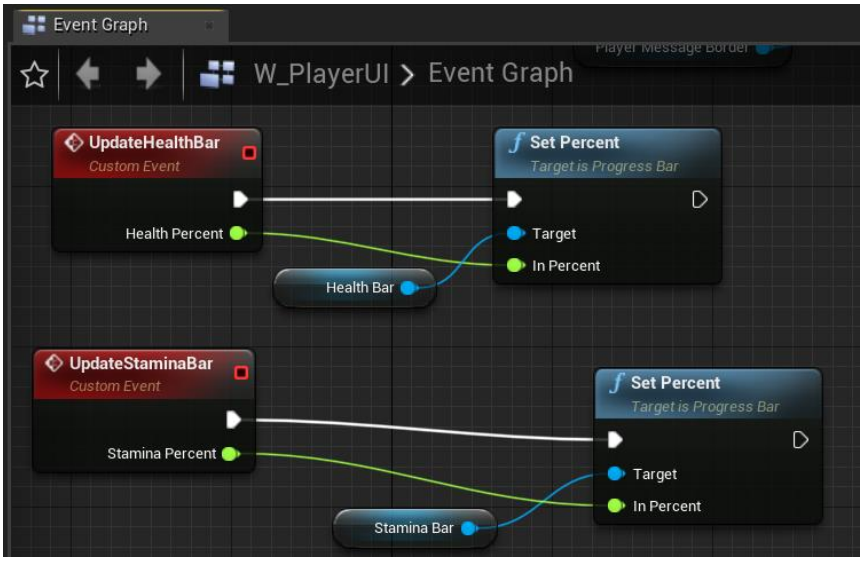
Funkcionalnosti crvene trake, odnosno traka za prikaz životnih bodova kao što sam naziv govori, služi za prikaz koliko životnih bodova ima naš igrač te kad se dogodi neka funkcija koja igraču smanjuje životne bodove (dalje u radu funkcija „Apply Damage“) ta traka će se smanjiti putem događaja „Event Update Health UI“ iz nacrta „BP_SoulsController“.



Slika 36: „Event Update Health UI“

Događaj nam poprima novu vrijednost životnih bodova koju ćemo dobivati oduzimanjem trenutnih životnih bodova sa štetom koju igrač primi.

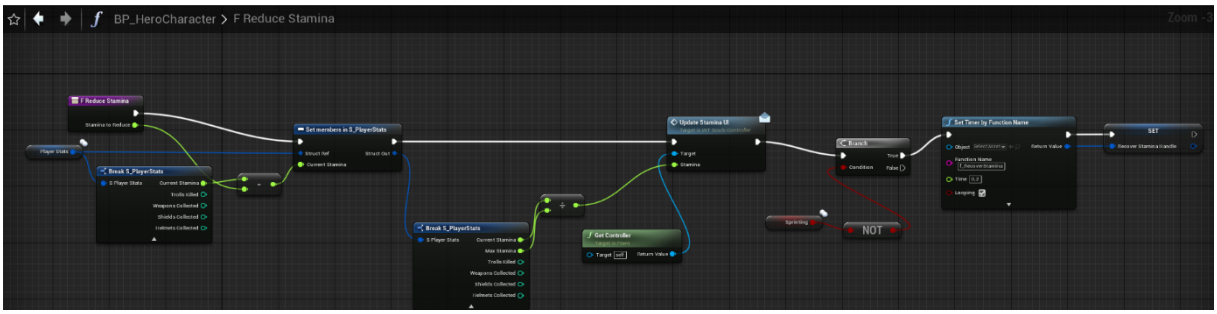
Nadalje događaj poziva funkciju „Update Health Bar“ koja se nalazi u samom widgetu „W_PlayerUI“ te unutar te funkcije korigira vrijednost trake za napredak *(eng. Progress bar)*.



Slika 37: „UpdateHealthBar“ i „UpdateStamina bar“

Kao što vidimo sa slike 37 isti princip za promjenu trake za napredak *(eng. Progress bar)* ima i traka za napredak za izdržljivost, ali je njena uporaba, odnosno pozivanje, van widgeta nešto drugačija. Kao u većini igara koji koriste mehaniku „izdržljivost“ *(eng. Stamina)* kao što su od prije navedene „Souls-like“ videoigre, potrebno je napraviti da izdržljivost se pasivno troši kroz korištenje napada, kotrljanja ili sprinta.

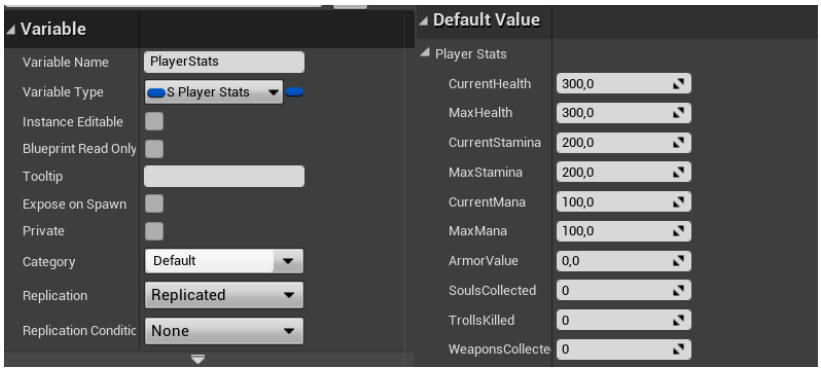
Kao što vidimo u slici 19 gdje usred događaja za kotrljanje pozivamo funkciju „F Reduce Stamina“.



Slika 38: „F Reduce Stamina“

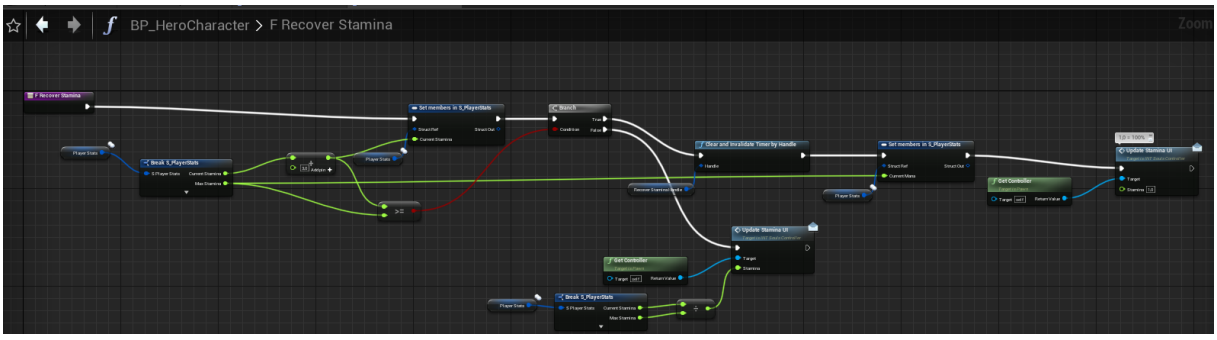
Funkcija „F Reduce Stamina“ kao ulaznu vrijednost poprima varijablu „Stamina to Reduce“ koja je tipa „float“ i pomoću nje smanjujemo igračevu izdržljivost.

Strukturu „Player Stats“ razdvajamo pomoću „Break S_PlayerStats“ funkcije kako bi izdvojili samo varijablu „Current Stamina“ te od „Current Stamina“ oduzimamo „Stamina to Reduce“ kako bi dobili novu vrijednost „Current Stamina“ varijable te je ponovo postavili u strukturu „S_PlayerStats“, nadalje kako bi ažurirali našu traku za napredak za izdržljivost potreban nam je postotak za tu traku, stoga uzimamo novu vrijednost „Current Stamina“ i nju dijelimo s „Max Stamina“ varijablom kako bi dobili postotak vrijednost koju prosljeđujemo „Update Stamina UI“ funkciji.



Slika 39: „PlayerStats“

Nakon svakog trošenja izdržljivost važno je započeti ponavljajuću funkciju koja će igraču vračati potrošenu izdržljivost. To izvodimo preko „Set Timer by Function Name“ funkcije koja će ponavljati funkciju „F Recover Stamina“ svake 0,2 sekunde te ju postavljamo na vrijednost „Looping“ kao istina *(eng. True)*.



Slika 40: „F Recover Stamina“

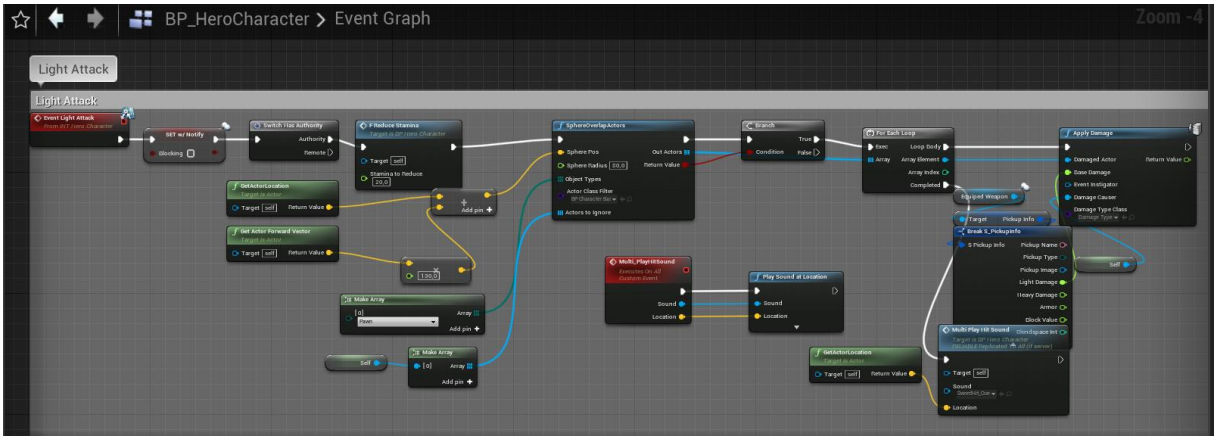
Funkcija „F Recover Stamina“ na isti princip kao „F Reduce Stamina“ razdvaja „Player Stats“ kako bi koristila varijable „Current Stamina“ i „Max Stamina“, funkcija uzima „Current Stamina“ te joj pridodaje 3 i postavlja kao novu vrijednost „Current Stamina“ u strukturi „S_PlayerStats“, dalje provjeravamo ukoliko je nova vrijednost „Current Stamina“ veća ili jednaka „Max Stamina“, odnosno ako je traka za napredak izdržljivosti puna.

Ako je veće ili jednaka zaustavljamo funkciju pomoću „Clear and Invalidate Timer by Handle“ te ažuriramo traku za napredak izdržljivosti preko „Update Stamina UI“ funkcije.

Ako je manja onda samo ažuriramo traku za izdržljivost pomoću funkcije „Update Stamina UI“ i ponavljamo „F Recover Stamina“ funkciju sve dok se uvjet ne zadovolji.

3.3.4.2. Uzrokovanje štete

Naš igrač može uzrokovati štetu neprijateljima preko dva napada, a to su „Light Attack“ i „Heavy Attack“ događaji *(eng. Events)*.



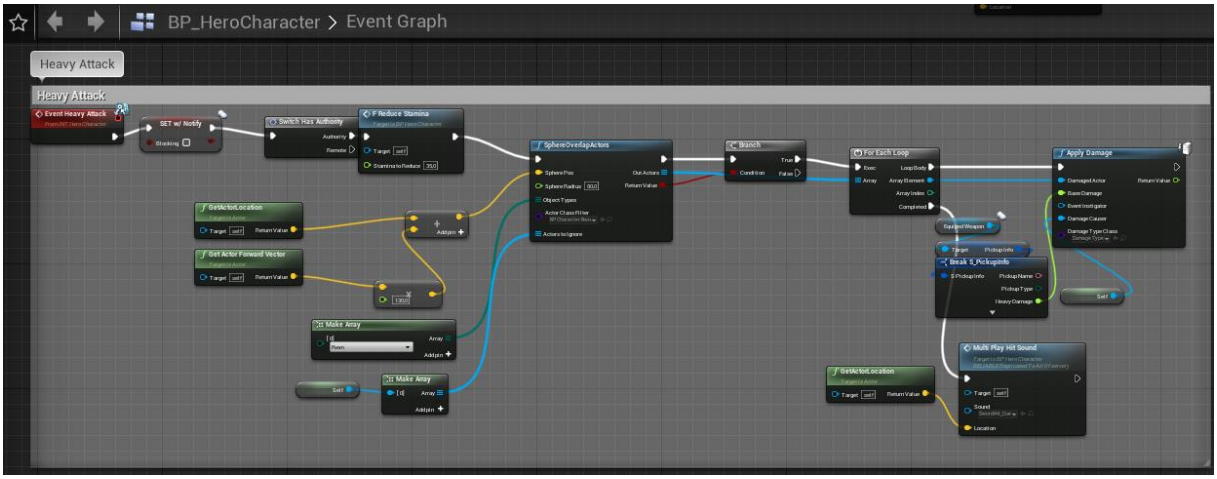
Slika 41: „Event Light Attack“

Ukoliko se u događaju sa slike 22 dogodi da se aktivira LightAttack, pokreće se događaj animacije i zajedno s njom „Event Light Attack“ preko „Animation Notify State“ što su događaji koji se pokreću pri pokretanju zadane animacije. „Event Light Attack“ prvo smanjuje izdržljivost igraču preko „F Reduce Stamina“ funkcije opisane u slici 36, zatim s obzirom na lokaciju igrača stvara sferu ispred igrača tako da izračunamo poziciju sfere i proslijedimo njenu lokaciju u „SphereOverlapActors“ funkciju, dodamo njen radijus, tip objekta te stavljamo da ignorira našeg igrača tako da referencu na ovaj nacrt *(eng. Blueprint)*, odnosno „self“, dodamo u niz *(eng. Array)* te proslijedimo u „Actors to Ignore“ unutar „SphereOverlapActors“.

Nadalje provjeravamo postoji li neka kolizija za sferom odnosno ako je „Return Value“ postavljen na istinu *(eng. True)* onda za svako ponavljanje, budući da možemo napasti više neprijatelja odjednom, izvodimo funkciju „Apply Damage“, kojoj prosljeđujemo sferu ostalih „actora“ s kojima naša sfera ima koliziju, broj štete koji se izvodi „Base Damage“ dobiven iz strukture „S_PickupInfo“ koji sadrži štetu koju nanosi naše oružje, koje smo pokupili te tko izvršava štetu, odnosno referenca na „Self“,a to je u ovom slučaju naš igrač iliti nacrt „BP_HeroCharacter“.

Nakon što se sva ponavljanja izvrše „For Each Loop“ ide u završnu fazu *(eng. Completed)* te pokreće „Multi Play Hit Sound“ funkciju na lokaciji gdje se nalazi naš igrač i pokreće zvuk „SwordHit_Cue“ na toj lokaciji.

Na identičan princip uz izmjene potrošnje izdržljivost i radi eventualne buduće izmjene je napravljen odvojeno „Heavy Attack“, odnosno događaj *(eng. Event)* „Event Heavy Attack“.



Slika 42: „Event Heavy Attack“

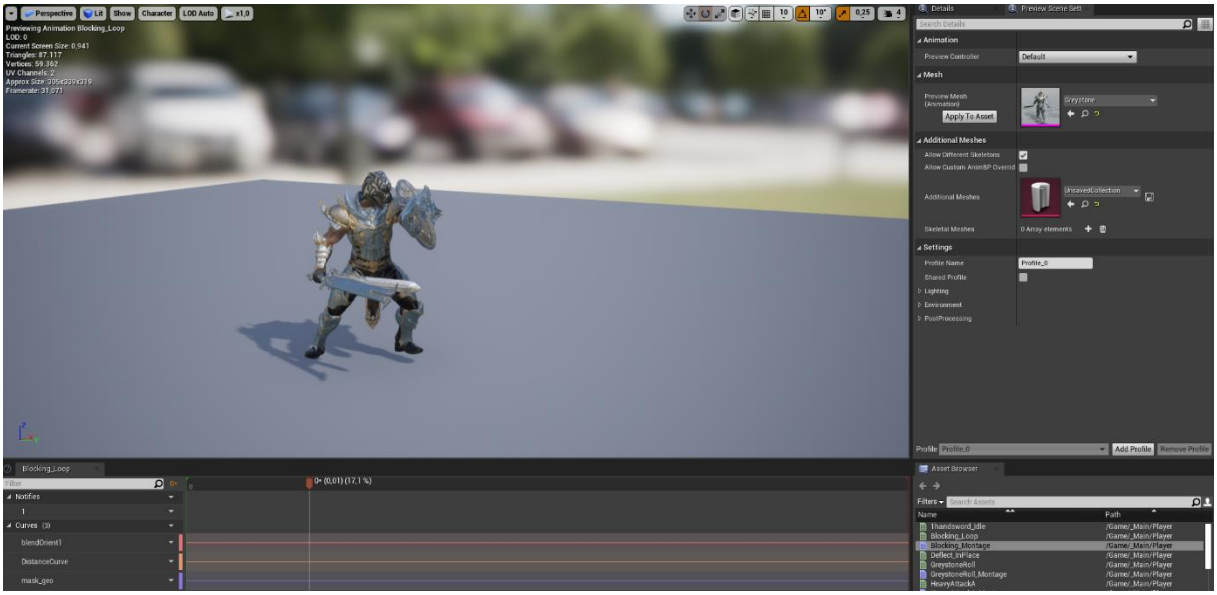
3.3.5. Osposobljavanje glavnih mehanika

3.3.5.1. Mehanika blokiranja sa štitom

Kako je mehanika blokiranja jedna od najznačajnijih aspekta „Souls-like“ videoigara za izbjegavanje nailazeće štete od protivnika, zajedno s kotrljanjem, potrebno je da i to imamo implementirano za našeg igrača.

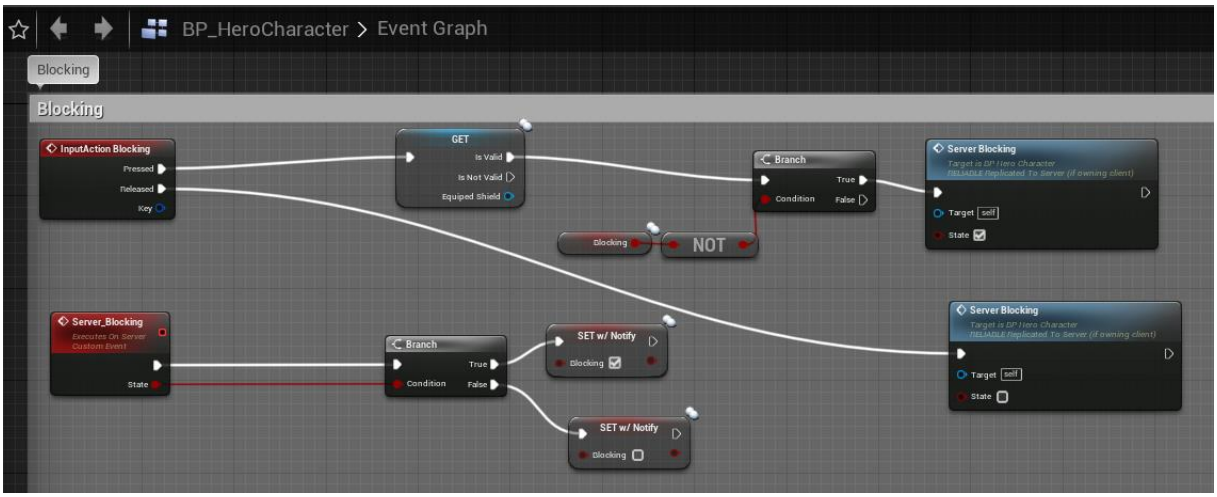
Kao što smo i prije radili za kotrljanje, napade i slično, bitno je prvo osposobiti montažu animacije *(eng. Animation montage)* za našeg Greystone lika prije nego što počnemo razvijati samu logiku blokiranja sa štitom.

Greystone ne dolazi s animacijom za blokiranje sa štitom stoga uzimamo sekvencu animacije njegovog „Ability_Q“ napada u kojem Greystone udara mačem u štit, no nama ne treba cijela animacija, nego samo dio gdje drži štit ispred sebe, stoga uzimamo tu sekvencu, skraćujemo je onoliko dovoljno da imamo samo petlju *(eng. Loop)* u kojoj Greystone samo drži štit ispred sebe.



Slika 43: „Blocking Loop“

Nakon što nam je animacija spremna za uporabu, možemo razviti jednostavnu početnu logiku u nacrtu *(eng. Blueprint)* „BP_HeroCharacter“ kako bi postavili igrača u stanje blokiranja.



Slika 44: „InputAction Blocking“

„InputAction Blocking“ jednostavan je događaj kojim pritiskom desne tipke miša prvo provjeravamo blokira li naš igrač već, ako ne blokira onda se pokreće događaj „Server_Blocking“ kojim postavljamo bool varijablu „Blocking“ na istinu *(eng. True)*, a ukoliko igrač pusti desni klik miša „Server Blocking“ se postavlja na neistinu *(eng. False)* te time ujedno i bool varijabla „Blocking“ na neistinu *(eng. False)*.

Nadalje. isto kako smo radili u „Animacije kretanja za glavnog lika“ djelu ovog završnog rada, moramo postaviti blokiranje kao stanje u kojem se nalazi naš igrač isto kao što smo napravili stanje mirovanja, trčanja i sprinta. Kao što vidimo na slici 11. ukoliko nam je igrač u stanju blokiranja, odnosno ako je bool varijabla „Blocking“ postavljena na istinu *(eng. True)* onda nam se pokreće „Play Blocking_Loop“ sekvenca kao „Output Animation Pose“.

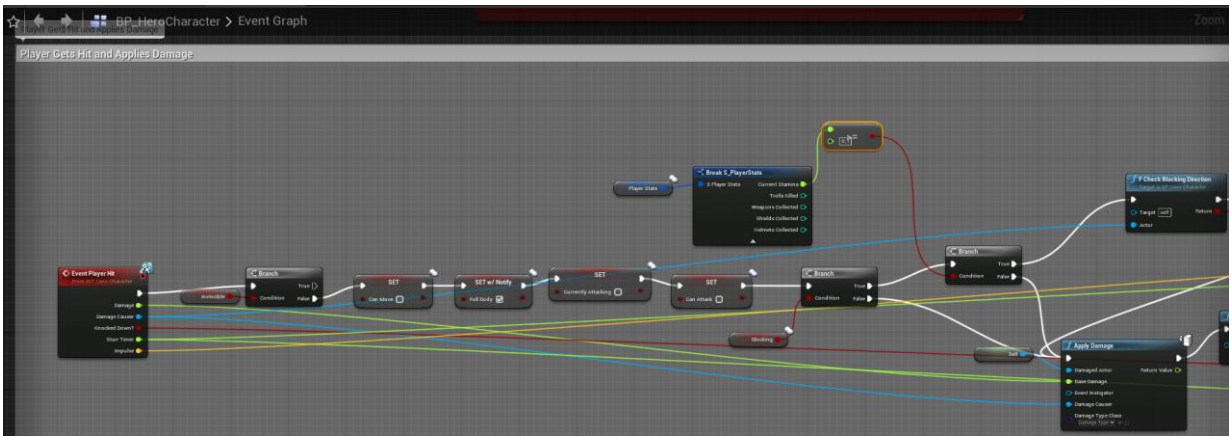
Važna je obratiti pozornost na to da ako nam se izvodi sekvenca „Play Blocking_Loop“ i igrač nam se pokreće on će kliziti po podu bez animacije hodanja, stoga je bitno da dok igrač blokira da se koristi samo „UpperBodyCache“ odnosno samo gornji dio tijela lika za animaciju blokiranja, a donji dio tijela ostane funkcionalan i dalje za animacije hodanja. Na slici 12. možemo uočiti kako smo stavili da ukoliko je igrač u stanju blokiranja odnosno ako je bool vrijednost „IsBlocking“ postavljena na istinu *(eng. True)* da će nam se odvijat obje sekvence „Play Jog_Fwd“ i „Play Blocking_Loop“ za odvojene dijelove modela našeg Greystone lika.

Kako imamo sada osposobljenu animaciju, potrebno je razviti logiku što se točno događa dok naš lik je u stanju blokiranja te kako to utječe na štetu koju prima naš igrač.

Kako bi definirali to, bitno je prvo definirati što se dešava kada naš igrač primi neki oblik štete.

3.3.5.2. Primanje štete i negiranje štete blokiranjem

Primanje štete jedna je od najkompliciranijih logika u ovom završnom radu te će zahtijevati detaljno praćenje kako bi se pohvatalo što se točno dešava u trenutku dok naš igrač primi štetu. Za početak, kad god neki neprijatelj ošteti našeg lika on poziva događaj *(eng. Event)* „Event Player Hit“, a sami poziv tog događaja ćemo detaljnije objasniti u dijelu rada gdje ćemo osposobljavati neprijatelja i njegovu umjetnu inteligenciju.



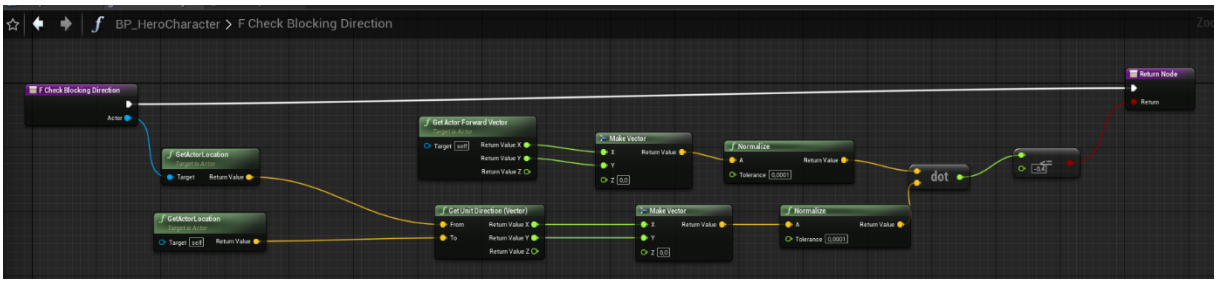
Slika 45: „Event Player Hit“, šteta blokirana

Za početak idemo pratiti putanju što se desi kada naš igrač bude pogođen, ali uspijeva blokirat štetu.

Prvo što provjeravamo je bool vrijednost „Invincible“ koja je aktivna za vrijeme trajanja naše animacije kotrljanja, ukoliko se desi kolizija sfera u kojoj naš igrač prima štetu, a igrač je usred animacije kotrljanja „Event Player Hit“ događaj staje ovdje te igrač ne prima nikakvu štetu jer se bool „Invincible“ postavlja na istinu *(eng. True)*.

Ukoliko igrač ne ispuni uvjet za grananje kod „Invincible“ idemo dalje u kodu te postavljamo kretanje lika odnosno bool vrijednost „Can Move“ na neistinu *(eng. False)*, postavljamo korištenje „Full Body“ na istinu *(eng. True)*, bool vrijednost „Currently Attacking“ na neistinu *(eng. False)* te bool vrijednost „Can Attack“ na neistinu *(eng. False)* te time prekidamo napad i onemogućavamo kretanje i daljnje napadanje u trenutku kad je igrač napadnut.

Dalje provjeravamo blokira li igrač preko bool vrijednosti „Blocking“, ukoliko je vrijednost istina *(eng. True)* provjeravamo ima li igrač dovoljno izdržljivosti *(eng. Stamina)* kako bi blokirao napad, ako je izdržljivost veća od 0.1 uvjet je ispunjen te se započinje funkcija „F Check Blocking Direction“, inače se započinje „Apply Damage“ funkcija.



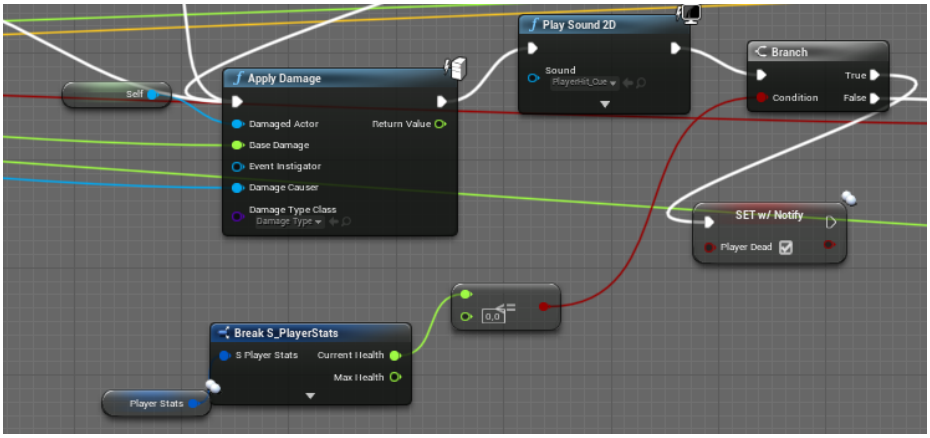
Slika 46: funkcija „F Check Blocking Direction“

Funkcija „F Check Blocking Direction“ nam uzima lokaciju našeg igrača preko „GetActorLocation (Target: self)“ i lokaciju neprijatelja („Damage Causer“) preko „GetActorLocation (Target: Actor)“ te pomoću matematičke formule vektora dobivamo 2 vektora i spajamo ih te time provjeravamo je li naš igrač usmjeren prema neprijatelju sa svojim štitom za vrijeme blokiranja.

Ukoliko igrač nije usmjeren prema neprijatelju za vrijeme primanja štete pokreće se „Apply Damage“ funkcija, a ako postoji povratna vrijednost *(eng. Return value)* igrač ne prima štetu već nastavlja dalje s „Multi Play Montage“ funkcijom koja pokreće animaciju posrtanja *(eng. Stagger)* modela Greystonea putem „Blocking Montage“ montaže animacije *(eng. Animation montage)* te nakon toga smanjuje izdržljivost *(eng. Stamina)* glavnog lika za 100.

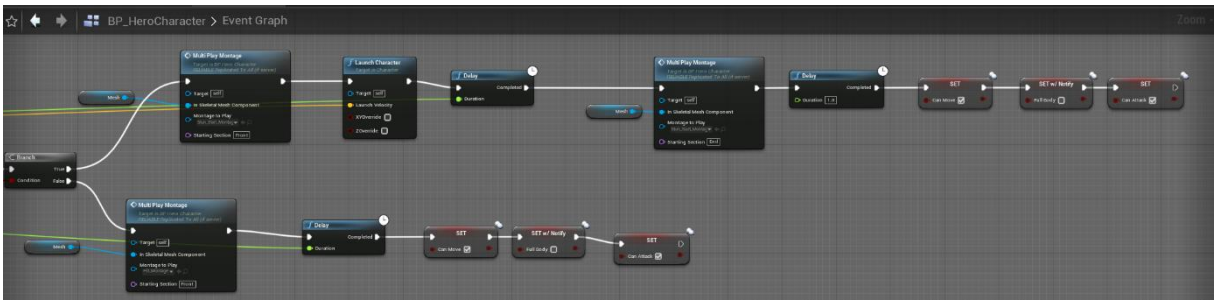
Nakon što prođe jedna sekunda, igrač se opet može pokretati i napadati jer postavljamo bool vrijednosti „Can Move“ i „Can Attack“ na istinu *(eng. True)*, a „Full Body“ postavljamo na neistinu *(eng. False)*.

Kao što smo prije već navodili ukoliko igrač ne uspije blokirat napad on prima štetu i pokreće se „Apply Damage“ funkcija.



Slika 47: funkcija „Apply Damage“

Pokrećemo zvuk udarca preko „Play Sound 2D“ „PlayerHit_Cue“ [7] te provjeravamo ukoliko igrač ima dovoljno životnih bodova *(eng. Health points)* kako bi primio štetu i preživio. Ukoliko igrač ima dovoljno životnih bodova granamo se na neistinitu *(eng. False)* granu, a ako je uvjet ispunjen postavljamo bool vrijednost „Player Dead“ na istinu *(eng. True)*.



Slika 48: funkcija „Apply Damage“, igrač živ, ali prima štetu

Sljedeći uvjet koji se provjerava je proslijeđen unutar događaja „Event Player Hit“ sa slike 45, a to je bool vrijednost „Knocked Down?“, float vrijednost „Stun Timer“ i vektor vrijednost „Impulse“.

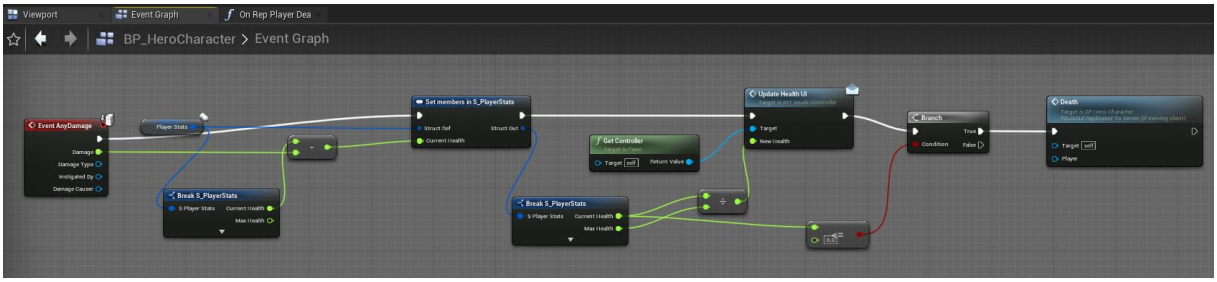
Ukoliko je bool vrijednost „Knocked Down?“ postavljena na istinu *(eng. True)* (gornje grananje) pokreće se montaža animacija *(eng. Animation montage)* preko funkcije „Multi Play Montage“,a to je „Stun_Start_montage“ animacija te bacamo našeg igrača preko „Launch Character“ funkcije kojoj prosljeđujemo vektor vrijednost „Impulse“ kao „Launch Veliocity“ vrijednost te onemogućimo liku kretanje za proslijeđenu float vrijednost „Stun“ kao vrijednost „Duration“ te opet funkcija za pokretanje animacije „Multi Play Montage“ gdje tek nakon jednu sekundu će igrač ponovo se moći pokretati i napadati.

Ukoliko je bool vrijednost „Knocked Down?“ postavljena na neistinu *(eng. False)* (donje grananje) pokreće se montaža animacija *(eng. Animation montage)* preko funkcije „Multi Play Montage“,a to je „Hit_montage“ animacija te se nakon što prođe „Delay“ funkcija zadana float varijablom „Stun“ kao „Duration“ igrač može ponovo kretati i napadati.

3.3.5.3. Primanje fatalne štete

Kao što smo nakon slike 47 napomenuli, kada nam igrač primi štetu koja će mu spustiti životne bodove *(Health points)* na nulu postavlja se bool vrijednost „Player Dead“ na istinu *(eng. True)*.

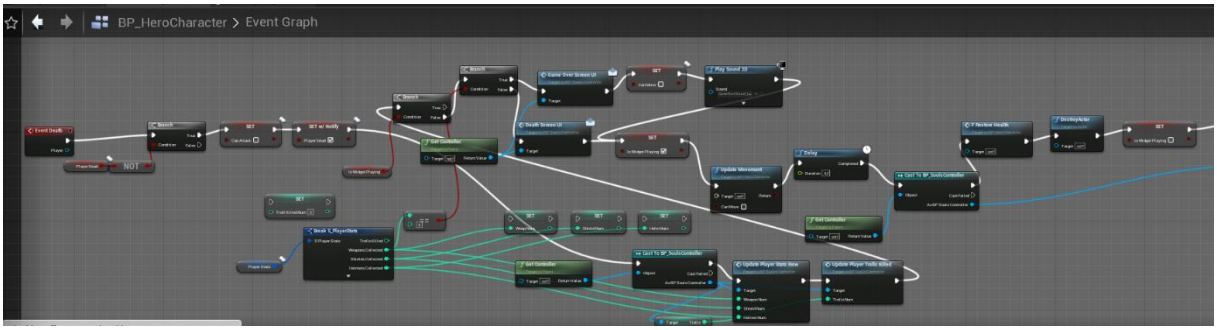
Dapače kad god naš lik primi neku štetu pokreće se događaj putem servera „Event AnyDamage“ te se putem njega igraču smanjuju životni bodovi *(eng. Health points)*.



Slika 49: server događaj „Event AnyDamage“

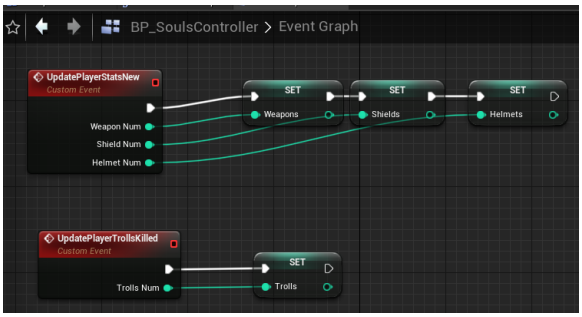
„Event AnyDamage“ uzima trenutne životne bodove igrača te ih smanjuje za proslijeđenu vrijednost štete koju prima te postavlja novu float vrijednost „Current Health“ te opet strukturu „S_PlayerStats“ razdvaja kako bi dobio postotak da ažurira traku za prikaz životnih bodova sa slike 29.

Nadalje provjerava ako su igraču životni bodovi *(eng. Health points)* manji ili jednaki 0 te ako jesu pokreće događaj *(eng. Event)* „Death“.



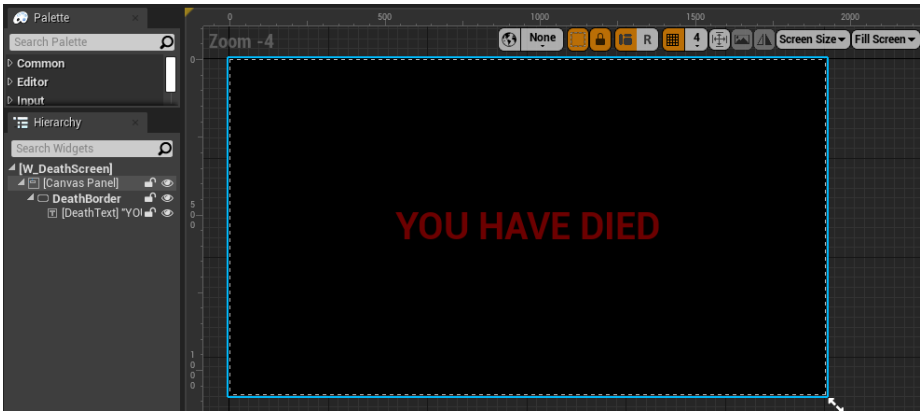
Slika 50: događaj „Death“

Posljednji put se provjerava je li bool vrijednost „Player Dead“ postavljena na istinu *(eng. True)* te ako nije postavljamo ju na istinu, a „Can Attack“ postavljamo na neistinu *(eng. False)*. Dalje pokrećemo „Cast to BP_SoulsController“ tako da možemo pospremiti float vrijednosti, odnosno broj oružja, štitova i šljemova koje smo pokupili preko strukture „S_PlayerStats“ prije nego što uništimo našeg glumca *(eng. Actor)* te pokrećemo te događaje u „Bp_SoulsControlleru“.



Slika 51: „UpdatePlayerStatsNew“ i „UpdatePlayerTrollsKilled“

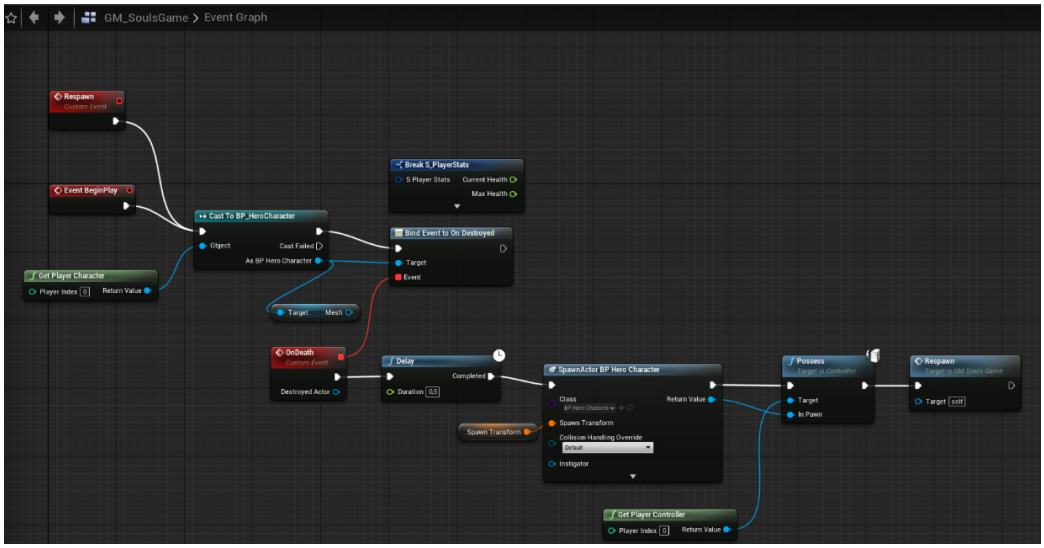
Nadalje kao nastavak koda sa slike 50 provjeravamo je li igrač pokupio svih šest oružja, ukoliko igrač još nije pokupio svih šest oružja pokreće se funkcija „Death Screen UI“, koja pomoću „Background Fade Animacije“ poziva widget „W_DeathScreen“ na ekran igrača.



Slika 52: „W_DeathScreen“

Dalje postavljamo bool vrijednost „Is Widget Playing“ na istinu *(eng. True)* te pokrećemo funkciju „Update Movement“ tako da onemogućavamo kretanje igraču te pomoću funkcije „Delay“ zaustavljamo izvođenje koda na sljedećih 4,2 sekunde te tek nakon toga uništavamo glumca *(eng. Actor)* i postavljamo bool vrijednost „Is Widget Playing“ na neistinu *(eng. False)*.

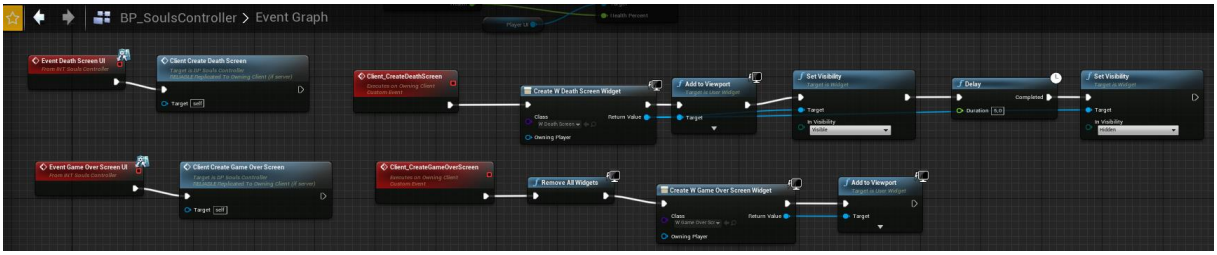
Nakon što nam je glumac uništen pomoću „DestroyActor“ funkcije pokreće nam se funkcija „Bind Event to On Destroyed“ iz nacrta *(eng. Blueprint)* „GM_SoulsGame“.



Slika 53: „GM_SoulsGame“

Funkcija „Bind Event to On Destroyed“ preuzima model i našeg lika iz „BP_HeroCharacter“ nacrta te pokreće događaj *(eng. Event)* „OnDeath“ koji nakon pola sekunde ponovno stavlja našeg glumca *(eng. Actor)* u svijet preko „SpawnActor BP Hero Character“ funckije te je potrebno staviti transform varijablu za veličinu našeg modela *(eng. Mesh)*. Dalje pokrećemo funkciju „Possess“ koja naš „Player Controller“ postavlja u kontrolu glumca *(eng. Actor)* iz „BP_HeroCharacter“ nacrta *(eng. Blueprint)* te nakon toga pokreće „Respawn“ funkciju sa slike 53.

Vratimo se na sliku 50 te ponovo provjeravamo je li igrač pokupio svih šest oružja, ukoliko je igrač pokupio svih šest oružja pokreće se funkcija „Game Over Screen UI“ za razliku od „Death Screen UI“ te se onemogućava kretanje igraču i pokreće glazba koja simbolizira neuspješan kraj igre preko „Play Sound 2D“ funkcije koja pokreće „GameOverSound_Cue“ [8].

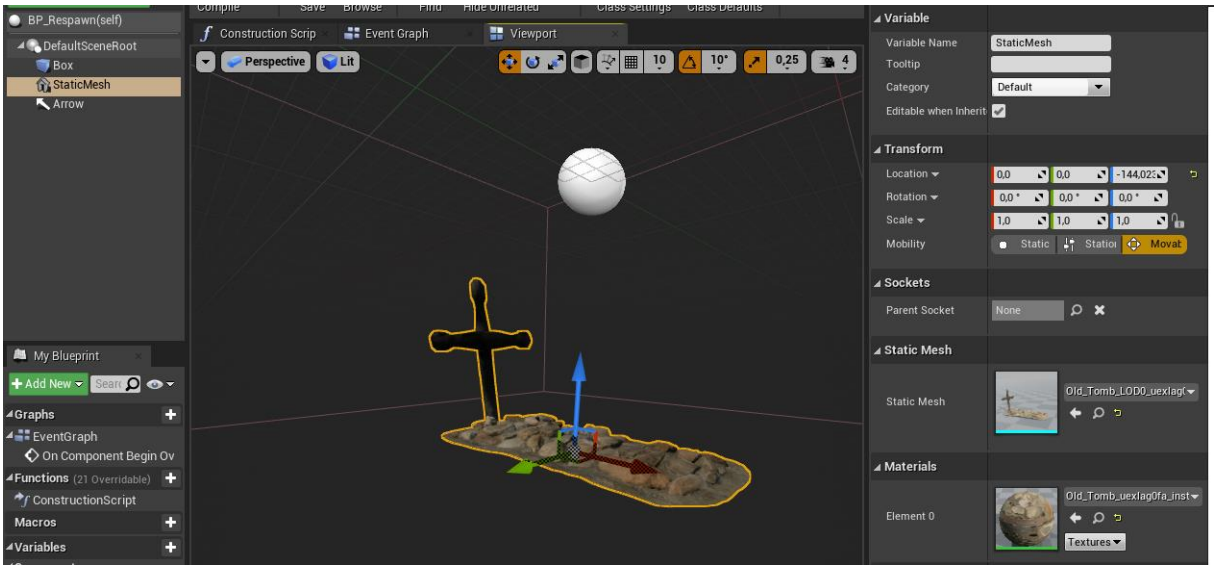


Slika 54: „BP_SoulsController“ i događaji „Event Death Scren UI“ i „Event Game Over Screen UI“

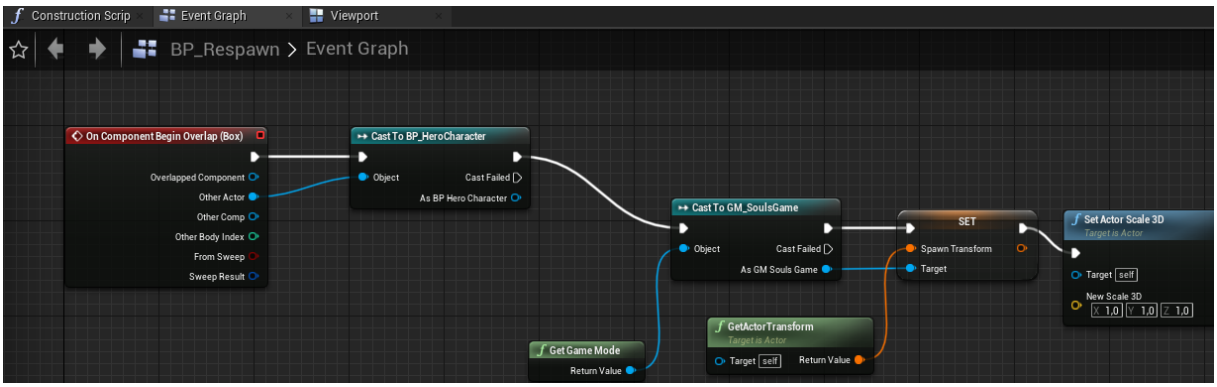
Kada se pozove događaj „Event Death Screen UI“, za razliku od „Event Game Over Screen UI“ događaja ne nestaje, nakon što se pozove na ekran igrača, to je zato što želimo da taj widget ostane budući da je igrač neuspješno završio videoigru, a kada se pokreće „Event Game Over Screen UI“ želimo putem widgeta obavijestiti igrača da je umro te nakon nekog vremena maknuti widget i omogućiti igraču da dalje nastavi s igrom.

3.3.5.4. Oživljavanje igrača

Kako bi odredili lokaciju gdje će se naš igrač oživljavati potrebno je napraviti novi nacrt *(eng. Blueprint)* „BP_Respawn“ s modelom *(eng. Mesh)* groba [9] pomoću kojeg ćemo definirati te lokacije na kojoj se naš igrač oživljava.

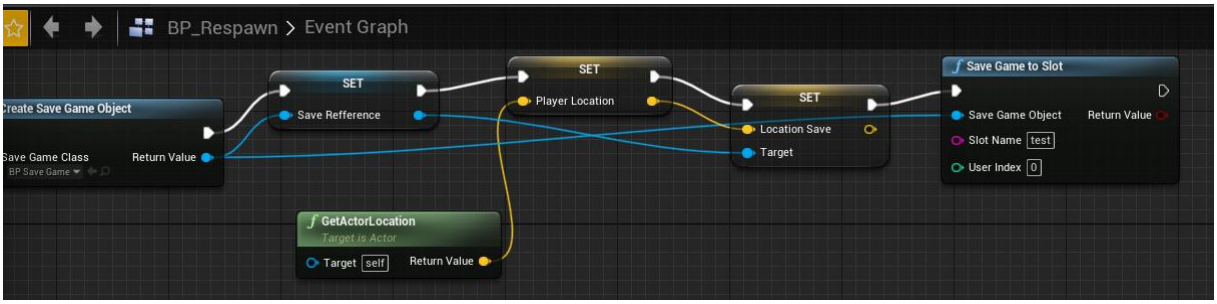


Slika 55: „BP_Respawn“



Slika 56: „BP_Respawn“

U trenutku kada sfera modela *(eng. Mesh)* našeg lika prijeđe preko sfere modela *(eng. Mesh)* za „BP_Respawn“ započinje se događaj „On Component Begin Overlap (Box)“ koji se prvo baca na „BP_HeroCharacter“ putem „Cast To BP_HeroCharacter“ funkcije te se zatim baca na „GM_SoulsGame“ putem „Cast To GM_SoulsGame“ funkcije. Nadalje je potrebno postavit od prije spomenutu „Spawn Transform“ transform varijablu iz „GM_SoulsGame“ pod veličinu *(eng. Scale)* x = 1, y = 1 i z = 1 putem „Set Actor Scale 3D“ funklcije jer model *(eng. Mesh)* nakon stvaranja bude puno veći nego što je namijenjeno.



Slika 57: „Create Save Game Object“

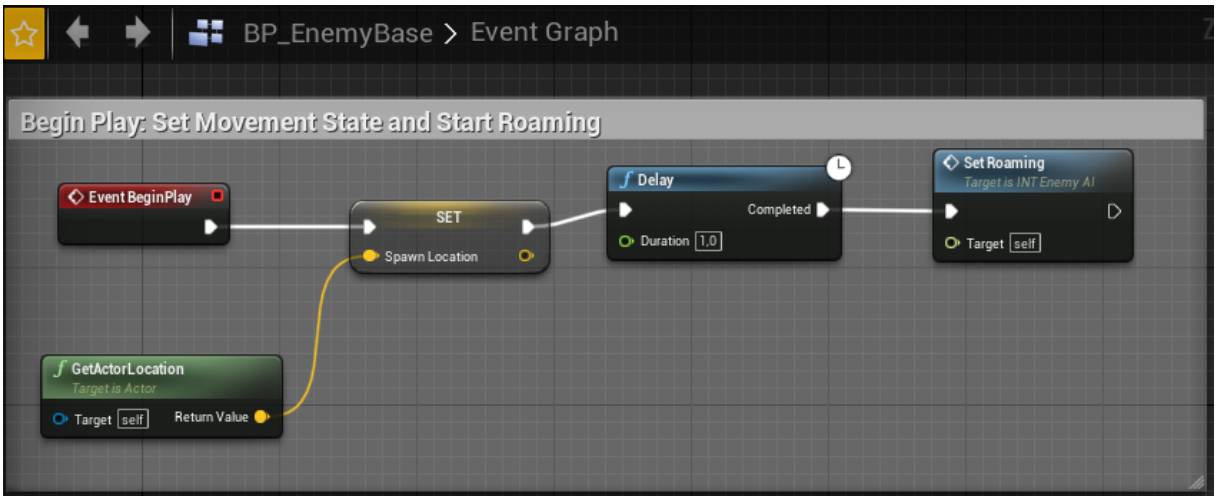
Funkcija „Create Save Game Object“ spremamo lokaciju zadnje kontrolne točke *(eng. Checkpoint)* na kojoj je igrač bio tako da sprema lokaciju igrača na lokaciju kontrolne točke *(eng. Checkpoint)* te zatim pokreće „Save Game to Slot“ funkciju. Stoga kad igrač umre novi glumac *(eng. Actor)* će se stvoriti na lokaciji gdje je bila zadnja kontrolna točka *(eng. Checkpoint)*.

3.3.6. Osposobljavanje neprijatelja i njegove umjetne inteligencije

Za razliku od našeg „BP_HeroCharacter“ nacrta *(eng. Blueprint)* neprijatelji će svi imati prvo jedan zajednički nacrt *(eng. Blueprint)* pomoću kojeg ćemo razvijati sve mehanike zajedničke neprijateljima, a to je „BP_EnemyBase“, sve napredne mehanike koje daju jedinstvenost neprijatelja radit ćemo u njima zasebnim nacrtima *(eng. Blueprint)*.

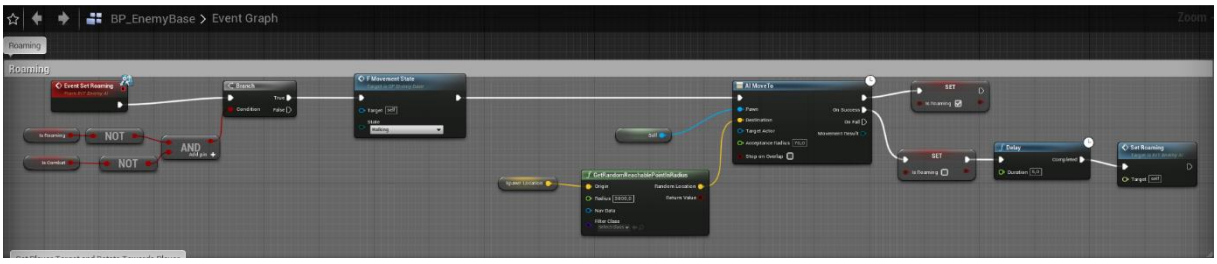
3.3.6.1. Jednostavna umjetna inteligencija za sve neprijatelje

Pri samom početku videoigre neprijatelji započinju svoj prvi akt umjetne inteligencije.



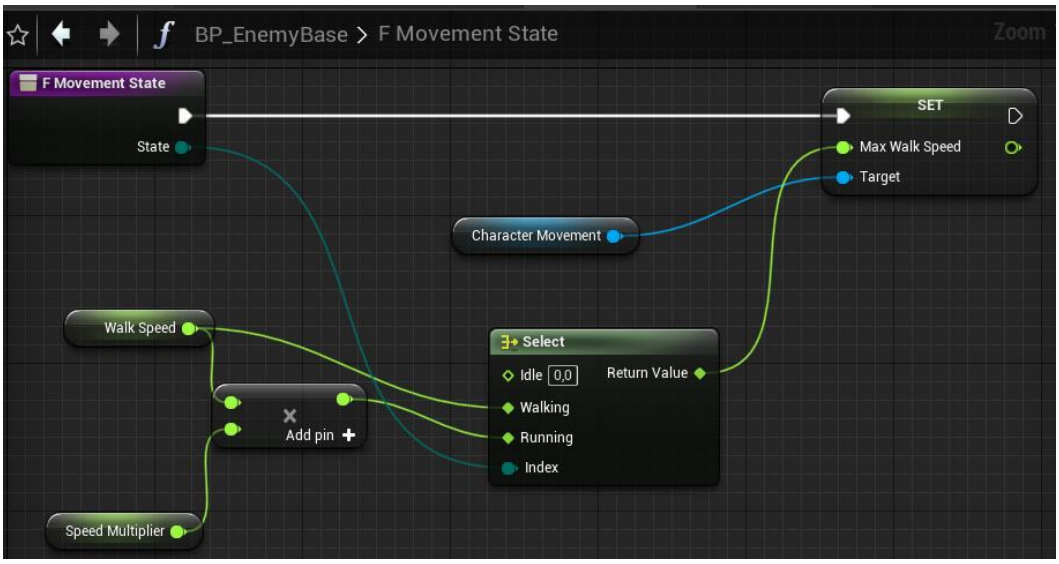
Slika 58: „Event BeginPlay“

Početkom igre započinje se „Event BeginPlay“ događaj kojim se zapisuje mjesto stvaranja u vektor varijablu „Spawn Location“ preko „GetActorLocation“ funkcije te nakon jedne sekunde poziva se „Event Set Roaming“ događaj.



Slika 59: „Event Set Roaming“

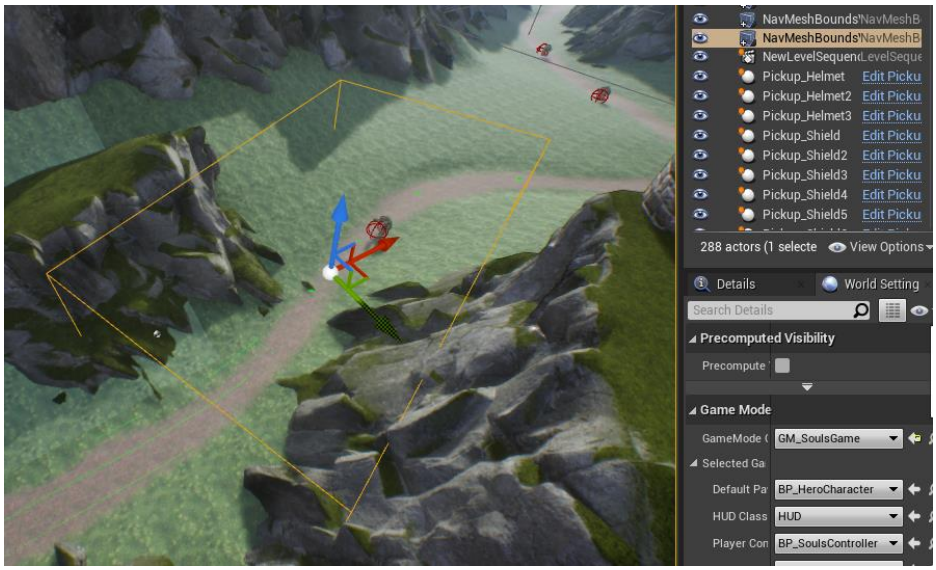
Prije nego što se događaj nastavi provjeravamo da nije naš glumac *(eng. Actor)* iz „BP_EnemyBase“ (dalje u radu spominjan kao „neprijatelj“) već u stanju lutanja *(eng. Roaming)* ili da nije u borbi *(eng. Combat)* s našim igračem, ako je uvjet zadovoljen nastavlja dalje s „F Movement State“ funkcijom.



Slika 60: „F Movement state“

Funkcijom „F Movement State“ definiramo zadanu brzinu kretanja preko float varijable „Walk Speed“ te ju množimo s float varijablom „Speed Multiplier“ kako bi dobili brzinu trčanja koju će neprijatelj pozivati kada uoči našeg igrača.

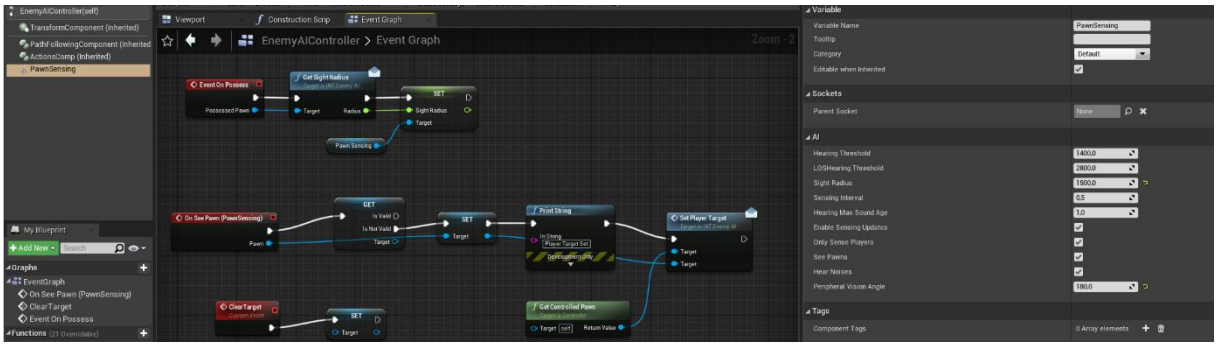
Nadalje, kao nastavak koda sa slike 59 dobivamo slučajnu točku u definiranoj okolini neprijatelja preko „GetRandomReachablePointInRadius“ funkcije i proslijeđene „Spawn Location“ vektor varijable te pomoću „AI Move To“ funkcije pomičemo našeg neprijatelja unutar omeđene zone koje ga stavimo, a ta omeđena zona u Unreal Engine 4 zove se „NavMeshBoundsVolume“.



Slika 61: „NavMeshBoundsVolume“

Nakon što završi „AI Move To“ funkcija sa slike 59 postavlja se stanje neprijatelja vezano za lutanje *(eng. Roaming)* na neistinu *(eng. False)* te nakon pet sekundi ponovo započinje događaj „Set Roaming“.

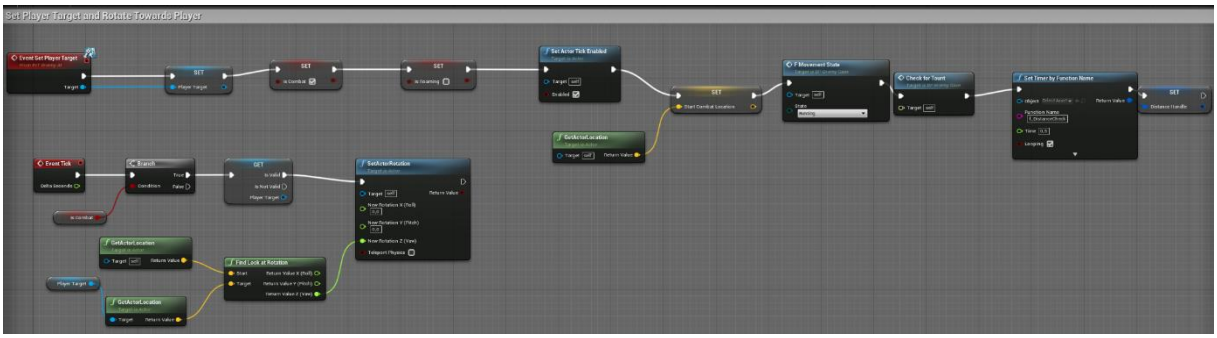
Paralelno dok se odvija događaj „Event Set Roaming“ važno je da napravimo novi nacrt *(eng. Blueprint)* „EnemyAIController“ kojim ćemo omogućavati našem neprijatelju da detektira igrača kada mu se dovoljno približi.



Slika 62: „EnemyAIController“ i „PawnSensing“

Čim se videoigra pokrene i „EnemyAIController“ posjeduje neprijatelja započinje se „Event On Possess“ događaj kojem prosljeđujemo „Pawn Sensing“ funkcionalnosti koje vidimo u desnom dijelu slike 62 pod „AI“ odjeljkom. Ono što je nama bitno i što koristimo za videoigru je „Sight Radius“ koji je postavljen na 1500 i to je radius unutar kojeg naš neprijatelj može vidjeti igrača, „Sensing Interval“ koji se poziva dvaput u sekundi znaci da svake 2 sekunde pokreće se funkcija kojom pomoću „Pawn Sensing“ pokušavamo detektirati igrača te „Peripheral Vision“, odnosno periferni vid koji nam je postavljen na 180 stupnjeva.

Ukoliko „Pawn Sensing“ detektira igrača pokreće se „On See Pawn (PawnSensing“ događaj koji prosljeđuje „Pawn“ metu, odnosno igrača, te ga postavlja kao „Target“ i pokreće događaj „Set Player Target“ unutar „BP_EnemyBase“ nacrta.



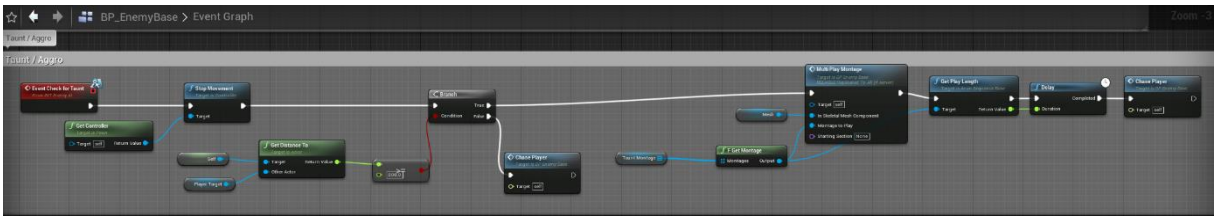
Slika 63: „Event Set Player Target“ događaj

Događaj „Event Set Player Target“ poprima vrijednost „Target“ koja mu je proslijeđena preko „Pawn Sensing“ sa slike 62 te se unutar „BP_EnemyBase“ ta varijabla tipa actor „Player Target“ koristi za informacije za neprijatelja vezane za igrača.

Dalje postavljamo bool varijablu „Is Combat“ na istinu *(eng. True)*, a „Is Roaming“ varijablu na neistinu *(eng. False)* kako bi započeli fazu borbe i obustavili lutanje.

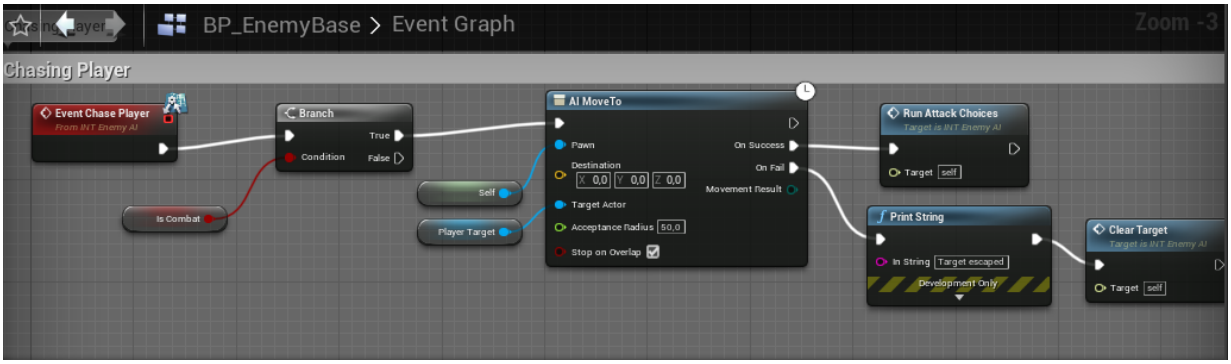
Započinjemo „Set Actor Tick Enabled“ što u jako kratkim intervalima sekunde ponavlja „Event Tick“ događaj unutar kojeg provjeravamo je li naš neprijatelj u borbi s igračem, odnosno ako je „Is Combat“ postavljen na istinu *(eng. True)* te s obzirom na lokaciju neprijatelja i lokaciju igrača okreće neprijatelja prema igraču po z-osi preko „SetActorRotation“ funkcije.

Nakon rotacije prema igraču, neprijatelj se postavlja u stanje „Running“, odnosno trčanja, preko „F Movement State“ funkcije objašnjene na slici 59 te pokreče događaj „Check for Taunt“.



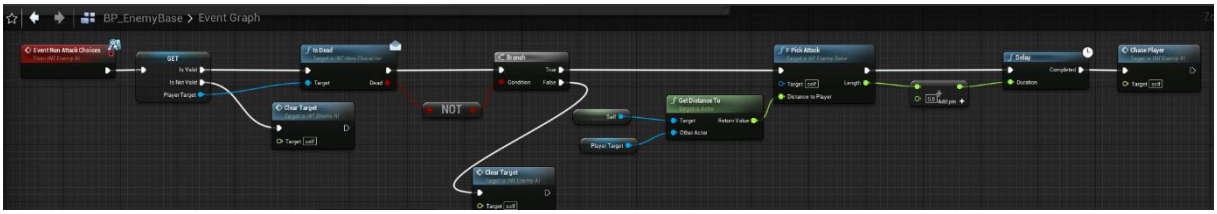
Slika 64: „Event Check for Taunt“ događaj

Događaj „Event Check for Taunt“ započinje „Stop Movement“ funkciju koja obustava kretanje neprijatelja te provjerava udaljenost igrača između neprijatelja i igrača. Ukoliko je igrač dovoljno daleko od neprijatelja, odnosno float vrijednost veća ili jednaka od 800 neprijatelj započinje „Multi Play Montage“ događaj kojim pokreće svoju animaciju kojom daje do znaka da je primijetio igrača te s obzirom na trajanje animacije toliko dugo se zaustavlja program te nakon završetka animacije pokreće „Chase Player“ događaj.



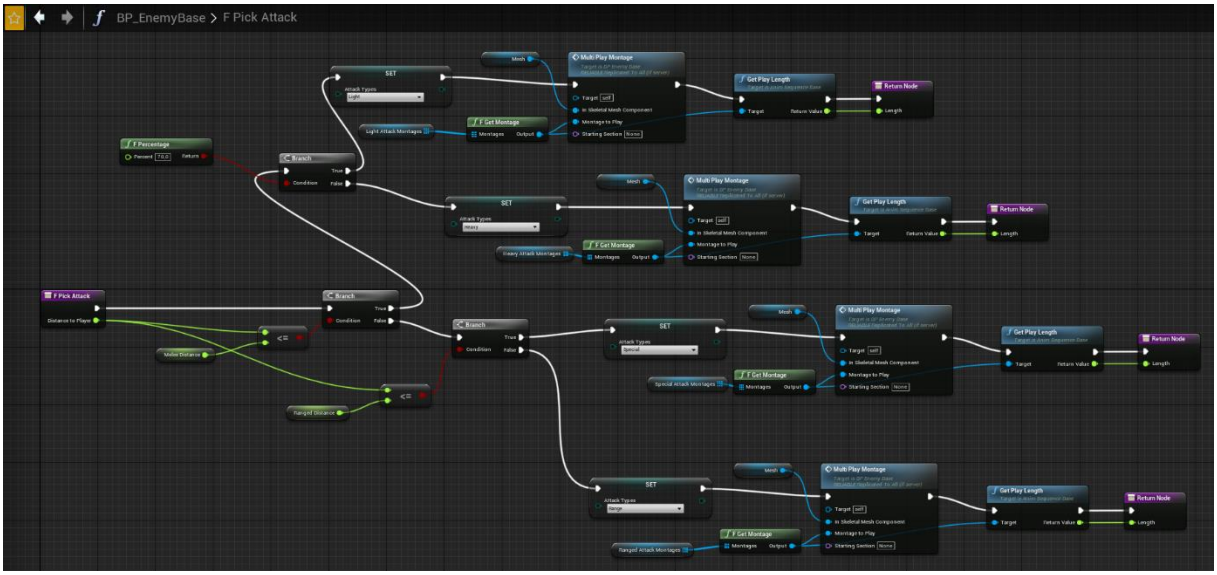
Slika 65: „Event Chase Player“ događaj

Događaj provjerava je li neprijatelj i dalje u borbi te pokreće „AI Move To“ funkciju koja usmjeruje njegovu putanju kretanja prema našem igraču te ukoliko uspije doći do lokacije našeg lika pokreće se „Run Attack Choices“ događaj.



Slika 66: „Event Run Attack Choices“ događaj

Prvo provjeramo unutar događaja „Event Run Attack Choices“ ukoliko meta još uvijek postoji, te ako postoji provjeravamo je li igrač mrtav preko „Is Dead“ funkcije koja je naslijeđena iz sučelja *(eng. Interface)* „INT_HeroCharacter“ koja sadrži bool vrijednost za istinu *(eng. True)* ukoliko je naš igrač mrtav te ako nije mrtav događaj se nastavlja dalje do funkcije „F Pick Attack“ koja zapisuje udaljenost igrača od neprijatelja radi vrste napada koje neprijatelj može koristiti.

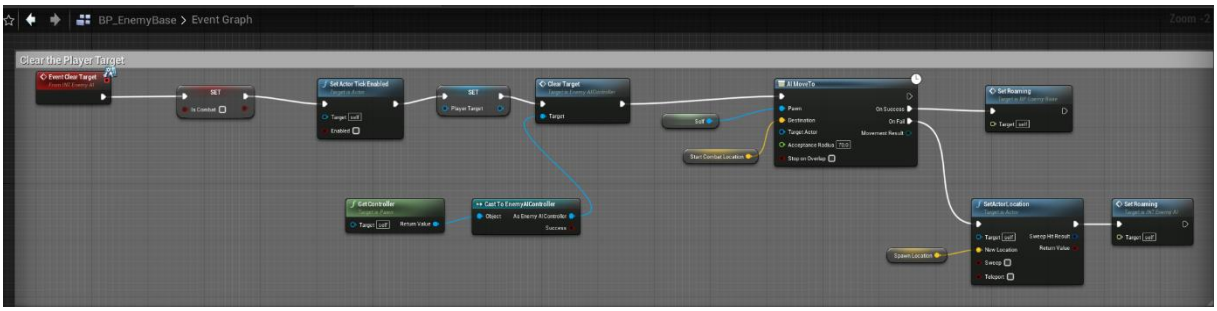


Slika 67: funkcija „F Pick Attack“

Funkcija prvo provjerava udaljenost te s obzirom na „Distance to Player“ funkcija „F Pick Attack“ se grana na napade na blizinu i na napade na daljinu.

Ukoliko je igrač unutar „Melee Distance“ pokreće se odabir između 2 napada s obzirom na vjerojatnost izvođenja napada, 70% šanse je da se izvede „Light“ napad te preostalih 30% da se izvede „Heavy“ napad. Ukoliko se igrač nalazi između „Melee Distance“ i „Ranged Distance“ izvodi se „Special“ napad, a ako je igrač dovoljno udaljen od neprijatelja funkcija pokreće „Range“ napad. Svaki od napada pokreće svoju animaciju vezanu za napad te zapisuje njeno trajanje. Objašnjenje napada se nastavlja u kasnijem odjeljku ovog rada pod naslovom „Napadi za Troll vrstu neprijatelja“.

Nakon što funkcija „F Pick Attack“ završi, zaustavlja se izvršavanje događaja na trajanje duljine animacije plus pola sekunde te nakon toga ponovo započinje „Chase Player“ događaj sve dok se ne pozove „Clear Target“ događaj.

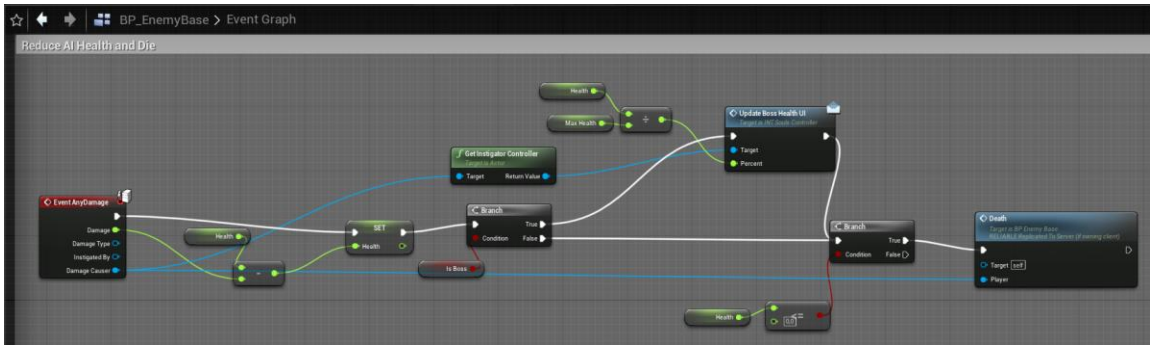


Slika 68: „Event Clear Target“ događaj

Ukoliko se dogodi da neprijatelj ubije igrača pokreće se događaj „Event Clear Target“, kojim se postavlja „Is Combat“ na neistinu *(eng. False)* te se zaustavlja funkcija „Set Actor Tick Enabled“, miče se vrijednost „Player Target“ te se pokreće „Clear Target“ događaj iz „EnemyAIController“ nacrta *(eng. Blueprint)* prikazan u slici 61. Nakon toga, neprijatelja pomičemo na mjesto gdje je započeo borbu s igračem te ukoliko ne može doći do zakazanog mjesta vraćamo ga na mjesto gdje se i prvi put stvorio preko „Spawn Location“, nakon što se bilo koji od smjerova funkcije „AI Move To“ izvrši neprijatelj ponovo započinje „Set Roaming“ događaj.

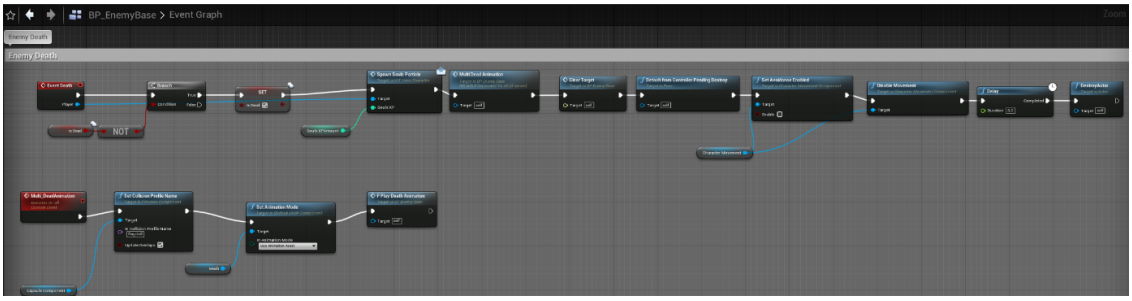
3.3.6.2. Primanje štete kod neprijatelja

Isto kao i za igrača, potrebno je osposobiti događaje za primanje štete te što se dogodi s neprijateljem ako mu životni bodovi padnu na nulu.



Slika 69: „Event AnyDamage“ događaj

Kad god neprijatelj primi štetu automatski se pokreće događaj „Event AnyDamage“ kojim uzimamo trenutnu vrijednost životnih bodova neprijatelja iz float varijable „Health“ te ju umanjujemo za vrijednost primljene štete i postavljamo novu vrijednost životnih bodova. Ukoliko je vrsta neprijatelja glavni i konačni neprijatelj onda pokrećemo funkciju koja će ažurirat njegovu traku životnih bodova koja je prikazana na ekranu igrača, ali detaljnije o tome u kasnijem dijelu rada. Na kraju provjeravamo jesu li životni bodovi neprijatelja ispod ili jednaki nuli te ako jesu pokrećemo „Death“ događaj.



Slika 70: „Event Death“ događaj

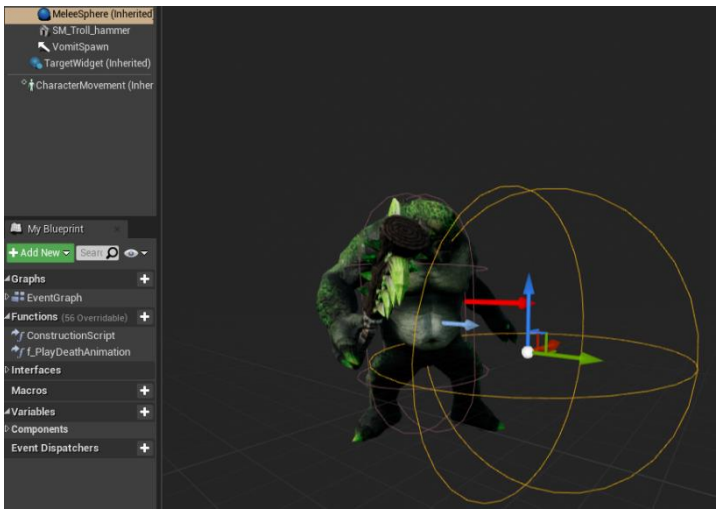
Događaj „Event Death“ prvo provjerava ukoliko je igrač već mrtav te ako nije postavlja ga kao mrtvog preko „SET“ funkcije za bool vrijednost „Is Dead“, pokreće „Spawn Souls Particle“ događaj o kojem ćemo više u naslovu „Interakcija igrača i neprijatelja sa svijetom“, zatim pokreće „Multi Dead Animation“ događaj koji postavlja koliziju neprijatelja na „Ragdoll“ preko „Set Collision Profile Name“ te pokreće funkciju „F Play Death Animation“ kojom započinju animaciju koja im je zadana.

3.3.6.3. Napadi za Troll vrstu neprijatelja

Kao što smo prije naveli kod igrača, postoje događaji koji se izvode usred izvođenja animacije, a to su „AnimNotifyState“ događaji.

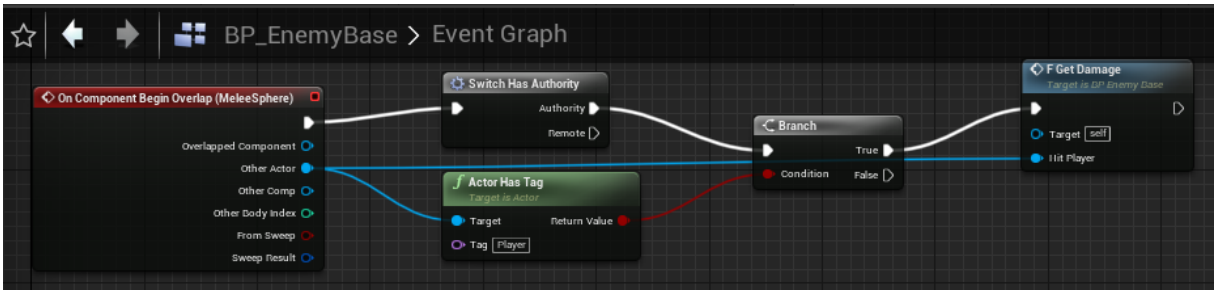
Slika 71: „Notify State“ za „Troll_LightAttackA“ vrstu napada

U trenutku kada započne „Notify State (Begin)“ sfera koja služi za koliziju za sferom igrača postavlja se njena kolizija na omogućeno *(eng. Enabled)* u nacrtu „Enemy_Troll“



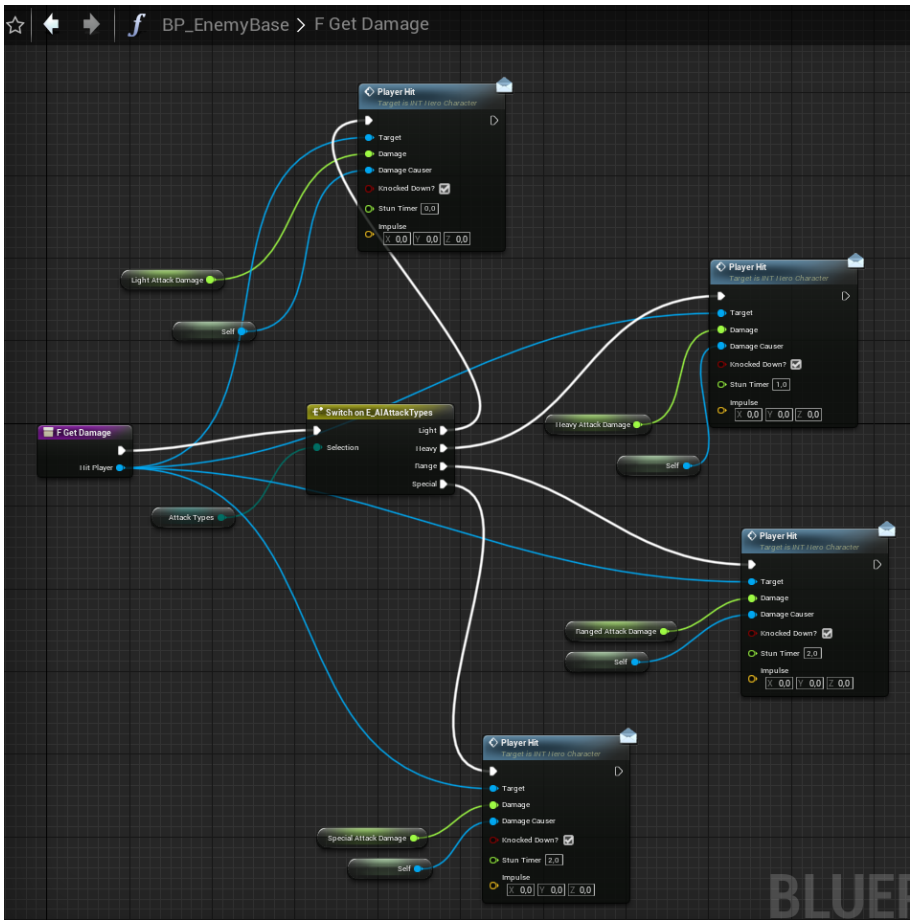
Slika 72: „MeleeSphere“ u „Enemy_Troll“ nacrtu

Ukoliko ta sfera ima koliziju s igračem pokreće se „On Component Begin Overlap (MeleeSphere)“ događaj iz „BP_EnemyBase“ nacrta *(eng. Blueprint)*.



Slika 73: „On Component Begin Overlap (MeleeSphere)“ događaj

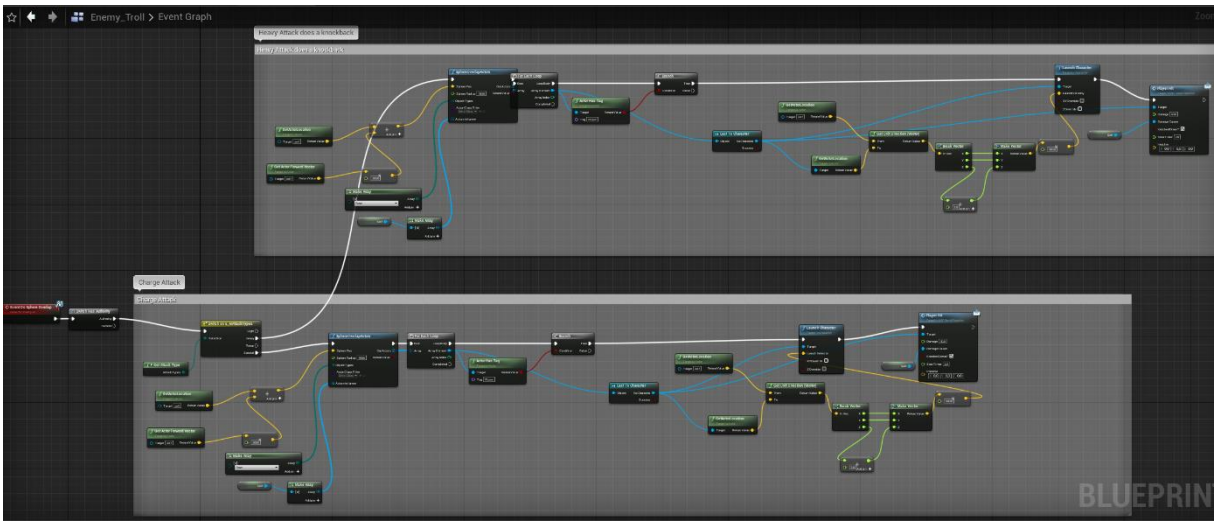
„On Component Begin Overlap (MeleeSphere)“ događaj pokreće funkciju „F Get Damage“.



Slika 74: „F Get Damage“ funkcija

Funkcija „F Get Damage“ s obzirom na enumeraciju „E_AIAttackTypes“ zapisuje vrijednost float varijable „Light/Heavy/Ranged/Special Damage“ koju dalje prosljeđujemo u „Player Hit“ događaj *(eng. Event)* sa slike 45.

No, primanje štete nije jedina stvar koja ometa igrača, tu još postoji stanje ošamućenosti *(eng. Stun state)* te daljina za koju napad baci igrača preko varijable „Impulse“. Budući da su ovo jedinstvene mehanike za svakog neprijatelja potrebno ih je napraviti u zasebnom nacrtu za svakog neprijatelja, u slučaju troll neprijatelja u „Enemy_Troll“ nacrtu *(eng. Blueprint)*.

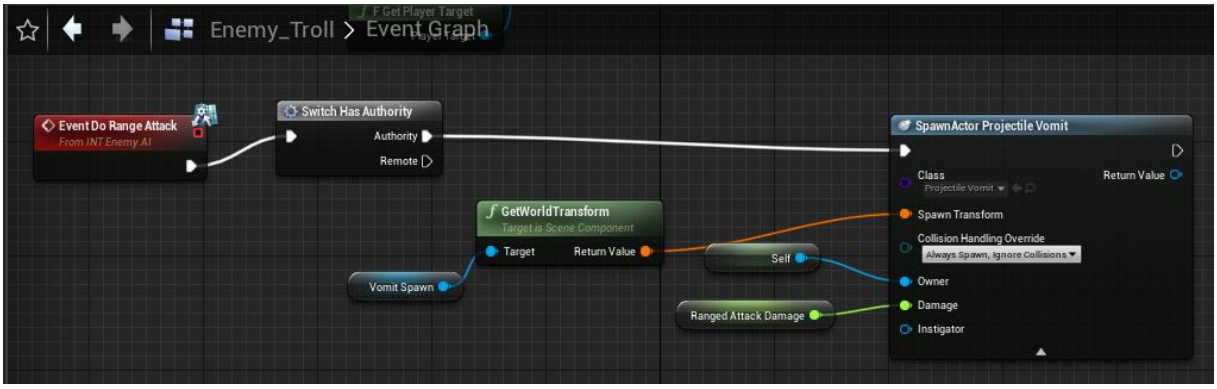


Slika 75: „Event Do Sphere Overlap“ događaj

„Event Do Sphere Overlap“ s obzirom na vrstu napada koju izvede „Heavy“ ili „Special“ računa poziciju igrača s pozicijom neprijatelja te s obzirom na dobivene vektor vrijednosti od „GetActorLocation“ (igrač) i „GetActorLocation“ (self) dobiva smjer vektora koji se prodire od neprijatelja do igrača te razdvajanjem vektora s „Break Vector“ funkcijom dodajemo z-osi plus jedan na njenu float varijablu te ponovo pravimo vektor preko „Make Vector“ funkcije.

Za svakog glumca *(eng. Actor)* s kojim sfera ima koliziju provjerava prvo je li glumac *(eng. Actor)* s kojim ima koliziju igračev glumac odnosno „BP_HeroCharacter“ te ako je pokreće „Launch Character“ funkciju kojoj pridodajemo izračunatu „Launch Velocity“ vektor varijablu kako bi nam se simulirano lansiranje igrača kao uzrok napada neprijatelja, a na kraju prelazimo u otprije poznat „Player Hit“ događaj sa zadanom štetom, tko uzrokuje štetu te vremenom ošamućenosti igrača.

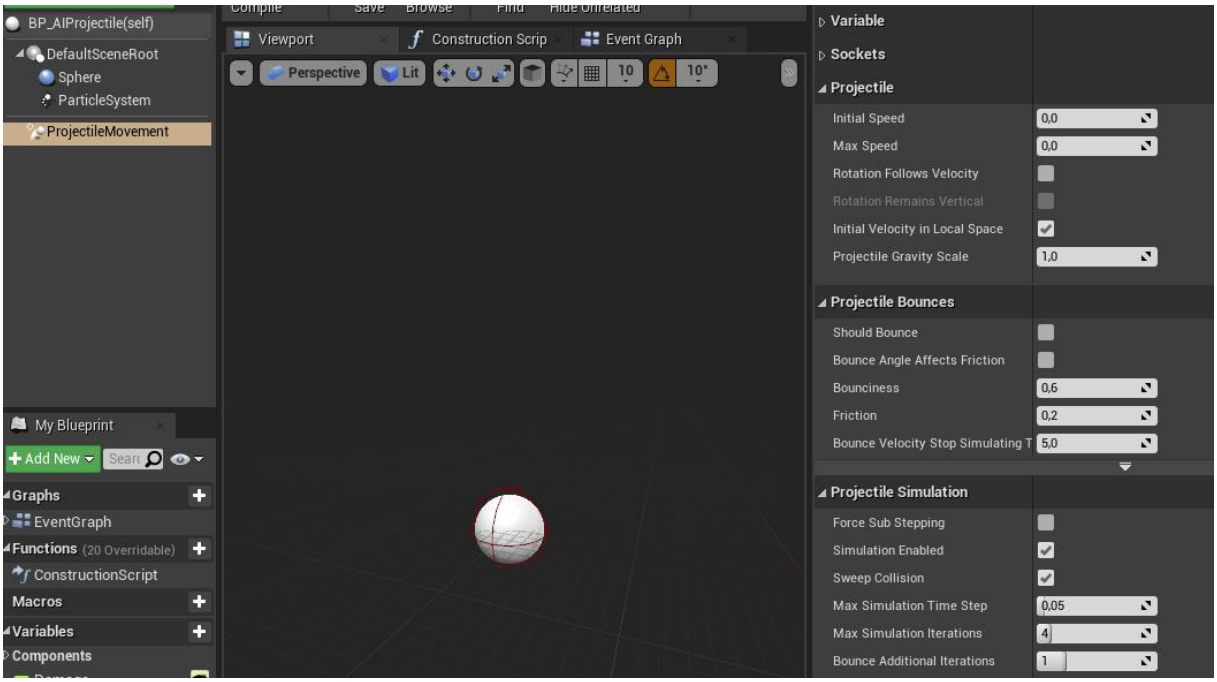
Još jedan napad koji moramo odvojeno definirati je napad na daljinu *(eng. Range attack)* koji troll neprijatelj baca kada je igrač dovoljno daleko od njega.



Slika 76: „Event Do Sphere Overlap“ događaj

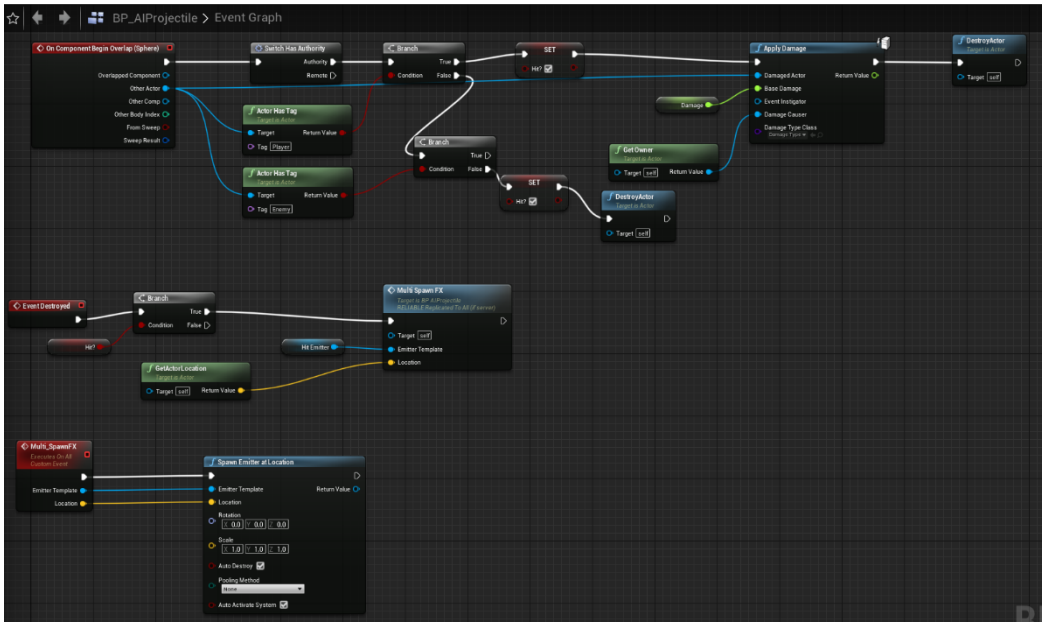
3.3.6.4. Projektil napad za Troll vrstu neprijatelja

Kada se dogodi poziv na događaj „Event Do Range Attack“ pokrećemo kroz nacrt *(eng. Blueprint)* „Enemy_Troll“ „SpawnActor ProjectileVomit“ funkciju, no ta funkcija samo stvara projektil u našoj igrici, svu logiku iza projektila moramo zasebno definirati u odvojenom nacrtu *(eng. Blueprint)* „BP_AIProjectile“.



Slika 77: „BP_AIProjectile“

Unutar „BP_AIProjectile“ postavljamo „ProjectileMovement Component“ koji služi za simulaciju letanja projektila, odnosno pokreće ga u našoj videoigri.



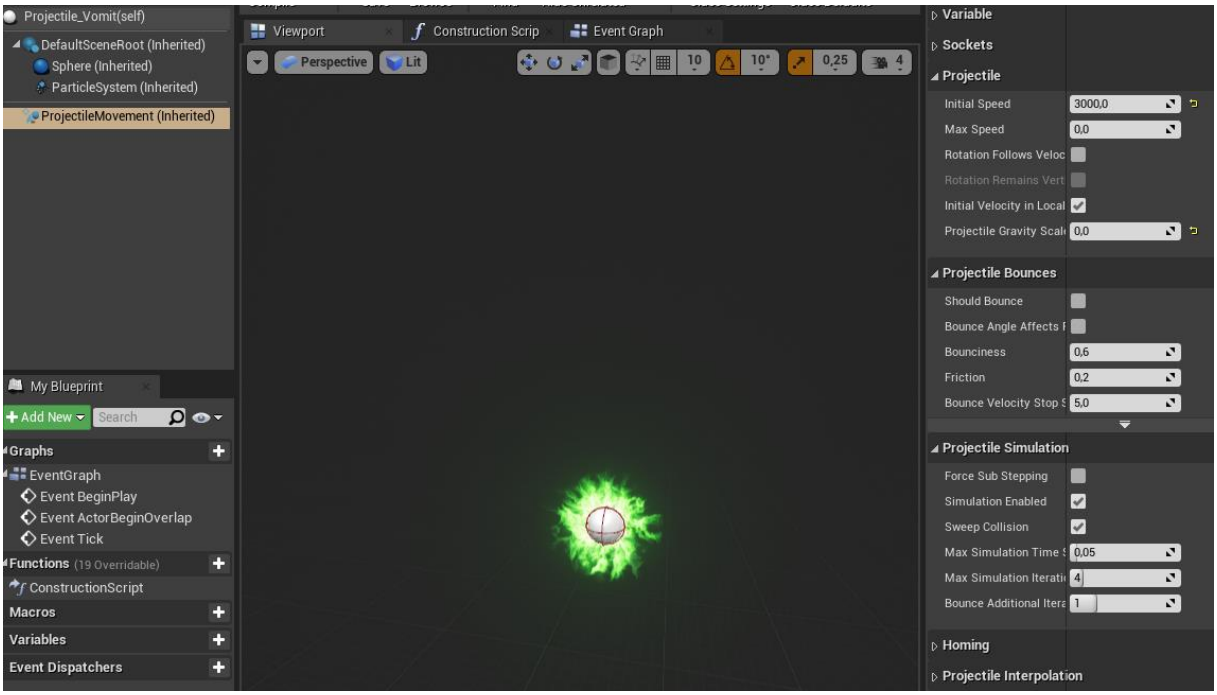
Slika 78: „BP_AIProjectile“

U nacrtu *(eng. Blueprint)* „BP_AIProjectile“ pozivamo događaj „On Component Begin Overlap (Sphere)“ kada se desi kolizija sa sferom našeg projektila s bilo čime drugim.

Provjeravamo je li projektil udario u igrača preko „Actor Has Tag“ funkcije te ako je postavljamo bool varijablu „Hit?“ na istinu *(eng. True)* i pozivamo „Apply Damage“ funkciju te zatim uništavamo projektil s „DestroyActor“ događajem. Ukoliko projektil ne pogodi igrača već neprijatelja ništa se ne dešava, ali ako promaši neprijatelja i pogodi nešto drugo kao što je teren mape onda se također poziva „DestroyActor“ događaj.

Događaj „Destroyed“ poziva se ako se projektil uništi te ukoliko je projektil uništen pokreće „Multi Spawn FX“ događaj na lokaciji gdje je bio u trenutku kada se uništio. Događaj „Multi_SpawnFX“ pokreće „Spawn Emitter at Location“ što je vizualni efekt koji se desi s projektilom kada se pokrene funkcija „Spawn Emitter at Location“.

„BP_AIProjectile“ nam sadrži svu logiku projektila te iz tog nacrta radimo „Child Blueprint Class“ nacrt *(eng. Blueprint)* za svaki novi projektil koji želimo koristiti, u ovom slučaju treba nam projektil za Trollov napad te pravimo nacrt *(eng. Blueprint)* „Projectile_Vomit“.



Slika 79: „Projectile_Vomit“ nacrt

Unutar nacrta *(eng. Blueprint)* „Projectile Vomit“ pod „ProjectileMovement (Inherited)“ definiramo brzinu projektila u „Projectile“ odjeljk, „Initial Speed“ postavljamo na 3000 te to predstavlja brzinu putovanja projektila, a „Bounce Velocity Stop Simulating Threshold“ na 5 što znači da nakon pet sekundi projektil će sam sebe uništiti.

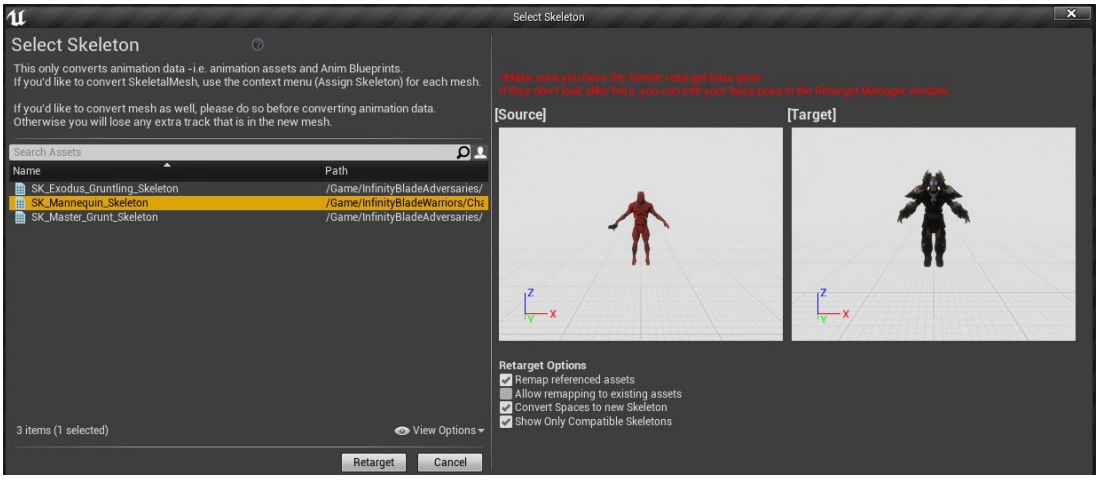
„ParticleSystem (Inherited)“ koristi vizualni efekt „P_Vomit_Projectile“ [10].

3.3.7. Forsaken Ogre King vrsta neprijatelja

„Forsaken Ogre King“ kojeg ćemo oslovljavati kao „glavni neprijatelj“ potrebno je kao i sa Trollom prvo pronaći modele *(eng. Mesh)* [12], oružje [12] i animacije[13] koje će uporabljivati prije nego što počnemo razvijati logiku za njegovu Umjetnu Inteligenciju.

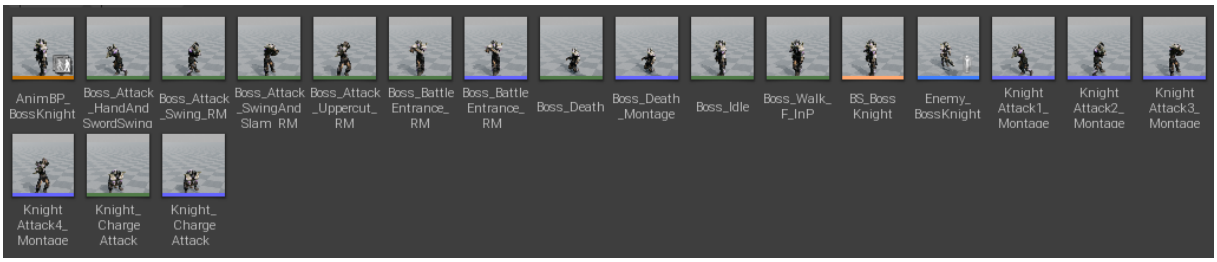
Slika 80: Model glavnog neprijatelja

Za model *(eng. Mesh)* glavnog neprijatelja uzimamo model „SK_CharM_Tusk“ iz „InfinityBladeWarriors“, još jedan od besplatnih modela koje „Epic Games Inc“ nudi na svojem Unreal Engine 4 marketplace-u. Nadalje kako bi osposobili model glavnog neprijatelja s utjecajnim animacijama kako bi dojam da je ovo glavni neprijatelj bio što veći koristimo „Bossy Enemy Animation Pack“ preuzet besplatno s Unreal Engine 4 Marketplace-a dok je skup animacija bio pod „Free For The Month“ odjeljku.



Slika 81: „Retarget to Another Skeleton“

Budući da moramo promijeniti kostur našeg modela *(eng. Mesh)* da bude kompatibilan s kosturom koji koristi animacije s „Bossy Enemy Animation Pack“ koristimo Unreal Engine 4 opciju „Retarget to Another Skeleton“ te mijenjamo kostur našeg modela na kostur modela koji se koristi u „Bossy Enemy Animation Pack“ animacijama.

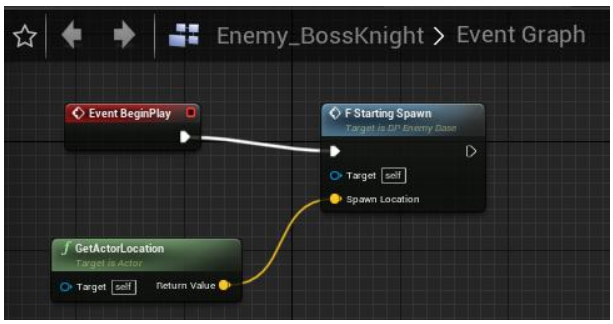


Slika 82: „Retarget to Another Skeleton“

Isto kao sa svim ostalim modelima *(eng. Mesh)*, kao što su Greystone i Troll, prije nego što počnemo razvijati logiku za sve mehanike glavnog neprijatelja potrebno je pripremiti sve animacije zajedno s njihovim „AnimNotifyState“ pozivima u korektnim vremenskim intervalima.

3.3.7.1. Mehanike glavnog neprijatelja

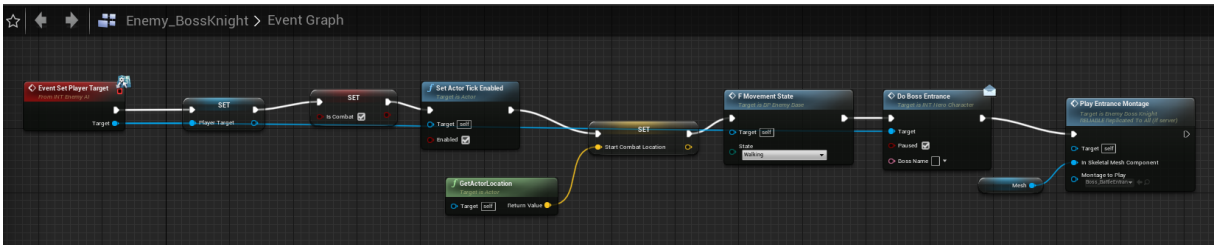
Glavni neprijatelj i dalje koristi nacrt *(eng. Blueprint)* „BP_EnemyBase“, ali će imati puno više novih funkcionalnosti i izmjena u svojem „Enemy_BossKnight“ nacrtu, nego što je imao Troll neprijatelj u „Enemy_Troll“ nacrtu zbog jedinstvenosti glavnog neprijatelja.



Slika 83: „Event BeginPlay“ događaj

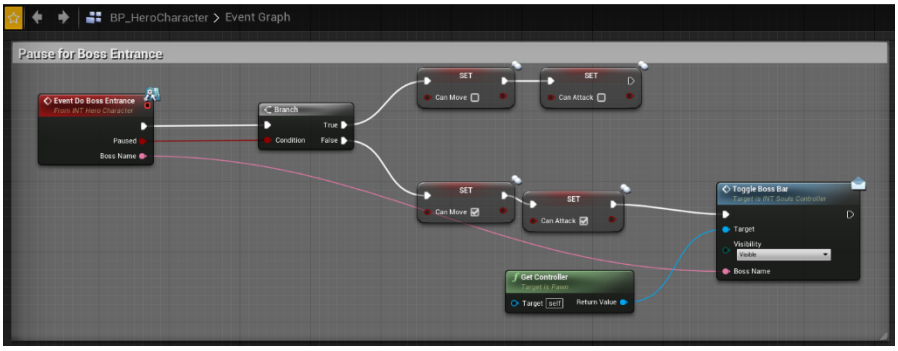
Pri samom početku, odnosno pozivu događaja *(eng. Event)* „Event BeginPlay“ prvo zapisujemo lokaciju na kojoj se stvara glavni neprijatelj preko vektor varijable „Spawn Location“.

Nakon toga glavni neprijatelj ostaje stacionaran na mjestu kojem ne stvoren i čeka da osjeti igrača preko „Pawn Sensing“ objašnjene na slici 61.



Slika 84: „Event Set Player Target“ događaj

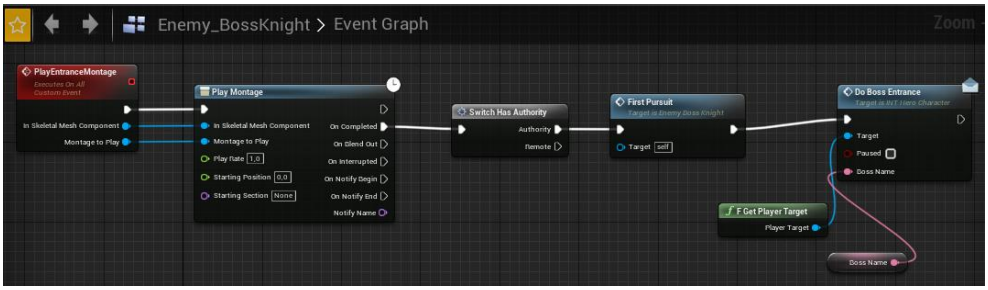
Nakon što glavni neprijatelj osjeti igrača pokreće se „Event Set Player Target“ isti kao u „BP_EnemyBase“ sa slike 63, ali sa pojedinim izmjenama i dodacima. Pokreće se „Do Boss Entrance“ događaj koji se nastavlja u nacrtu „BP_HeroCharacter“,



Slika 85: „Event Do Boss Entrance“ događaj

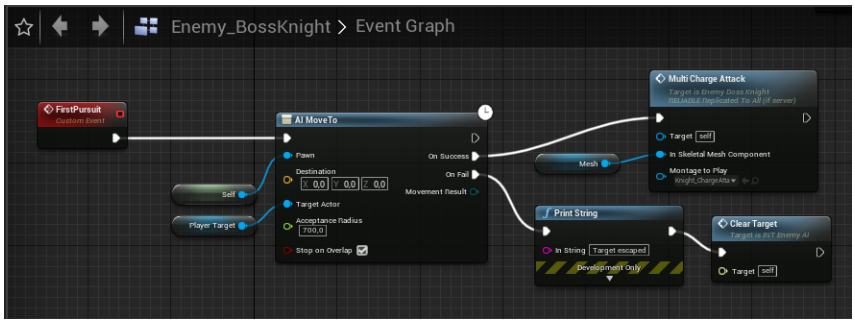
Događaj onemogućuje kretanje i napadanje igrača te kada se ponovo pozove „Do Boss Entrance“ s vrijednosti bool varijable „Paused“ postavljene kao neistina *(eng. False)* pokreće se „Toggle Boss Bar“ funkcija koja postavlja vidljivu traku za životne bodove glavnog neprijatelja (slika 29).

Kao nastavak „Event Set Player Target“ događaja sa slike 84 pokreće se „Play Entrance Montage“ događaj.



Slika 86: „PlayEntranceMontage“ događaj

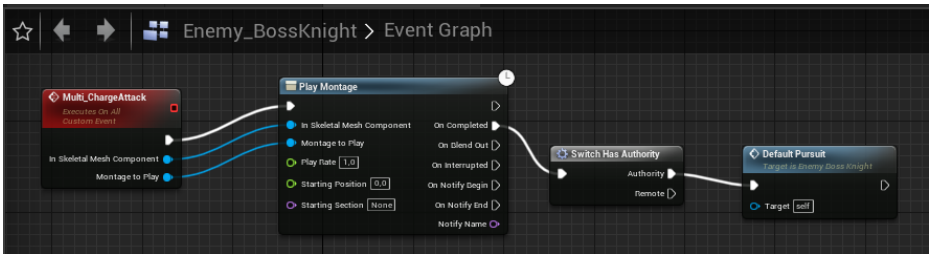
Događaj „Play Entrance Montage“ nakon što završi animaciju pokreće „First Pursuit“ događaj te postavlja „Do Boss Entrance“ bool varijablu „Paused“ na neistinu *(eng. False)*.



Slika 87: „FirstPursuit“ događaj

Preko događaja „FirstPursuit“ pomičemo glavnog neprijatelja u smjeru našeg igrača te po uspješnom izvođenju započinje „Multi Charge Attack“ događaj.

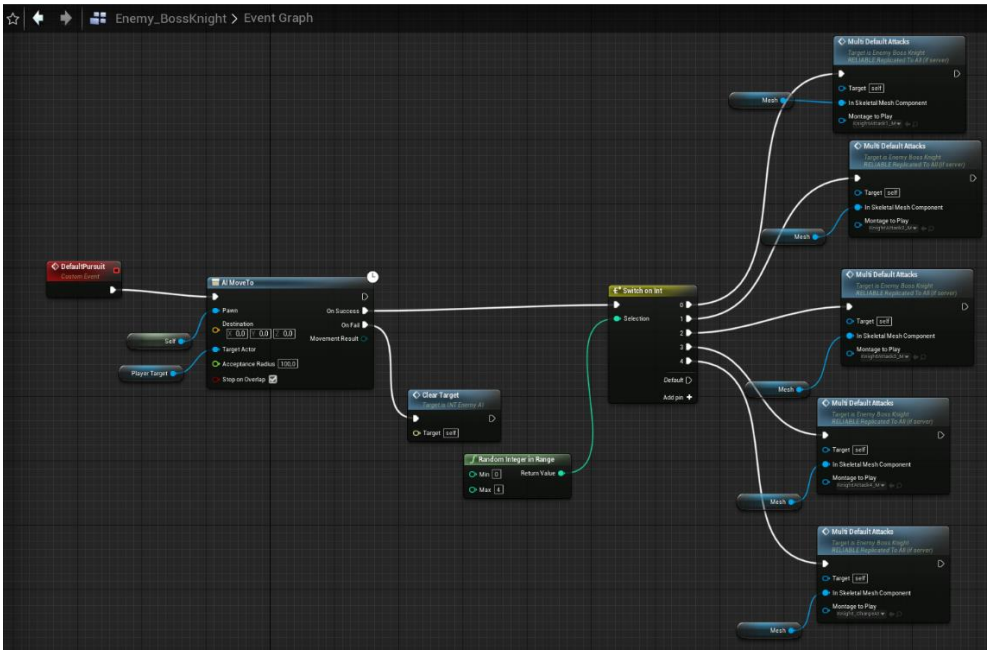
3.3.7.2. Napadi glavnog neprijatelja



Slika 88: „Multi_ChargeAttack“ događaj

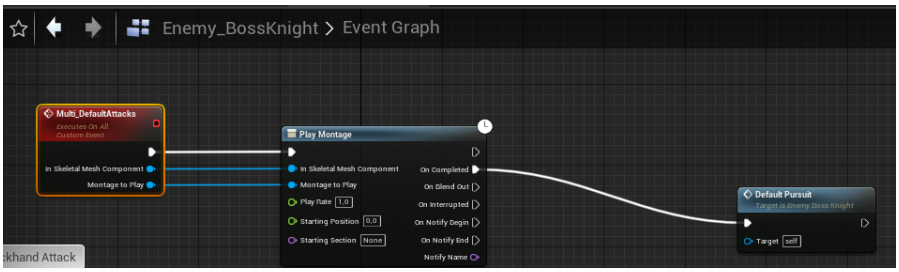
„Multi_ChargeAttack“ događaj nam služi za pokretanje početnog napada s kojim će glavni neprijatelj uvijek prvo započinjati prije nego što nastavi s „Default Pursuit“ događajem.

Slika 89: „Knight_ChargeAttack“ napad



Slika 90: „DefaultPursuit“ događaj

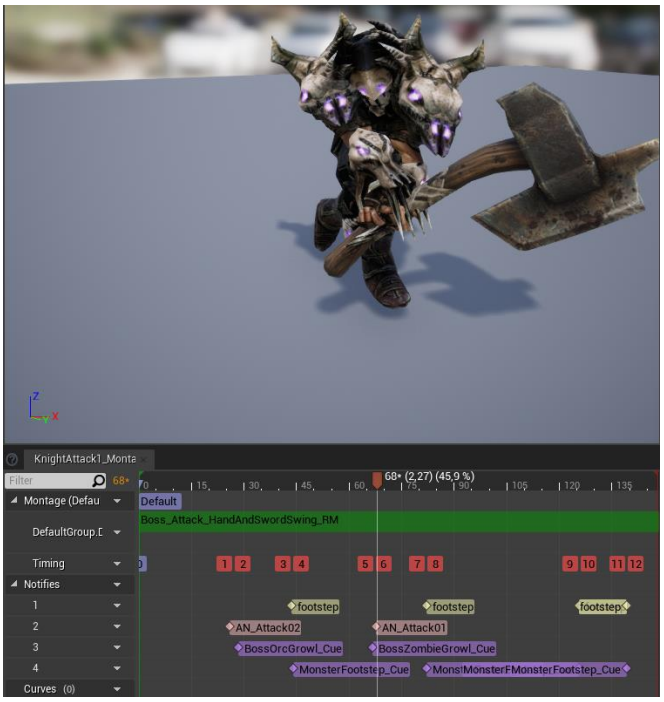
„DefaultPursuit“ događaj slično kao „FirstPursuit“ događaj pokreće „AI Move To“ funkciju no po uspjehu pokreće „Switch on Int“ funkciju koja poprima vrijednost jednog od nasumičnih brojeva između 0 i 4 preko „Random Integer in Range“ funkcije te na osnovu dobivenog broja pokreće jedan od 5 „Multi Default Attacks“ događaja koji pokreću njima zadanu animaciju.



Slika 91: „Multi_DefaultAttacks“ događaj

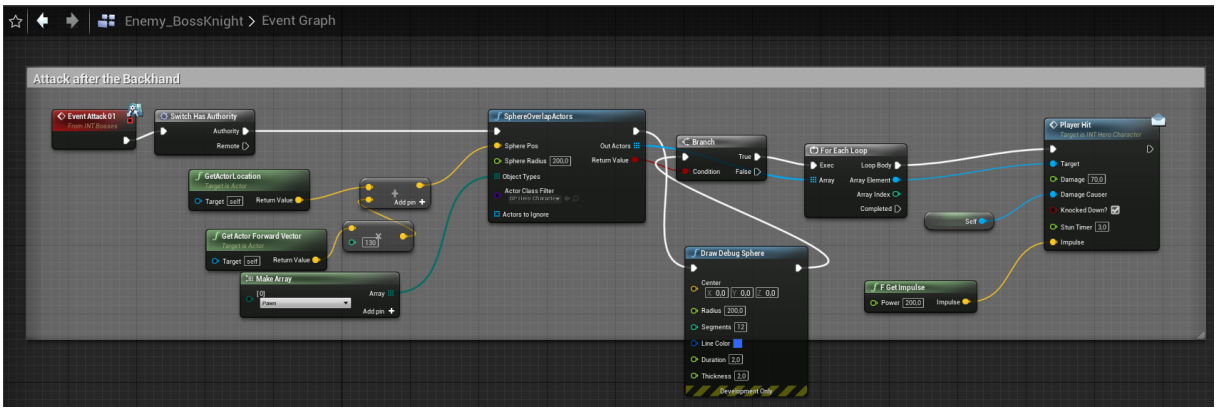
„Multi Default Attacks“ događaj nakon izvršavanja funkcije „Play Montage“ pokreće ponovo „Default Pursuit“ događaj.

Ovisno koju animaciju pokrene „Multi_DefaultAttacks“ pokreće se određeni „AnimNotifyState“ te s njime logika vezana za napade. Važna napomena je da jedna animacija nije vezana za jedan „AnimNotifyState“ već uslijed animacije možemo definirati u kojem trenutku se pokreće koji „AnimNotifyState“.



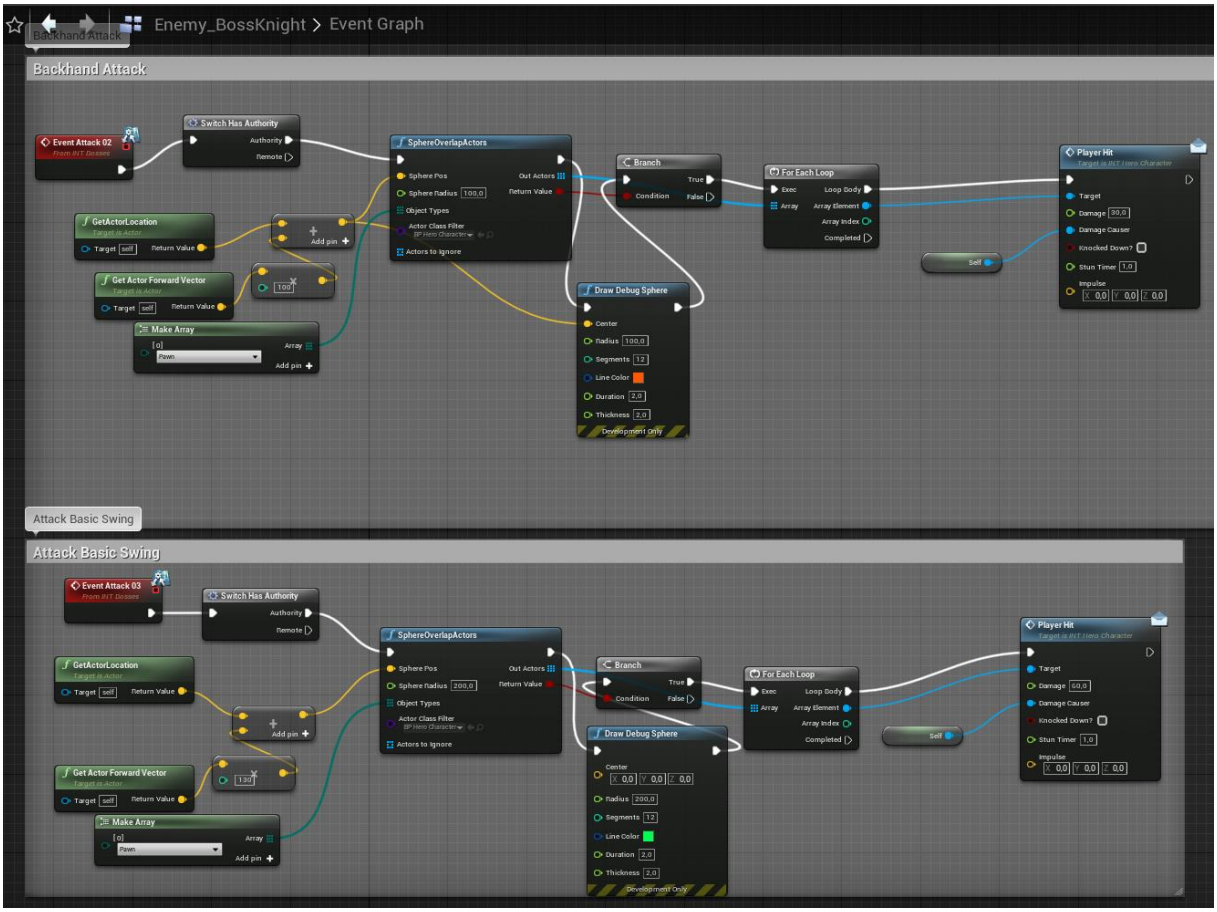
Slika 92: „KnightAttack1_Montage“ montaža animacije

Kao što vidimo prvo se pokreće „AN_Attack02“ „AnimNotifyState“ te nakon njega u 2,27 sekundi „AN_Attack01“ „AnimNotifyState“.

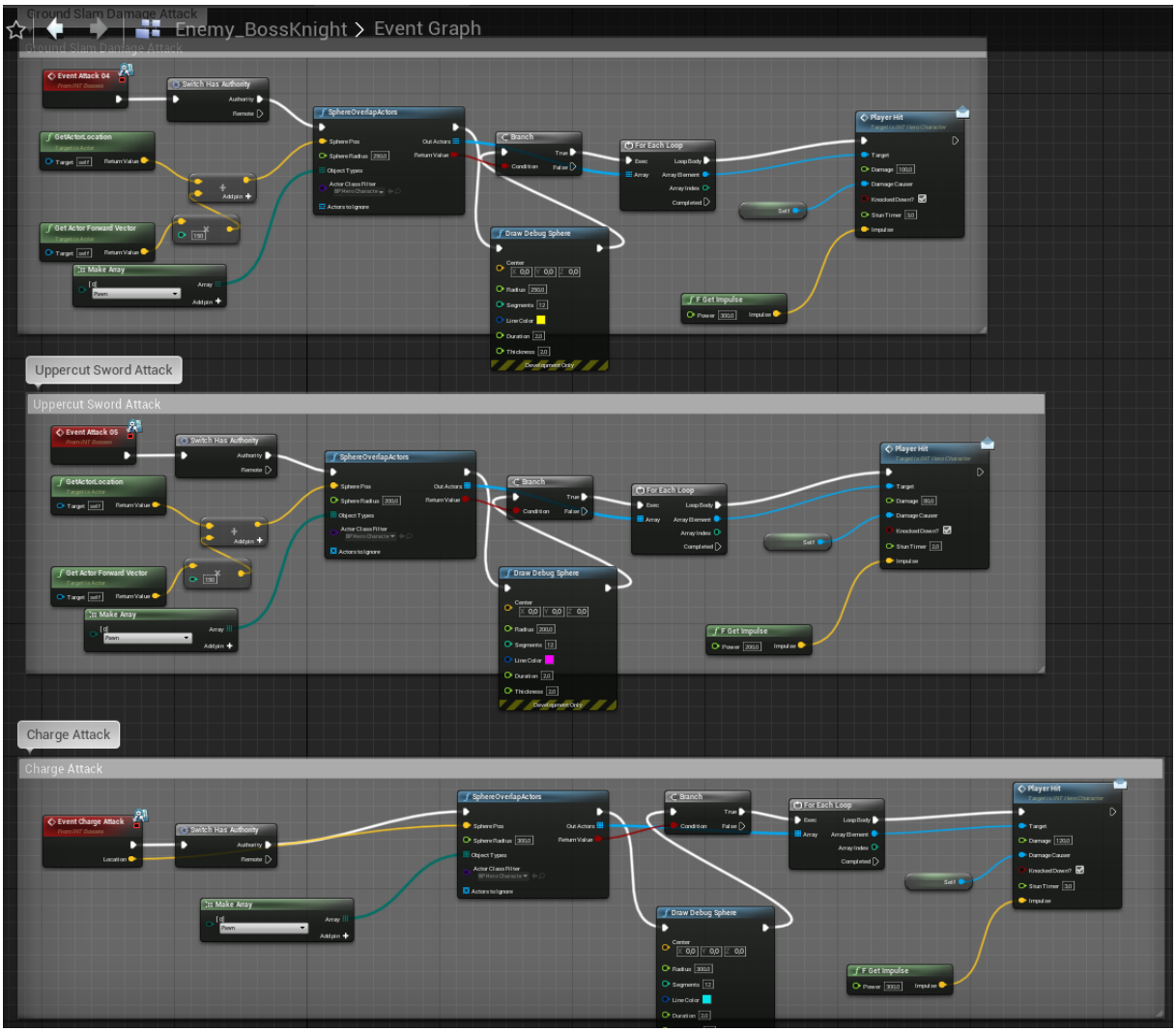


Slika 93: „Event Attack 01“ događaj

Prvi napad s obzirom na pozvani „AnimNotifyState“ pokreće standardan napad u kojem stvaramo „SphereOverlapActors“ funkciju s proslijeđenim parametrima o lokaciji našeg glavnog neprijatelja te ukoliko sfera ima koliziju sa sferom igrača odnosno „BP_HeroCharacter“ nacrta *(eng. Blueprint)* pokreće „Player Hit“ događaj sa slike 45.



Slika 94: „Event Attack 02“ i „Event Attack 03“ događaji



Slika 95: „Event Attack 04“, „Event Attack 05“ i „Event Charge Attack“ događaji

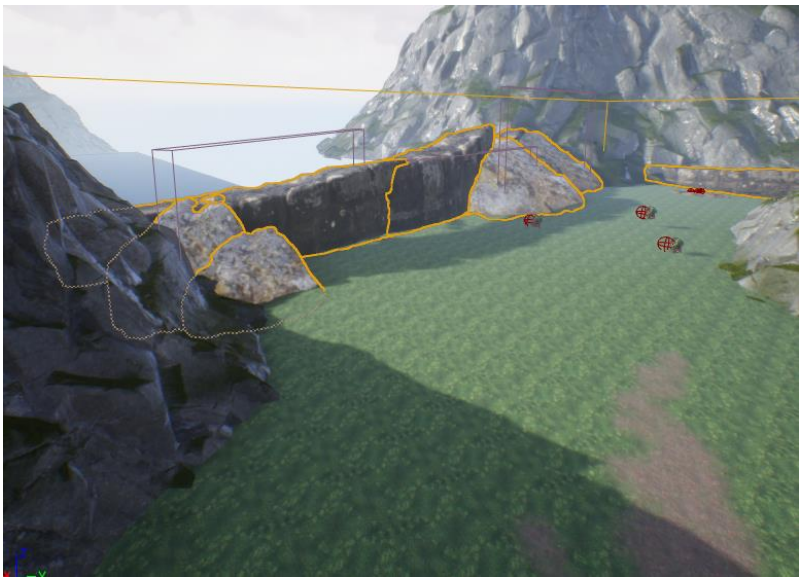
3.3.8.Kreiranje vanjskog svijeta za videoigru

Iako je primarna ideja bila napraviti ručno svijet za kretanje u videoigri preko „Heightmap“ je rasterska slika koja se uglavnom koristi kao diskretna globalna mreža u sekundarnom modeliranju nadmorske visine te se može uključiti u Unreal Engine 4 kao automatski generiran svijet s obzirom na uključeni „Heightmap“.

Konačno je prevagnula ideja korištenja jednog vrlo detaljnog i lijepo izrađenog svijeta savršen za videoigru ovog završnog rada, a to je „The Valley“ [14].

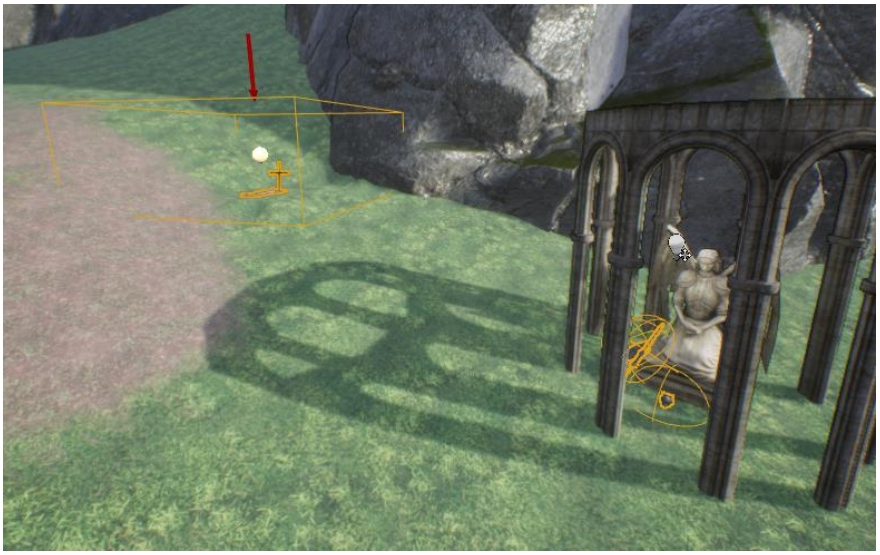
Slika 96: The Valley

„The Valley“ iako vrlo detaljan i kvalitetno izrađen zahtijevao je kojekakve dorade kako bi se mogla koristiti za videoigru kao što su umetanje modela, predmeta, terena te micanja biljaka/trave/stabala *(eng. Foliage)* kako bi videoigra imala bolje performanse.



Slika 97: Dorade na „The Valley“

Ubačeni su modeli kamenja [10] kako igrač ne bi mogao otići izvan „The Valley“ svijeta te postavljeni posvuda Troll neprijatelji (deset Troll neprijatelja).

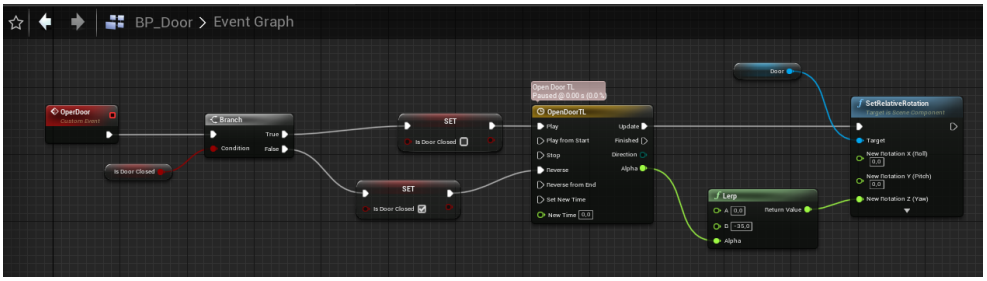


Slika 98: Postavljanje kontrolnih točaka i predmeta

Postavljanje 6 oružja, 6 štitova i 3 šljema kako bi ih igrač mogao pokupiti te 3 kontrolne točke *(eng. Checkpoint)*.

Slika 99: Postavljanje vrata

Postavljamo vrata koja onemogućuju igraču kretanje do glavnog neprijatelja sve dok ne zadovolji uvjete da ih otvori.



Slika 100: „BP_Door“ nacrt

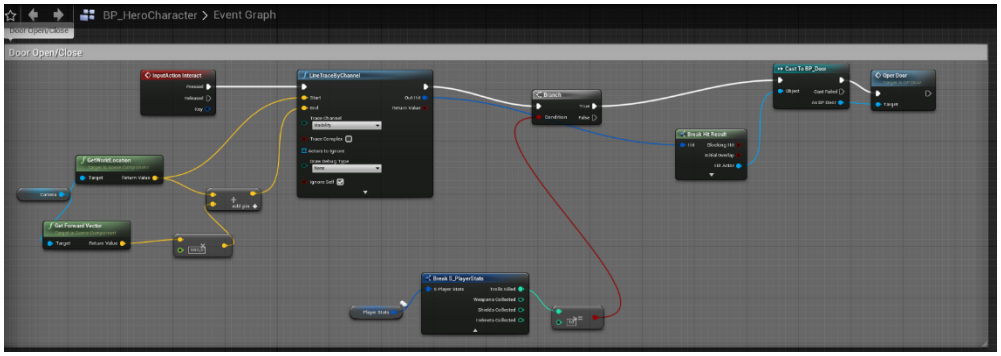
Kako bi igrač mogao otvarati vrata potrebno je napraviti zaseban nacrt *(eng. Blueprint)* za vrata „BP_Door“ u kojem definiramo događaj „OpenDoor“.

„OpenDoor“ prvo provjerava jesu li zatvorena vrata te ako su zatvorena postavlja bool vrijednost „Is Door Closed“ na neistinu *(eng. False)* te pokreće „OpenDoorTL“ funkciju koja rotira model vrata kroz animaciju te im postavlja novu poziciju preko „SetRelativeRotation“ funkcije.

Slika 101: „OpenDoorTL“

Ukoliko vrata nisu zatvorena, odnosno ako su već otvorena onda se pokreće „OpenDoorTL“ funkcija, ali u obrnutom smjeru.

Kako bi igrač mogao komunicirati s vratima potrebno je unutar „BP_HeroCharacter“ nacrta *(eng. Blueprint)* napraviti događaj „InputAction Interact“.



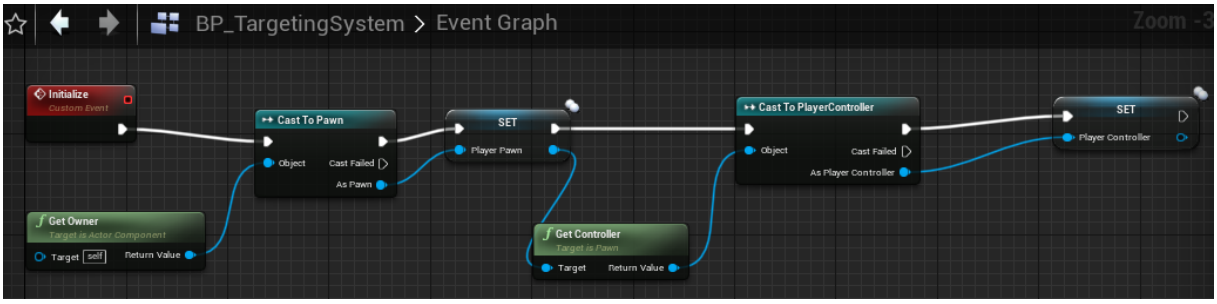
Slika 102: „InputAction Interact“

Unutar „InputAction Interact“ događaja koristimo „LineTraceByChannel“ funkciju koja s obzirom na poziciju kamere i vektora unaprijed stvara ravnu liniju te ako dotakne model vrata koji smo postavili u „The Valley“ svijet pokreće „Open Door“ događaj, ali prije toga igrač mora zadovoljiti uvjet tako da je ubio svih deset Troll neprijatelja.

3.3.9.„Targeting system“

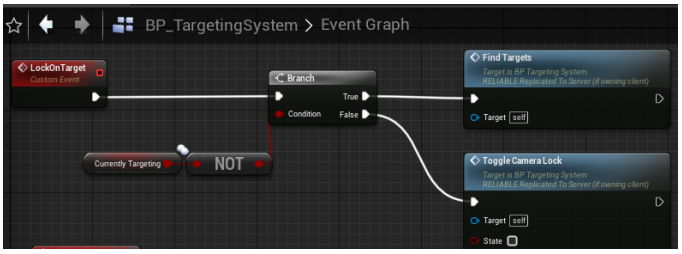
„Targeting system“ jedna je od glavnih mehanika „Souls-like“ videoigara koja omogućava lakše kretanje oko neprijatelja s kojim se igrač bori te da je jednostavnije pogoditi neprijatelja svojim napadima, ali i obraniti se od nanesene štete s kotrljanjem ili blokiranjem.

3.3.9.1. „BP_TargetingSystem“ „Targeting System“ nacrt



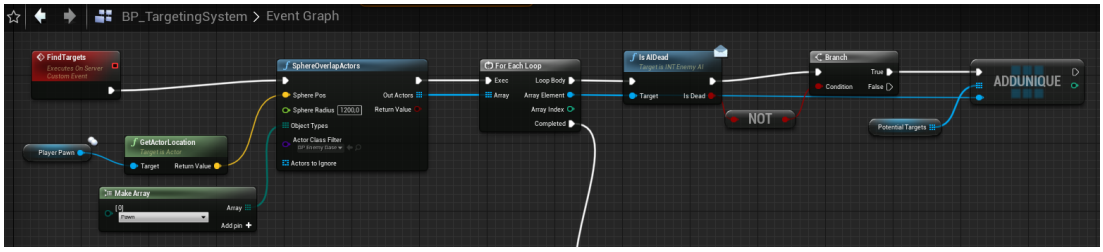
Slika 103: „BP_TargetingSystem“

Prvo postavljamo 2 varijable, a to su „Player Pawn“ i „Player Controller“. Nadalje, glavni događaj biti će „LockOnTarget“.



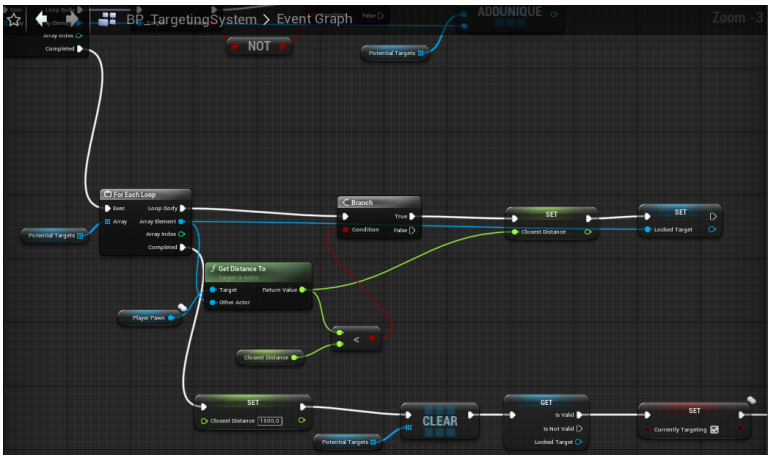
Slika 104: „LockOnTarget“ događaj

Pozivom ovog događaja provjeravamo je li igrač već zaključan na nekog neprijatelja te ako nije pokrećemo „Find Targets“ događaj *(eng. Event)*.



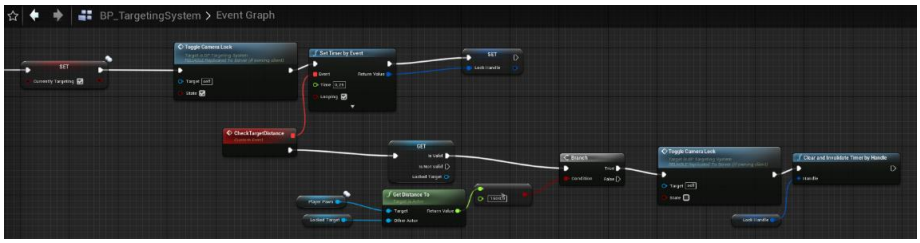
Slika 105: „Find Targets“ događaj #1

„Find Targets“ događaj prvo stvara veliku sferu oko našeg igrača s radijusom od 1200 te sfera detektira sve „Pawn“ objekte i za svaki „Pawn“ koji detektira provjerava je li taj „Pawn“ mrtav preko „Is AIDead“ funkcije. Ukoliko „Pawn“, odnosno neprijatelj/glavni neprijatelj, nije mrtav dodaje se u niz potencijalnih meta „Potential Targets“ preko „ADDUNIQUE“ funkcije.



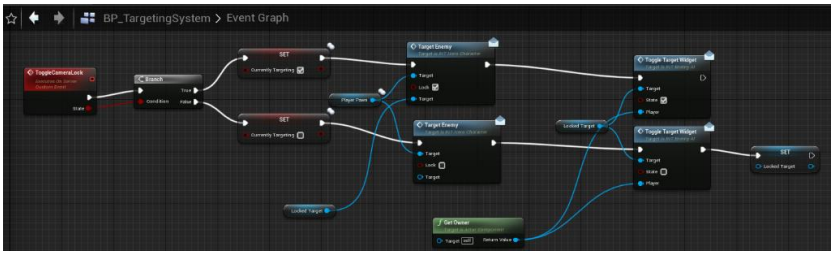
Slika 106: „Find Targets“ događaj #2

Nakon što se petlja završi za svakog detektiranog neprijatelja pokreće novu petlju koja prima vrijednosti niza „Potential Targets“ te računa udaljenost igrača do tih potencijalnih meta, s obzirom na najmanju vrijednost „Closest Distance“ jedna potencijalna meta iz niza „Potential Targets“ postaje odabrana meta „Locked Target“. „Closest Distance“ varijabla postavlja se na 1500 te se niz „Potential Targets“ čisti svih unosa, a stanje igrača bool „Currently Targeting“ postavlja se na istinu *(eng. True)*.



Slika 107: „Find Targets“ događaj #3

Pokreće se događaj „Toggle Camera Lock“ te „Set Timer by Event“ funkcija koja pokreće događaj „CheckTargetDistance“ svakih 0,25 sekundi. Ujedno postavljena je „Timer Handle Structure“ pod nazivom „Lock Handle“ koja određuje koliko će se dugo ponavljati događaj. „CheckTargetDistance“ provjerava udaljenost igrača od odabranog neprijatelja te ako vrijednost njihove udaljenosti pređe 1500 ponovo se pokreće „Toggle Camera Lock“ funkcija i „Lock Handle“ se miče preko „Clear and Invalidate Timer by Handle“ funkcijom.

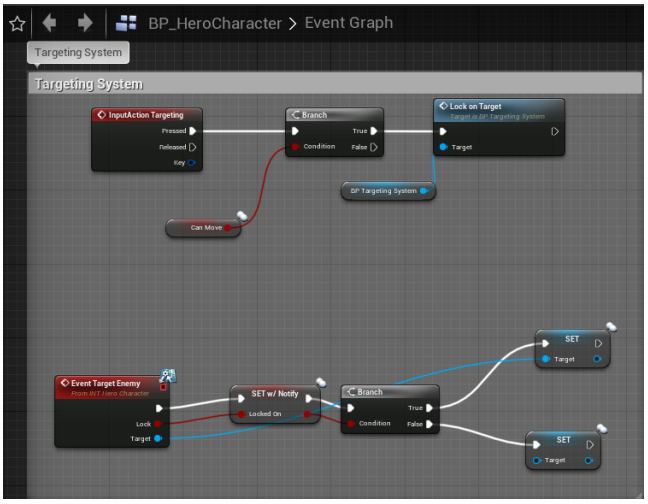


Slika 108: „ToggleCameraLock“ događaj

S obzirom je li događaj „ToggleCameraLock“ pozvan da aktivira ili deaktivira stanje zaključane kamere ovisi stanje koje joj je proslijeđeno. Ako je postavljeno da se želi zaključan kamera na neprijatelja pokreće se istinita *(eng. True)* grana te se postavlja bool vrijednost „Currently Targeting“ na istinu *(eng. True)* i pokreće se događaj „Target Enemy“ s proslijeđenim vrijednostima igrača te odabrane mete te nakon njega događaj „Toggle Target Widget“ koji ovisno o smjeru grananja briše „Locked Target“ vrijednost ukoliko događaj „ToggleCameraLock“ je korišten u stanju neistine *(eng. False)*.

3.3.9.2. „BP_HeroCharacter“ „Targeting System“ nacrt

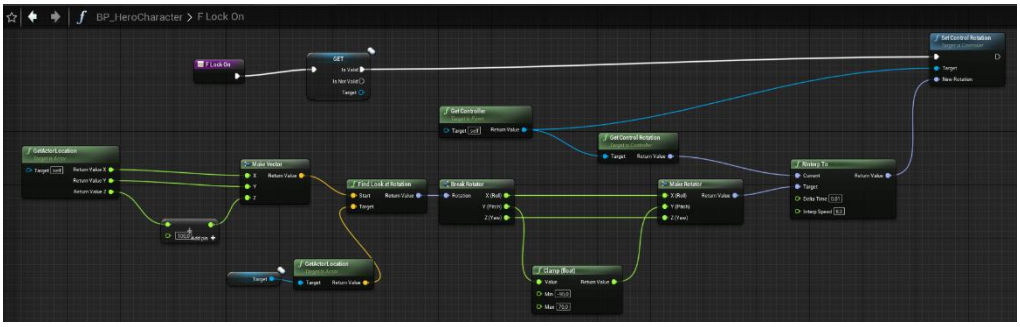
Naime kako bi igrač pozvao sve navedene funkcionalnosti oko „Targeting Systema“ potrebno je pozvati te događaje preko klika tipke na tipkovnici „Tab“.



Slika 109: „InputAction Targeting“ događaj

Pritiskom tipke „Tab“ pokreće se događaj „InputAction Targeting“ te se provjerava može li se igrač kretati te ako može započinjemo događaj „Lock On Target“ objašnjen u slici 103.

Događaj „Event Target Enemy“ poziva se iz nacrta „BP_TargetingSystem“ uslijed „ToggleCameraLock“ događaja sa slike 107 te postavlja igrača u stanje „Locked On“ preko bool varijable „Locked On“ te ako je „Locked On“ postavljen na istinu *(eng. True)* postavlja se meta „Target“, a ako je neistina *(eng. False)* miče se meta „Target“.

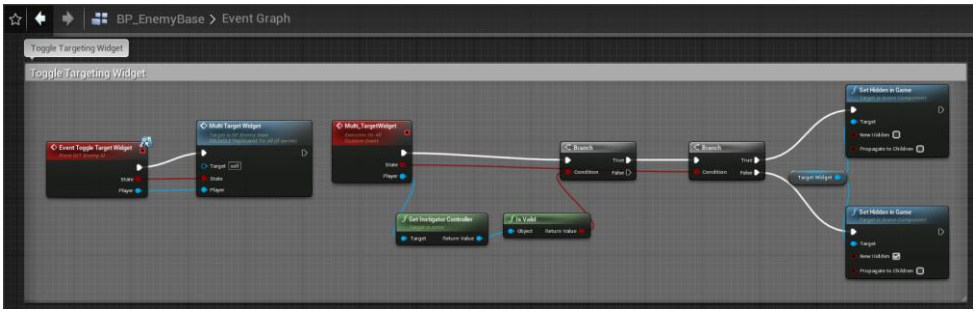


Slika 110: „F Lock On“ funkcija

Sad kad nam je postavljena meta, odnosno „Target“ možemo pozvati funkciju „F Lock On“ sa slike 8 te se u toj funkciji pomoću matematičkih izračuna postavlja rotacija kamere igrača oko neprijatelja na kojeg je zaključana i rotira oko neprijatelja preko „Set Control Rotation“ funkcije.

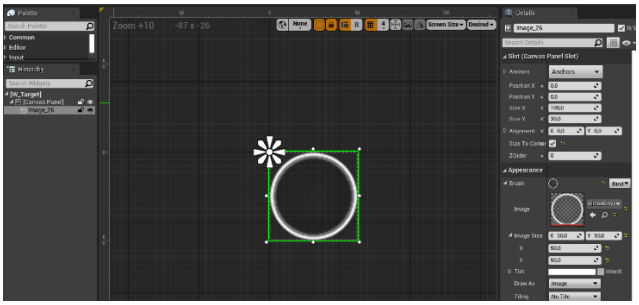
3.3.9.3. „BP_EnemyBase“ „Targeting System“ nacrt

Dio koji mora biti obavljen sa strane neprijatelja je da se widget nacrta na modelu *(eng. Mesh)* neprijatelja te ćemo te događaje pozivati u nacrtu „BP_EnemyBase“.



Slika 111: „Event Toggle Target Widget“ i „Multi_TargetWidget“ događaji

Oba događaja služe za pozivanje i stavljanje našeg widgeta s kojim označujemo na kojeg neprijatelja je kamera zaključana.



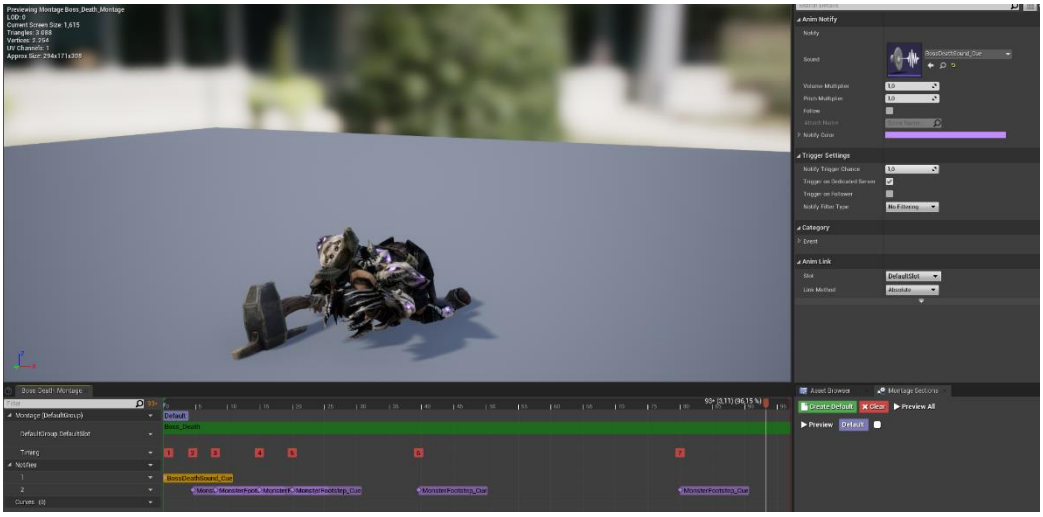
Slika 112: „W_Target“

„W_Target“ jednostavan je widget sa slikom koju koristimo da prati neprijatelja na kojem je kamera igrača zaključana.

Slika 113: Prikaz „W_Target“ widgeta u videoigri

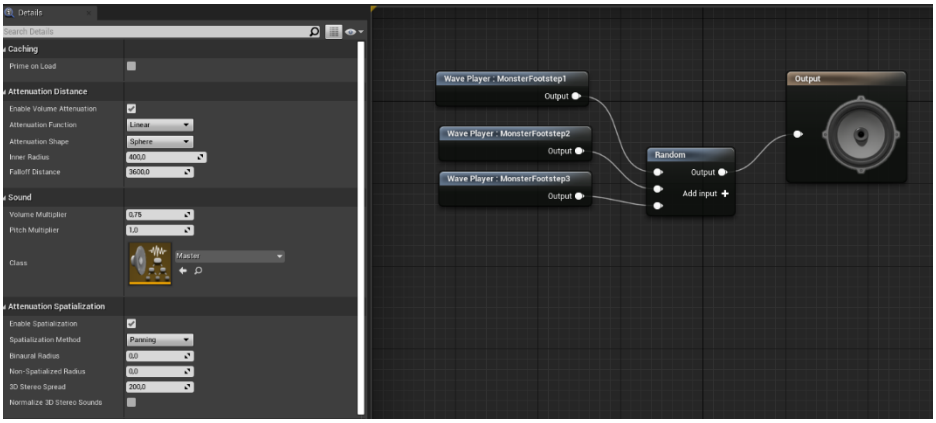
3.3.10. Interakcija igrača i neprijatelja sa svijetom

Osim animacija, zvuk je jedan od glavnih aspekta u bilo kojoj igri te je potrebno da i ova videoigra ima osnovne zvukove koji se pozivaju preko „AnimNotifyState“.



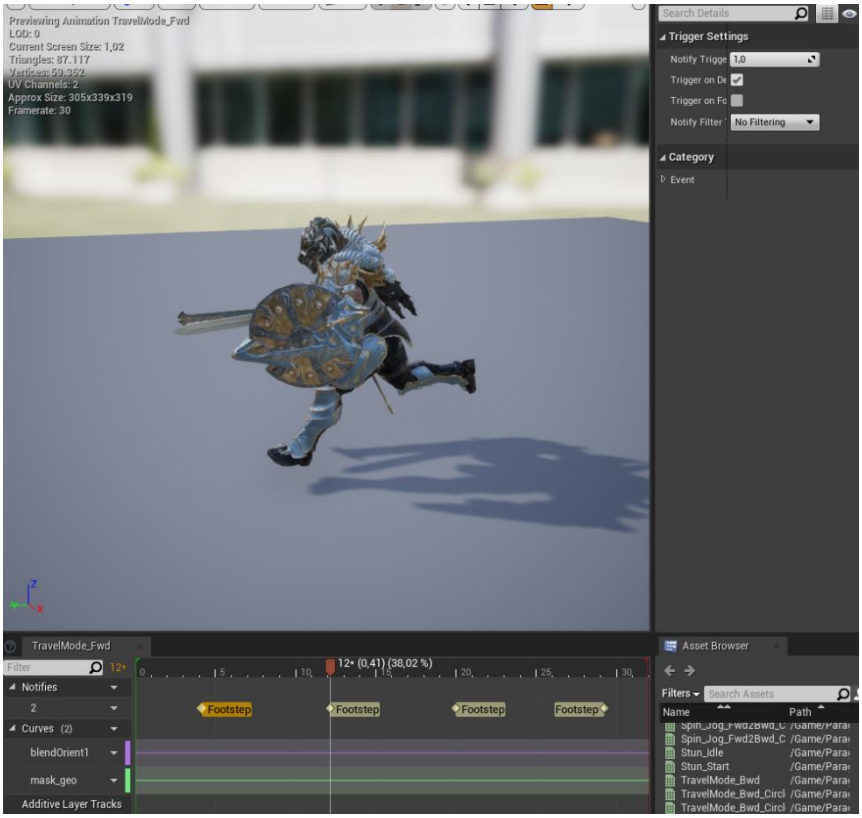
Slika 114: „AnimNotifyState“ zvukovi za „Boss_Death_Montage“ animaciju

Sa slike 114 vidimo da u trenutku dok započne animacija za „Boss_Death_Montage“ pokreće se zvuk „BossDeathSound_Cue“ zajedno s puno „MonsterFootstep_Cue“ koji su postavljeni svaki put kada glavni neprijatelj napravi bilo koji korak u svojim animacijama.



Slika 115: „MonsterFootstep_Cue“

Za razliku od neprijatelja, model igrača koristi drugačiju metodu za zvukove vezane za korake te na način na koji se oni pojavljuju.



Slika 116: „Footstep“ okidači

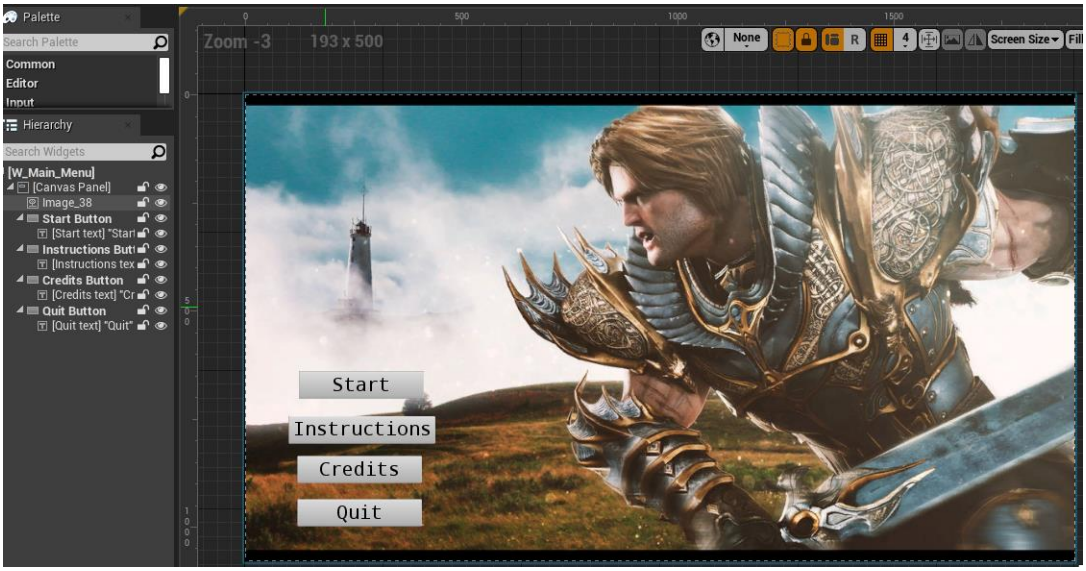
Dok neprijatelji za svoje korake koriste „MonsterFootstep_Cue“ igračev model Greystone-a koristi okidače *(eng. Trigger)* koji puštaju zvukove ovisno o površini, odnosno materijalu *(eng. Material)*, na kojoj igrač hoda (trava, zemlja, kamen).

Slika 117: Materijal površine po kojoj igrač hoda

Slika 118: Zvukovi hodanja po 3 vrste površine

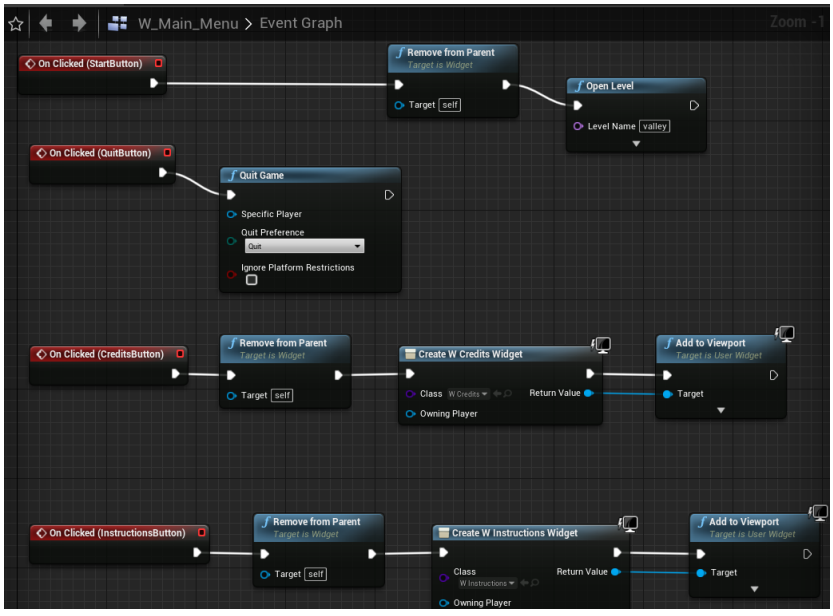
3.3.11. Glavni izbornik

Kako bi igrač bio upućen u kontrole videoigre te koji je cilj videoigre najbolje ne napraviti jednostavan glavni izbornik *(eng. Main menu)* prije ulaska u samu videoigru.



Slika 119: „W_Main_Menu“

U glavnom izborniku imamo opciju započeti igru preko gumba „Start“, pročitati upute preko gumba „Instructions“, pročitati tko je napravio videoigru preko gumba „Credits“ te izaći iz videoigre preko gumba „Quit“.



Slika 120: „W_Main_Menu“ nacrt

Događaji koji se pozivaju klikom na određene gumbove su:

- „On Clicked (StartButton)“ kojim se widget zatvara te pokreće nivo „valley“
- „On Clicked (QuitButton)“ kojim se izlazi iz videoigre.
- „On Clicked (QuitButton)“ kojim se miče „W_Main_Menu“ widget i poziva „W_Credits“ widget

Slika 121: „W_Credits“

- „On Clicked (InstructionsButton)“ kojim se miče „W_Main_Menu“ widget i poziva „W_Instructions“ widget

Slika 122: „W_Instructions“

4. Zaključak

Na kraju možemo zaključiti da izrada videoigre nije nimalo lagan posao, pogotovo ne za jednu osobu. Izuzetno puno unaprijed razmišljanja je potrebno pri izradi videoigre te detaljno razrađen plan djelovanja kako bi se spriječilo nazadovanje usred programiranje te korigiranje prethodno programiranih funkcionalnosti.

Osim kompleksnosti koje dolaze sa izradom videoigre vrijeme igra jedan jako veliki faktor, da bi se neka naizgled jednostavna funkcionalnost napravila potrebno je dosta vremena, a još više ako se nalete na neke probleme usred programiranja ili neočekivane bug-ove koji zahtijevaju revidiranje logike iza koda kojim smo osposobljavali funkcionalnosti.

Unatoč svim preprekama izrada videoigre jedan je od najzanimljivijih iskustava koje programiranje može pružiti iz razloga što god možemo zamisliti to u našoj videoigri možemo i realizirati te unatoč jednostavnosti ove videoigre pokriveno je izuzetno puno funkcionalnosti što koriste gotovo sve igre i iskustvo stečeno izradom ove videoigre nikad neće biti izgubljeno, a vrijeme potrošeno nikad bolje utrošeno.

Unreal Engine 4 izuzetno je kvalitetan pogon za igre *(eng. Game engine)* te korištenje nacrta *(eng. Blueprint)* je stvarno revolucionaran korak u smjeru programiranja zbog svoje vizualne estetičnosti i lakog snalaženja te svoje praktičnosti u usporedbi sa standardnim načinom programiranja kao što je u slučaju C++ programskog jezika kojeg Unreal Engine 4 podržava. Za početnik, ali i naprednije programere u programiranju videoigara nacrti *(eng. Blueprints)* su stvarno idealan pristup za lako snalaženje u kodu i jednostavnije shvaćanje tijeka programskog koda.

5. Popis literature

[1] Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic, Wallace Witkowski. Available: https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990. [Pokušaj pristupa 25. svibanj 2021.]

[2] What is Unreal Engine?, Thomas Denham. Available: https://conceptartempire.com/what-is-unreal-engine/. [Pokušaj pristupa 25. svibanj 2021.]

[3] Unreal Engine documentation. Available: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/index.html. [Pokušaj pristupa 25. svibanj 2021.]

[4] Souls-like action rpg game with multiplayer. Available: https://devaddict.teachable.com/courses/souls-like-action-rpg-game-with-multiplayer/lectures/24976710. [Pokušaj pristupa 30. ožujak 2021.]

[5] $17,000,000 of Paragon content for free. Available: https://www.unrealengine.com/en-US/paragon. [Pokušaj pristupa 28. ožujak 2021.]

[6] Blender. Available: https://www.blender.org/. [Pokušaj pristupa 28. ožujak 2021.]

[7] Infinity Blade: Adversaries. Available: https://www.unrealengine.com/marketplace/en-US/product/infinity-blade-enemies. [Pokušaj pristupa 28. ožujak 2021.]

[8] PlayerHit_Cue. Available: https://freesound.org/. [Pokušaj pristupa 16. travanj 2021.]

[9] GameOverSound_Cue. Available: https://freesound.org/. [Pokušaj pristupa 16. travanj 2021.]

[10] Model groba, kamenja, vrata. Available: https://quixel.com/bridge. [Pokušaj pristupa 20. travanj 2021.]

[11] FXVarietyPack. Available: https://www.unrealengine.com/marketplace/en-US/product/a36bac8b05004e999dd4b1d332501f49. [Pokušaj pristupa 28. ožujak 2021.]

[12] Infinity Blade: Warriors. Available: https://www.unrealengine.com/marketplace/en-US/product/infinity-blade-characters. [Pokušaj pristupa 28. ožujak 2021.]

[13] Bossy Enemy Animation Pack. Available: https://www.unrealengine.com/marketplace/en-US/product/bossy-enemy-animation-pack. [Pokušaj pristupa 28. ožujak 2021.]

[14] The Valey. Available: https://www.reddit.com/r/unrealengine/comments/9hrksk/free_download_the_valley_speed_level_design/. [Pokušaj pristupa 01. travanj 2021.]

[15] BossDeathSound_Cue. Available: https://freesound.org/. [Pokušaj pristupa 16. travanj 2021.]

6. Popis slika