RAZVOJ RAČUNALNE IGRE KROZ UNREAL ENGINE PLATFORMU

Butigan, Matej

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

Matej Butigan

Z A V R Š N I R A D

Razvoj računalne igre kroz Unreal Engine platformu

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

Predmet: Uvod u programiranje

Z A V R Š N I R A D

Kandidat: Matej Butigan

Naslov rada: Razvoj računalne igre kroz Unreal Engine platformu

Mentor: Teo Žuljević, viši predavač

Split, rujan 2024.

Sadržaj

Sažetak

Cilj ovog završnog rada bio je napraviti modernu igru koristeći Unreal Engine, jedan od naprednijih alata za razvoj videoigara. Kroz proces razvoja, istraženi su ključni elementi ovog žanra, poput preživljavanja u neprijateljskom okruženju, borbe protiv neprijateljskih jedinica vođenih umjetnom inteligencijom te upravljanja resursima koji su ključni za opstanak igrača.

Inspiracija za ovu igru dolazi iz popularnih naslova u žanru preživljavanja, no željelo se postići originalno iskustvo koje stavlja fokus na dinamično generiranje neprijatelja, atmosfersko okruženje te izazovnu mehaniku preživljavanja. Korištenjem Unreal Enginea, demonstrirane su mogućnosti realistične fizike te implementacija složene umjetne inteligencije koja kontrolira ponašanje zombija.

Koncept igre je jednostavan, potrebno je preživjeti napade zombija što je duže moguće. Svaka nova razina znači ujedno i povećavanje težine, pružajući igraču sve veći izazov kako igra napreduje. Rad također prikazuje razvojne procese, izazove i rješenja implementirana tijekom razvoja igre, kao i mogućnosti za daljnja poboljšanja.

Ključne riječi: igra preživljavanja, umjetna inteligencija, Unreal Engine, videoigra

Summary

Computer game development through the Unreal Engine platform

The goal of this final paper was to create a modern video game using Unreal Engine, one of the more advanced tools for video game development. Throughout the development process, key elements of this genre were explored, such as surviving in a hostile environment, combating enemy units driven by artificial intelligence, and managing resources crucial for the player's survival.

The inspiration for this game comes from popular titles in the survival genre, but the aim was to create an original experience that focuses on dynamic enemy generation, atmospheric environments, and challenging survival mechanics. By using Unreal Engine, the project demonstrated the capabilities of realistic physics and the implementation of complex artificial intelligence that controls zombie behavior.

The concept of the game is simple: survive zombie attacks for as long as possible. Each new level also increases the difficulty, providing the player with a greater challenge as the game progresses. The project also showcases the development processes, challenges, and solutions implemented during the game's development, as well as opportunities for further improvements.

Keywords: artificial intelligence, survival game, Unreal Engine, video game

1. Uvod

Videoigre su postale neizostavni dio moderne kulture, a njihov utjecaj proteže se daleko izvan same zabave. Od samih početaka, videoigre su redefinirale način na koji komuniciramo s tehnologijom, pružajući igračima interaktivna iskustva koja kombiniraju vizualnu umjetnost, zvuk, priču i mehaniku igranja. FPS (engl. *First-Person Shooter*) žanr, populariziran s klasičnim igrama poput Counter-Strikea, ostavio je dubok trag na gaming industriji, oblikujući buduće generacije igara.

Inspiracija za ovaj projekt dolazi iz ljubavi prema igrama, posebno onima iz FPS žanra, te želje za stvaranjem vlastitog virtualnog svijeta. Cilj ovog rada je detaljno prikazati proces izrade FPS igre preživljavanja koristeći Unreal Engine, jedan od vodećih alata za razvoj videoigara. Unreal Engine omogućuje stvaranje kompleksnih 3D okruženja, napredne fizike, te složenih sustava umjetne inteligencije, što ga čini savršenim izborom za ovakav projekt.

Kroz rad se prolaze sve faze razvoja igre, od početne ideje i planiranja, preko dizajna razina i likova, pa sve do programiranja i implementacije *gameplaya.* Fokus je stavljen na stvaranje atmosferskog iskustva preživljavanja u kojem igrač mora balansirati resurse, preživjeti napade zombija, te istraživati neprijateljski svijet. Posebna pažnja posvećena je sustavima borbe, upravljanja resursima i generiranja neprijatelja, koji čine temelj ove igre.

Izazovi nastali pri izradi ove igre omogućili su dublje razumijevanje procesa stvaranja videoigara i načina na koji različiti elementi igre zajedno stvaraju jedinstveno iskustvo za igrača.

Rad je podijeljen u šest poglavlja. U prvom poglavlju se kratko opisuju Unreal Engine njegove značajke i tržište. Drugo poglavlje govori općenito o pucačkim igrama. Preostala četiri poglavlja uz pomoć slika približavaju proces razvijanja igre kroz tematike postavljanja kontroliranog igrača oružja, animacija te inteligentnih protivnika.

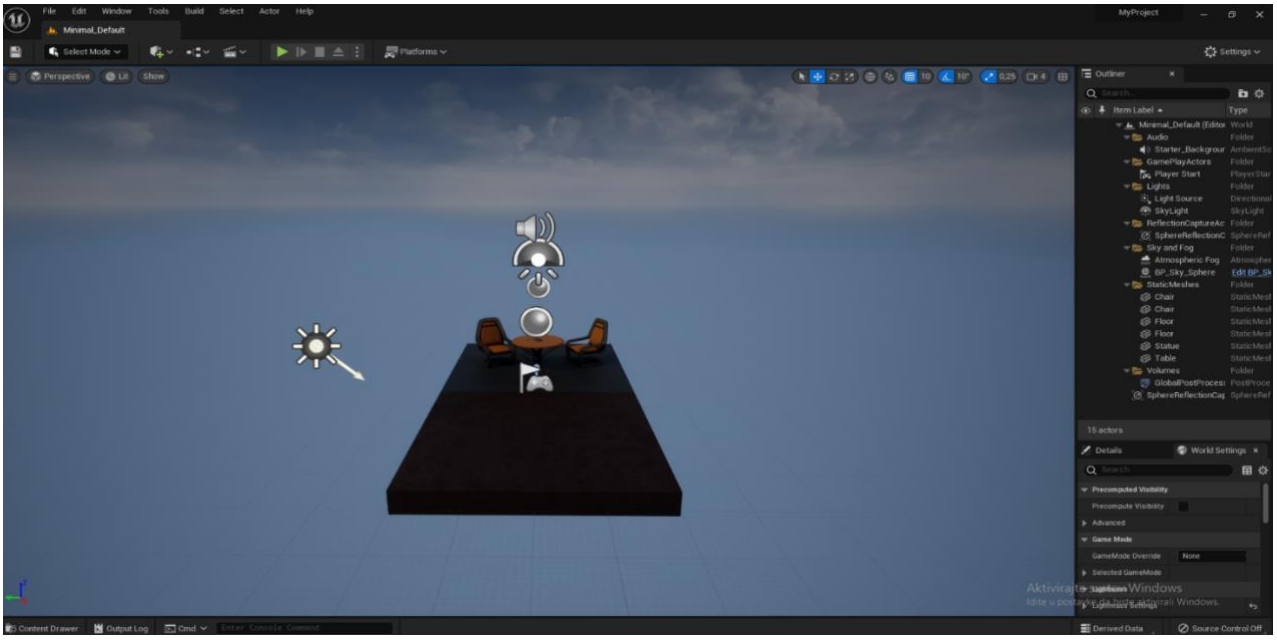
2. Pogonski alat Unreal

2.1. Osnovne značajke pogonskog okruženja

Alat korišten pri izradi ove igre je mulitplatformski alat koji je razvio Epic Games. Prvi put je predstavljen još 1998. godine kao pogonski alat iza pucačke igre iz prvog lica „Unreal“. Riječ je o Unreal Engine-u, koji je do danas ostvario značajan napredak te podržava razvoj 2D i 3D igara za razne platforme, uključujući Windows, macOS, Linux, mobilne uređaje, konzole i sustave virtualne stvarnosti[2].

Od korisnika ovog alata se očekuje da posjeduje osnovno znanje iz programskog jezika C++ ili pak korištenje vizualnog skriptiranja koje Unreal nudi putem sustava Blueprints. Po instalaciji i pokretanju Unreal Enginea, korisnik može koristiti Unreal Project Browser kojim upravlja projektima. Unreal Engine također omogućuje instalaciju dodatnih alata i paketa za razvoj specifičnih vrsta igara, poput VR podrške i alata za optimizaciju performansi.

Verzija Unreal Enginea korištena u ovom radu je 5.0.2, koja pruža napredne mogućnosti u pogledu real-time renderiranja, fizike, animacija i drugih ključnih elemenata potrebnih za izradu moderne videoigre.



Slika 1: Unreal Engine sučelje

Kao što je vidljivo na slici 1, radno sučelje Unreal Enginea podijeljeno je na nekoliko dijelova. Prvi, i vjerojatno najvažniji, dio sučelja je Viewport, gdje se odvija pregled i uređivanje scene. Ovdje je omogućeno postavljanje objekata, uređivanje materijala i testiranje interakcija unutar 3D prostora.

Drugi ključan dio sučelja je Content Browser, koji omogućuje pregled i upravljanje svim datotekama projekta, uključujući teksture, modele, animacije i skripte. Ovo je zapravo središte za organizaciju svih resursa igre.

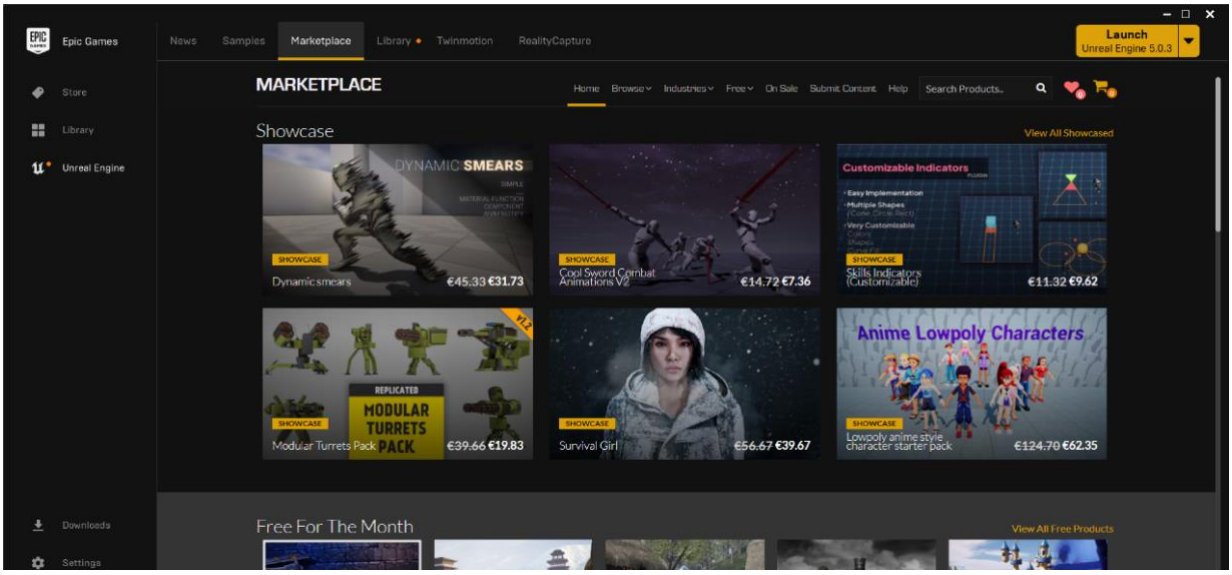
Treći dio sučelja je panel Details, koji služi za pregled i podešavanje svojstava odabranih objekata u sceni. Ovdje korisnik može detaljno konfigurirati parametre poput pozicije, rotacije i veličine objekata, kao i specifičnih postavki poput fizike i kolizije.

Navigacijska traka na vrhu sučelja omogućuje pristup alatima za kreiranje novih objekata, upravljanje postavkama projekata te izvođenje igre u Play modu, gdje se testiraju gameplay elementi unutar samog enginea.

Unreal Engine pruža snažan set alata koji omogućuje korisnicima da kreiraju visokokvalitetne videoigre s naprednom grafikom i realističnom fizikom, a sve unutar jednog integriranog razvojnog okruženja.

2.2 Unreal tržište

Čitava videoigra može se izraditi od originalnih objekata, modela, zvukova, animacija i sl., no za lakšu izradu koristi se Unreal Marketplace. Na slici 2 prikazano je kako izgleda Unreal Marketplace prilikom posjećivanja:



Slika 2: Unreal Marketplace

Prilikom korištenja Unreal Marketplacea može se uočiti kako se tu mogu pronaći razni elementi korisni za izradu videoigara – 3D modeli, materijali, zvučni efekti, animacije i skripte. Nažalost, većinu je moguće koristiti tek nakon kupovine istih, no mogu se pronaći i oni koji su besplatni, a značajni su za razvoj igara. Svi korišteni elementi u ovom radu bili su besplatni, iako neki od njih tijekom izrade projekta mogu prijeći u plaćenu verziju[1].

Jedna od prednosti Unreal Marketplacea koja se izdvaja je upravo velika zajednica korisnika i programer koji su voljni podijeliti vlastito iskustvo i resurse. To implicira olakšanu suradnju, učenje i brži napredak. Zahvaljujući tome, mnoge kompanije i samostalni programeri se odlučuju za Unreal Marketplace u svrhu bržeg razvoja vlastitih projekata te poboljšanja kvalitete istih. Time je olakšan put ka uspješnom plasiranju visokokvalitetnih proizvoda na tržište.

Unreal Marketplace čini iznimno korisno mjesto za razmjenu znanja i resursa unutar Unreal zajednice s ulogom u širenja popularnosti ovog razvojnog okvira i poticanja kreativnosti razvojnih timova diljem svijeta.

3. Klasična videoigra i njene značajke

3.1 Pucačka videoigra

Ovakve videoigre testiraju prostornu svijest, reflekse te brzinu igrača, kako u izoliranim jednoigračkim (engl. *singleplayer* – igre koje nemaju mrežnu povezanost te je igrač sam u okruženju) okruženjima, tako i u mrežnom višeigračkom načinu (engl. *multiplayer* – mrežno povezani igrači na serverima dijele okruženja) [5]. Ovim podžanrom obuhvaćene su mnoge igre koje imaju zajedničku crtu usmjerenosti na radnje avatara koji se oružjem bori protiv protivničkih NPC likova (engl. NPC – *non-playable character*; lik kojim igrač ne može upravljati) ili drugih avatara koje kontroliraju ostali igrači. Ova vrsta akcijskih videoigara pažnju gotovo potpuno usmjerava na poraz protivnika uz pomoć oružja koje je igraču dodijeljeno. Najčešće su to vatrena oružja ili neko drugo oružje dužeg dometa, koje se može koristiti u kombinaciji s drugim alatima (poput granata) za indirektni napad, dodatnu obranu (oklop) ili modificiranje ponašanja oružja (teleskopski nišan). Dosta pucačkih igara kao resurse obično koriste streljivo, oklop ili „nadogradnje“ (engl. *upgrades*). koje poboljšavaju oružje lika igrača.

3.2. Igra iz prvog lica

Kao i kod većine pucačkih videoigara, sadržaj igre ovakvog tipa uključuje avatar, jedno ili više oružja s dugim ili kraćim dometom i različitim brojem neprijatelja. Kako se radnja ovih igara odvija u 3D okruženju, one su realističnije od igara 2D formata te uz to imaju preciznije prikaze gravitacije, osvjetljenja, zvuka i sudara (engl. *collision* – kolizija). Pucačke igre iz prvog lica mogu se smatrati zasebnim žanrom ili vrstom pucačkih igara, koja je pak podžanr šireg žanra akcijskih igara. Nakon što je izašla igra Doom (1993.), igre ovog stila su često nazivane klonovima Dooma (engl. *Doom clones*). Tijekom narednih godina dolazi do zamjene ovog izraza s onim „pucačina iz prvog lica“. Uvoditeljem ovog žanra često se smatra Wolfenstein 3D (1992.). Kritičari pak prepoznaju slične, iako manje napredne igre, razvijane unazad do 1973. godine. Povremeno dolazi do neslaganja oko specifičnih dizajnerskih elemenata koji čine pucačinu iz prvog lica. Primjerice, naslovi poput Deus Ex ili BioShock mogu se smatrati pucačima iz prvog lica, no mogu se uklopiti i u žanr „uzimanja uloga“ (engl. *role playing game*) jer se znatno oslanjaju na taj žanr [8].

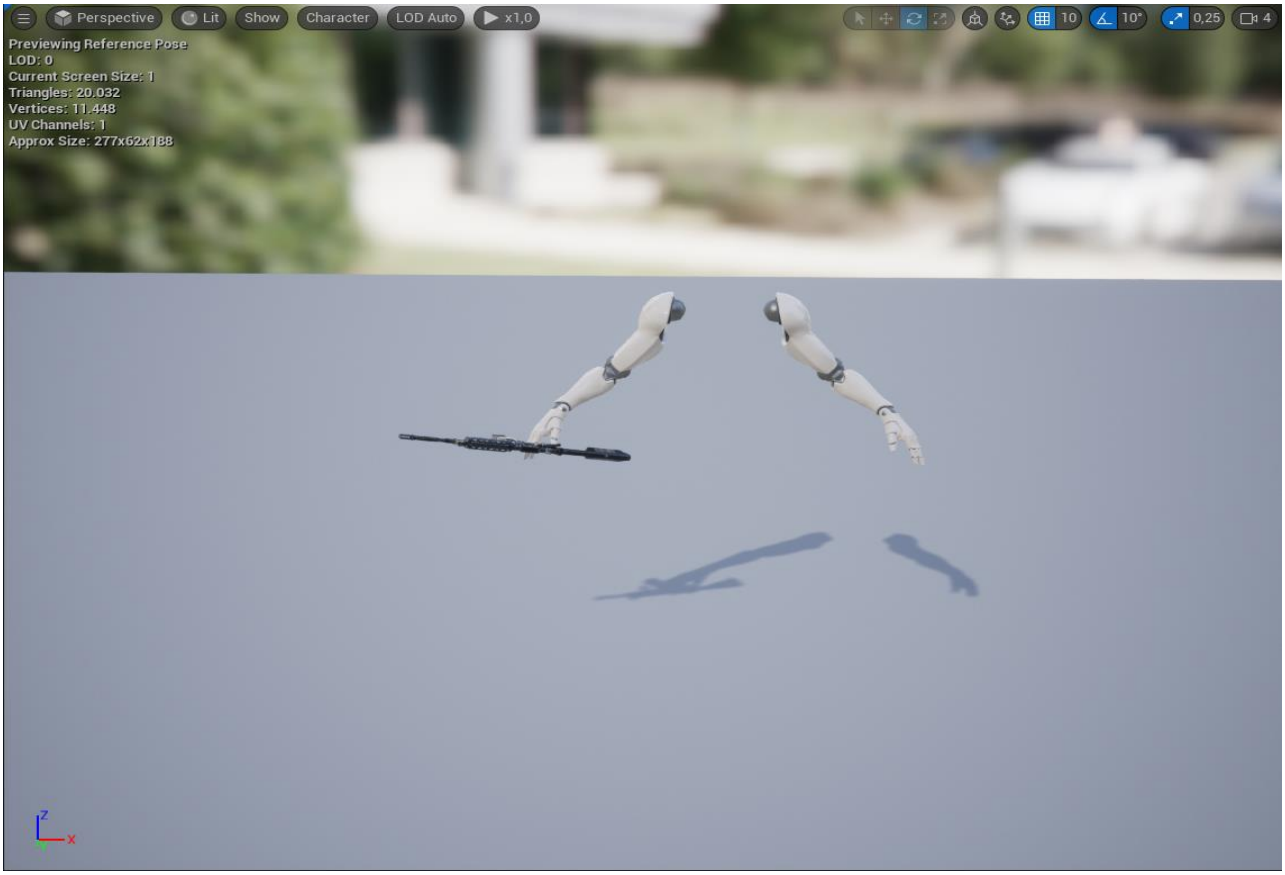
4. Prvi koraci u Unreal Engineu

4.1. Uvod

Igrač iz prve perspektive (engl. *First Person Character*) predstavlja igrača kojim se upravlja. Kako bi igrač bio pokretan te sposoban obavljati određene operacije, one se prvo moraju definirati u njegovom kontroleru. Unreal Engine pruža solidnu osnovu za jednostavna kretanja, upravljanje kamerom kao i interakciju sa okolinom[3].

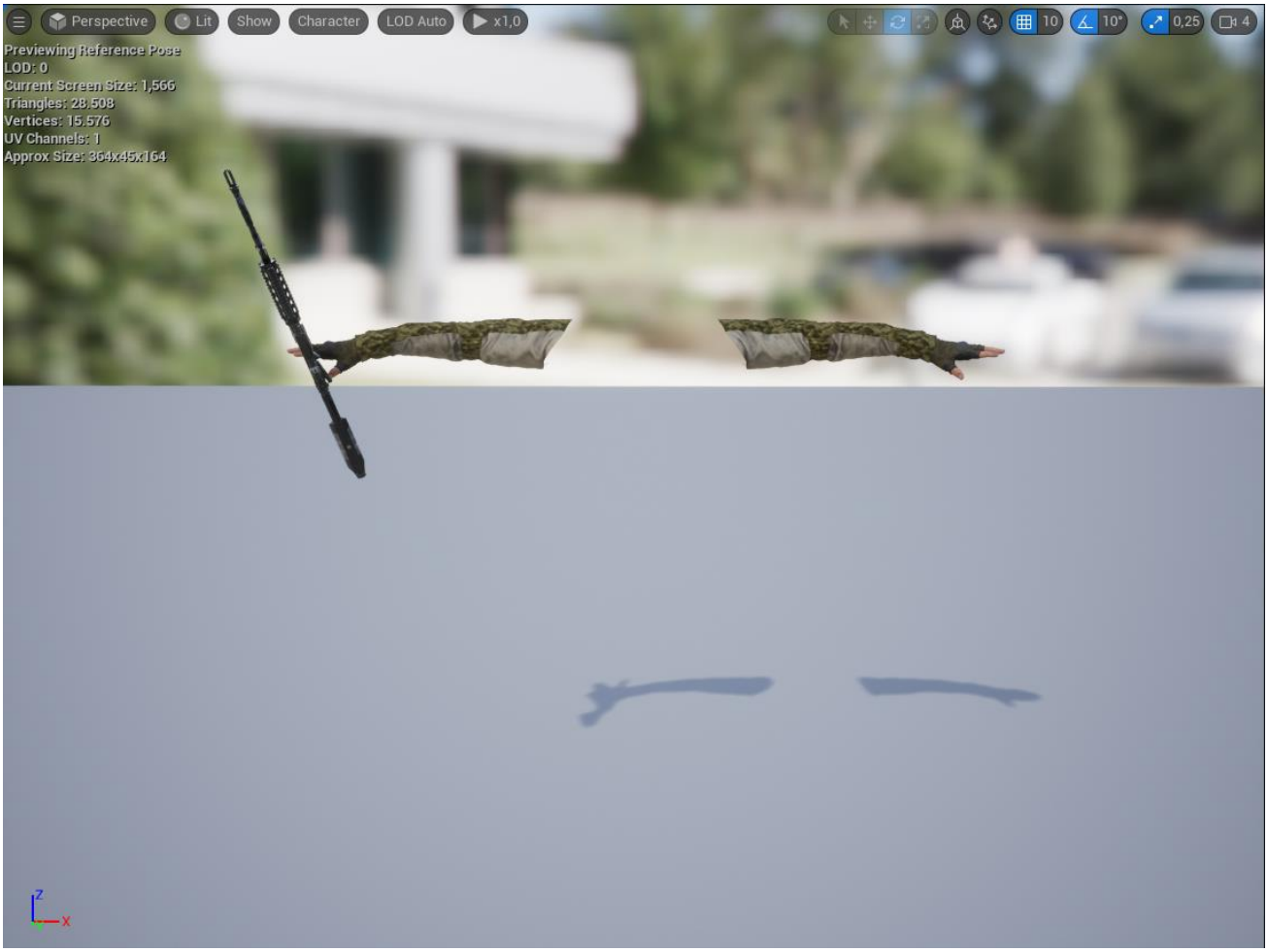
4.2. Postavljanje i implementacija

Implementacija je započeta postavljanjem igrača iz prve perspektive, kreiranjem osnovnog *First Person Charactera* (FPC) unutar Unreal Enginea koristeći *First Person Template*. Ovaj predložak uključuje osnovnu postavku kamere, kretanja i osnovnih funkcionalnosti.



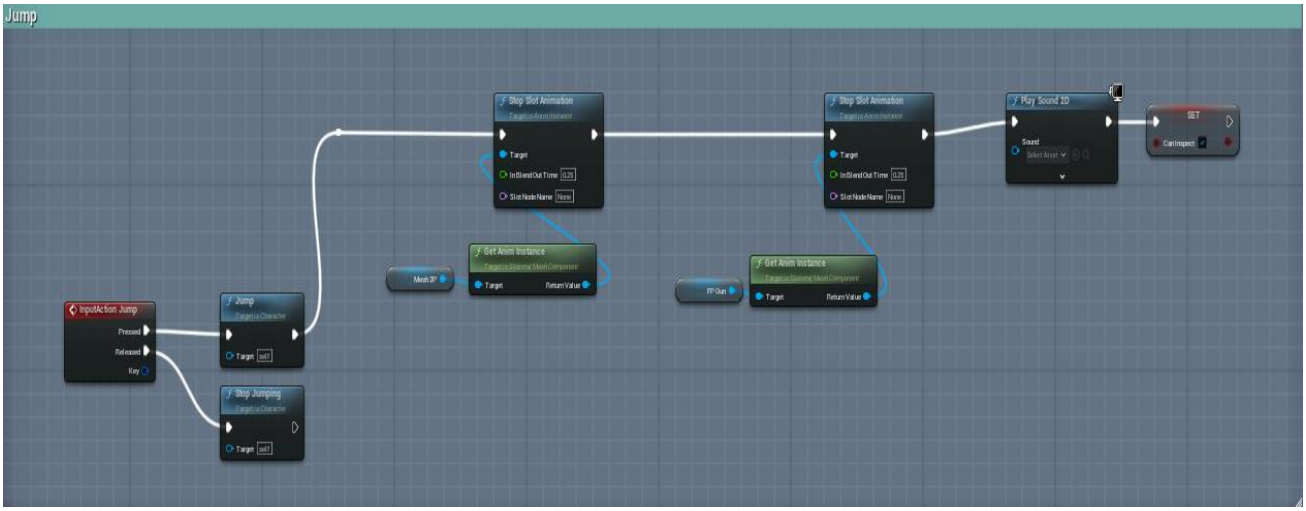
Slika 3: Osnovni model ruke bez dodanog mesha

Slika 3 prikazuje osnovni model ruku unutar Unreal Enginea prije dodavanja detaljnog mesha. Kao što se vidi, u ovoj fazi, ruke su jednostavne i služe kao *placeholder* za daljnji razvoj. U sljedećem koraku sa Unreal Marketplacea je preuzet mesh za ruke te implementiran na postojeći model ruku.



Slika 4: Model ruku nakon dodavanja mesha

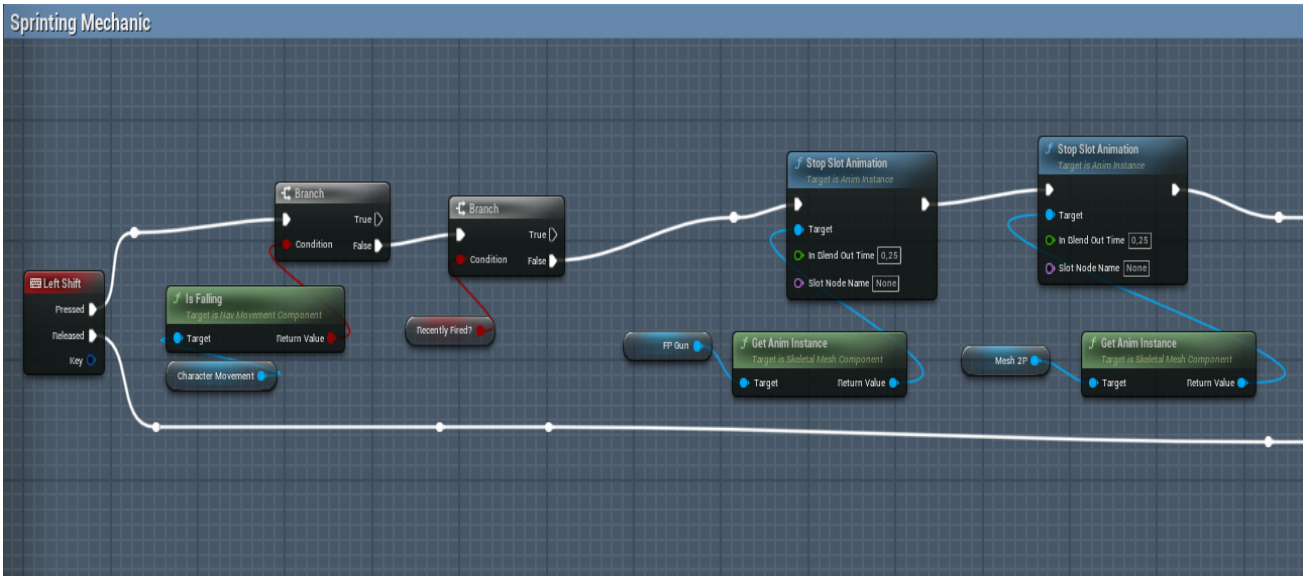
Nakon što je uređen lik kao na slici 4 može se započeti s implementacijom pripadajućeg kontrolera a prvi korak je postavljanje *Blueprinta* za lika. Unutar *Blueprinta* obuhvaćene su sve funkcionalnosti koje igrač može izvoditi.



Slika 5: Skok Blueprint

Na slici 5 je prikazan jednostavan *Blueprint* koji implementira funkcionalnost skakanja. Ovaj *Blueprint* sastoji se od nekoliko ključnih elemenata:

- **ImpulseAction Jump**: Ovaj čvor je početak *Blueprinta* i pokreće funkciju skakanja kada igrač pritisne definiranu tipku (*Space*). Funkcija je zadužena za dodavanje vertikalne sile na igrača kako bi se postigao skok.
- **Jump i Stop Jumping**: Čvorovi "*Jump*" i "*Stop Jumping*" povezuju akciju skakanja s animacijom skoka. Kada igrač pritisne *Space*, aktivira se funkcija skakanja, a kada otpusti tipku, skakanje prestaje.
- **Animacije i zvukovi**: *Blueprint* također uključuje animaciju i zvučni efekt koji se pokreću prilikom skakanja. Na slici se može vidjeti da se koriste čvorovi kao što su *Stop Slot Animation* i *Play Sound* za upravljanje animacijom i zvukom skoka.
- **Korištenje Mesh i FP Gun instance**: Ovi čvorovi se koriste za pristup animacijskom sustavu igrača kako bi se osiguralo da se ispravne animacije i zvukovi pokrenu tijekom skakanja.

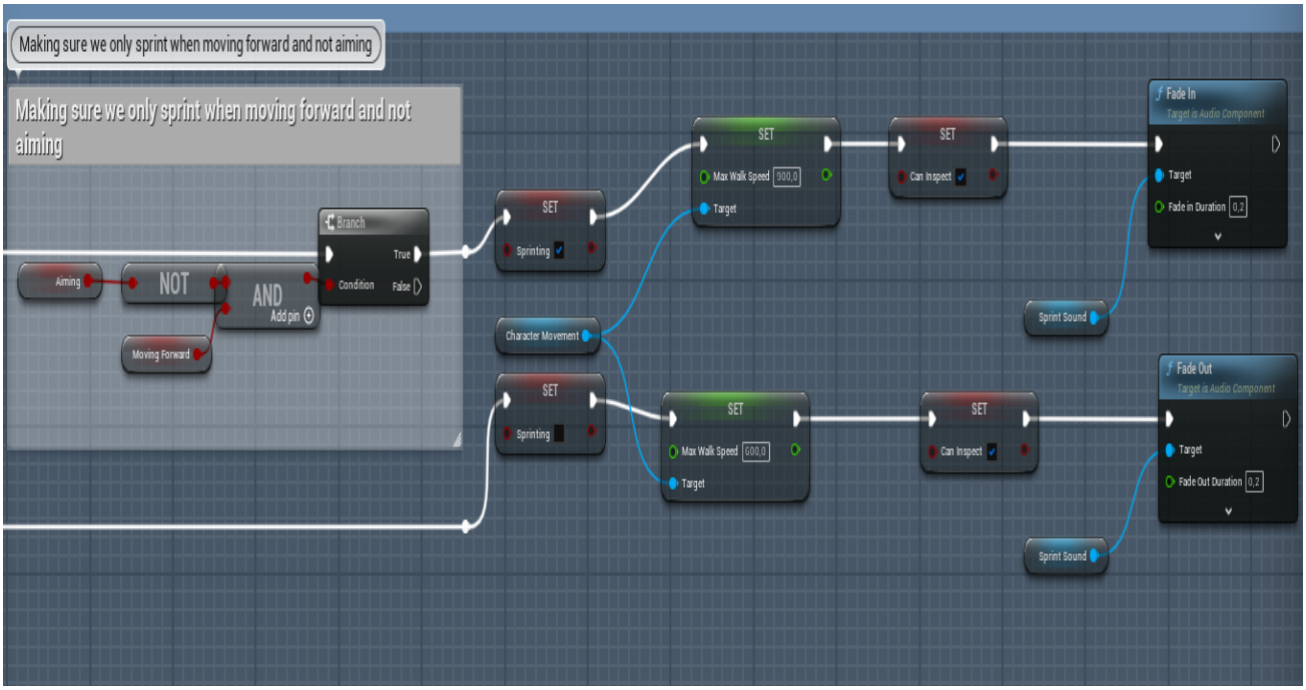


Slika 6: Sprint Blueprint

Kao što prikazuje slika 6, *Blueprint* za sprintanje sastoji se od nekoliko ključnih elemenata:

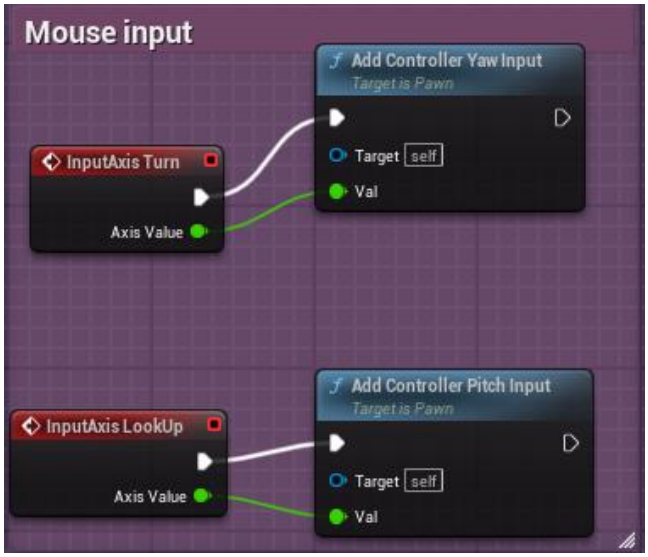
- **Left Shift Pressed/Released**: Ova dva čvora odgovaraju na pritisak i otpuštanje tipke *Left Shift*. Pritisak aktivira sprint, dok otpuštanje zaustavlja ovu akciju i vraća igrača na normalnu brzinu kretanja.
- **Is Falling i Branch logika**: Čvor *Is Falling* provjerava je li igrač u zraku. Ako igrač trenutno skače ili pada, sprintanje neće biti aktivirano, što se kontrolira pomoću *Branch* čvorova koji uvode logiku provjere uvjeta. Ako je igrač u zraku (uvjet „True“), funkcija sprintanja se neće aktivirati. Ako nije (uvjet „False“), omogućit će se sprintanje.
- **Animacija i smanjenje brzine nakon pucanja**: Dodatni *Branch* čvor provjerava je li igrač nedavno pucao (*Recently Fired*). Ako je igrač pucao, sprintanje se ne aktivira, a animacija se vraća u neutralno stanje pomoću *Stop Slot Animation* čvora.
- **Animacije i zvukovi**: U Blueprintu su uključeni čvorovi za upravljanje animacijom sprintanja. Koriste se *Stop Slot Animation* čvorovi kako bi se osiguralo da se animacije ispravno zaustave kada igrač prestane sprintati.

Također, u nastavku koda za sprint se postavljaju uvjeti koji onemogućavaju sprint dok igrač cilja sa oružjem, te da je sprint moguć samo dok se igrač kreće prema naprijed.



Slika 7: Aiming i Moving Forward *dio Blueprinta*

Slika 7. prikazuje dio *Blueprinta* koji se odnosi na provjeru smjera kretanja i nišanjenja (Aiming i Moving Forward) osiguravajući da igrač može sprintati samo u slučaju kretanja prema naprijed i ne korištenja funkcije nišanjenja (aiming). Za stvaranje logike koja omogućava sprintanje samo kada oba uvjeta budu ispunjena koristi se kombinacija *NOT* i *AND* čvorova – igrač mora hodati naprijed i ne smije biti u načinu nišanjenja. Dok *NOT* čvor provjerava stanje nišanjenja i onemogućava sprintanje ako je nišanjenje aktivno, *AND* čvor osigurava da igrač može sprintati samo ako su oba uvjeta zadovoljena (kretanje naprijed i nedostatak nišanjenja). Nakon provjere spomenutih uvjeta moguće je postaviti brzinu kretanja. Za promjenu brzine kretanja (engl. *Max Walk Speed*) koriste se SET čvorovi (postavljanje maksimalne brzine kretanja). U slučaju da se aktivira sprint (uvjet „True“), dolazi do povećanja brzine kretanja na 900 jedinica. Ako uvjet pak nije zadovoljen (uvjet „False“), brzina se vraća na normalnu brzinu hodanja od 600 jedinica.



Slika 8: Mouse Input

Postavljanje osi za miš (engl. *Mouse Input*)

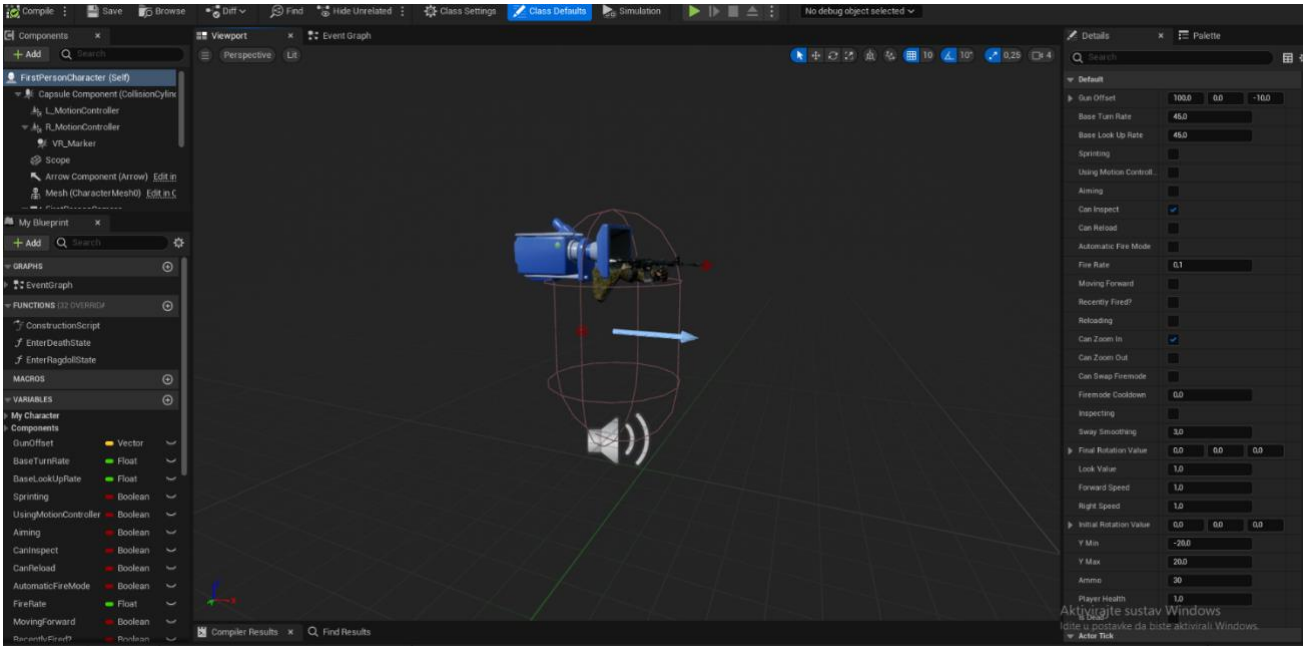
Kako bi se omogućilo upravljanje kamerom pomoću miša, potrebno je postaviti odgovarajuće inpute koji povezuju pokrete miša s rotacijom kamere unutar igre. To se postiže korištenjem dvije osi. Prva je InputAxis Turn i povezana je s horizontalnim pokretom miša (X-os). U slučaju pomicanja miša lijevo ili desno, ovaj input kontrolira horizontalnu rotaciju kamere, poznatiju kao *yaw*. Drugi čvor (os) je InputAxis LookUp i on je povezan s vertikalnim pokretom miša (Y-os). Kada igrač pomakne miš gore ili dolje, ovaj input kontrolira vertikalnu rotaciju kamere, poznatu kao *pitch*.

Primjena inputa za rotaciju kamere

- Kada su osjetnici pokreta miša postavljeni, oni se povezuju s funkcijama koje rotiraju kameru lika. Slika 8 prikazuje dvije glavne funkcije koje dodaju kontrolu rotacije. Jedna od njih je Add Controller Yaw Input koja predstavlja čvor za prihvat vrijednosti iz *InputAxis Turn* koja se koristi za dodavanje horizontalne rotacije (*yaw*) na kameru lika, što igraču omogućuje da se okreće lijevo i desno pomoću miša. Druga je Add Controller Pitch Input. Taj čvor prima vrijednost iz *InputAxis LookUp* i koristi je za dodavanje vertikalne rotacije (*pitch*) na kameru lika, što igraču omogućava da pomiče pogled gore i dolje.

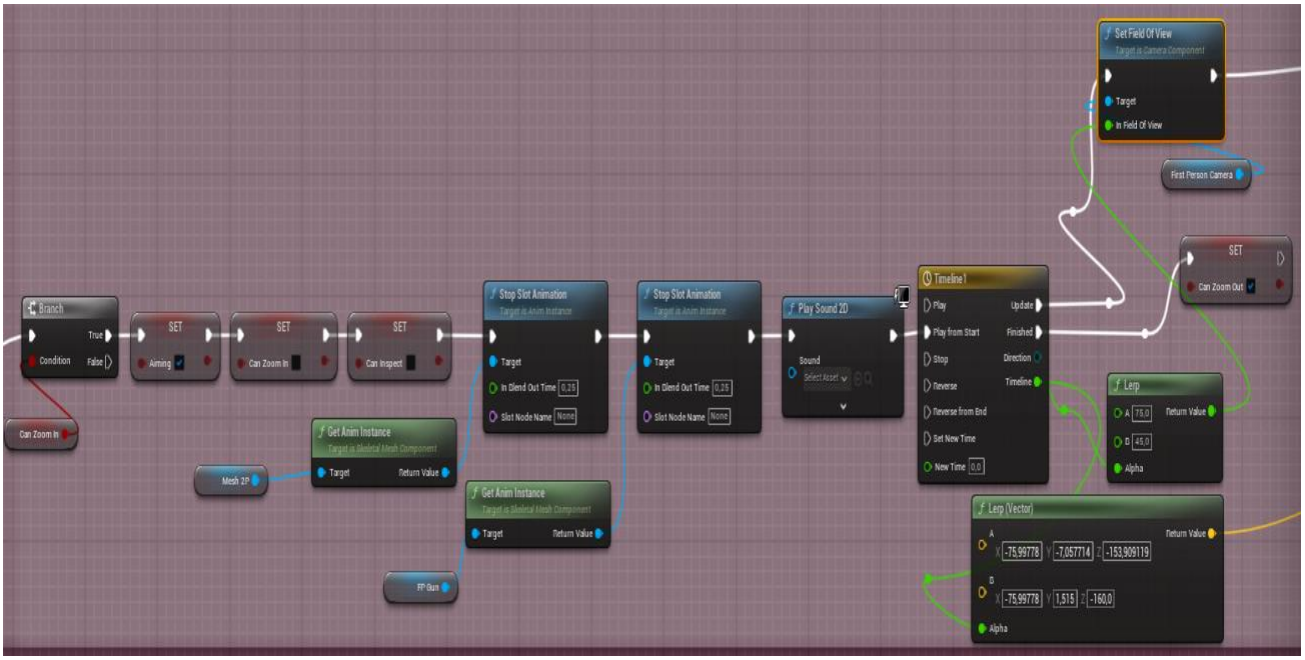
5. Implementacija funkcionalnosti oružja

Puška je osnovni alat igrača za borbu protiv neprijatelja, stoga je bitno pažljivo implementirati njezinu funkcionalnost, kao i njezin izgled unutar prvog lica. Puška korištena u projektu je, kao i ruke, preuzeta sa Unreal Marketplacea. Unutar First Person Charactera puška je spojena s rukama te je pozicionirana u odgovarajući položaj [6].



Slika 9: Pozicioniranje puške

Što se tiče pozicioniranja puške, na slici 9 je prikazan model smješten unutar ruku igrača. Puška je priključena na ruke kako bi pratila njihove pokrete. Uz pomoć Blueprinta može se precizno postaviti model u odnosu na ostale dijelove tijela lika. Varijabla kojom se definira položaj puške u odnosu na ruke lika je *Gun Offset* (prikazana na desnoj strani slike). U ovom slučaju, puška je pozicionirana na temelju vrijednosti offseta (100.0, 0.0, -10.0) kako bi se pravilno uskladila s animacijama ruku.



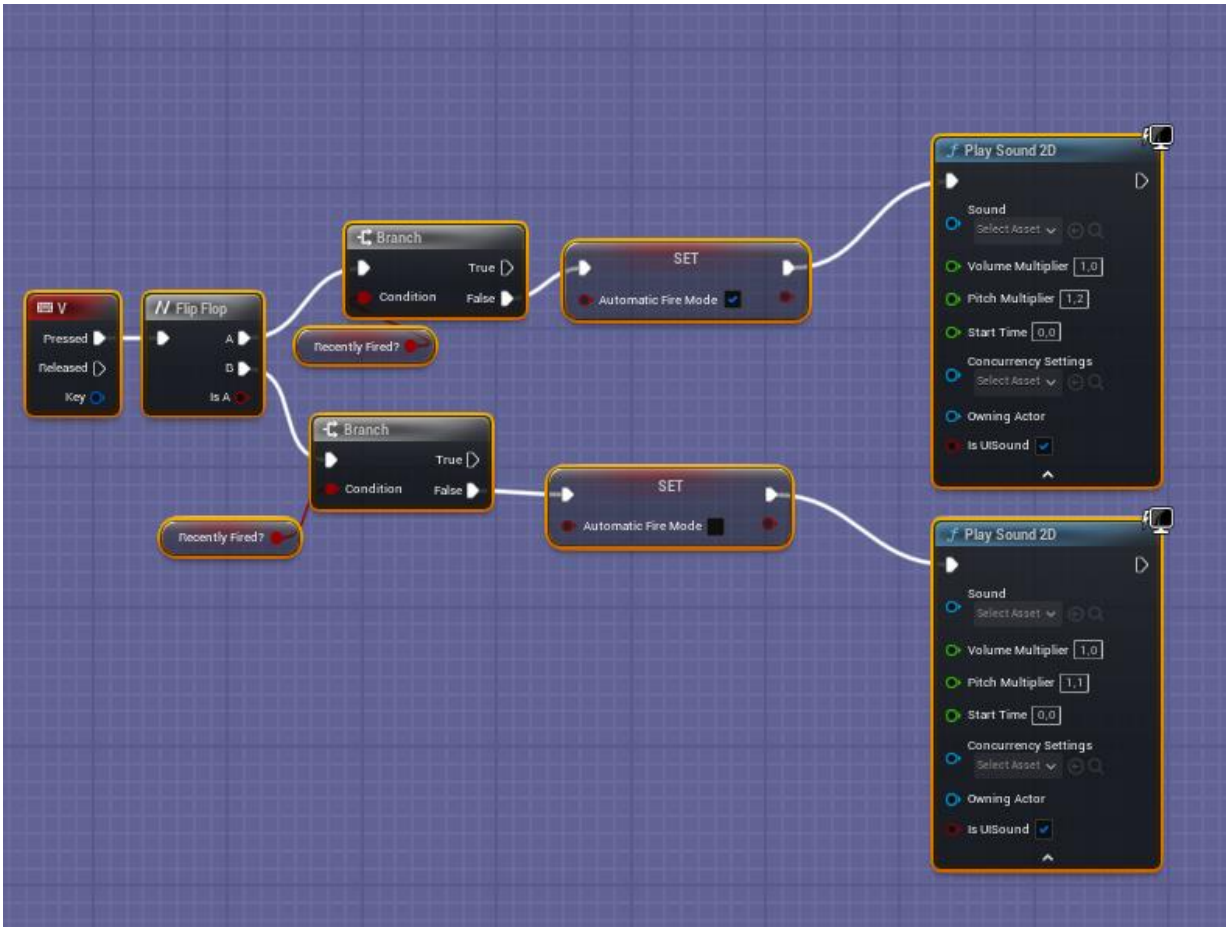
Slika 10: Zoom In i Zoom Out

Funkcija Zoom In

Prvi dio *Blueprinta* prikazan na slici 10 prikazuje mehaniku *Zoom In*, koja se aktivira kada igrač pritisne desnu tipku miša. Ova funkcionalnost omogućuje igraču približavanje cilju putem optičkog nišana na pušci. Za provjeru uvjeta zumiranja koristi se čvor u gornjem dijelu *Blueprinta* koji koristi *Branch* čvor za provjeru je li igrač u stanju da može zumirati (*Can Zoom In* varijabla). Nakon što su ispunjeni uvjeti (uvjet je True), započinje proces zumiranja. Jednako nakon potvrde uvjeta *Blueprint* pokreće specifične animacije i zvučne efekte.*Play Sound 2D* čvor koji reproducira zvuk prilikom prelaska u zumirani način ciljana, čime se dodaje realističan osjećaj. Za postavljanje kamere i Field of Viewa (FOV) ključno je da se u funkciji postavi pogled igrača. Čvor Field of View postavlja uži FOV, što stvara efekt zumiranja. Vrijednost FOV-a se smanjuje kako bi se simuliralo gledanje kroz optički nišan. Podešavanje položaja i rotacije puške osigurava točan vizualni dojam. Čvorovi *Set Relative Location and Rotation* ostavljaju novu poziciju i orijentaciju puške tijekom zumiranja. Ovaj korak je ključan kako bi puška izgledala prirodno dok igrač cilja.

Funkcija Zoom Out

Drugi dio *Blueprinta* prikazuje mehaniku *Zoom Out*, koja se aktivira kada igrač otpusti desnu tipku miša. Ova funkcija vraća pogled natrag na normalan način igranja bez optičkog nišana. Uvjet za vraćanje pogleda se provjerava pomoću čvora *Branch* provjerom varijable *Can Zoom Out*. Ako su uvjeti ispunjeni, funkcija vraća kameru u prvobitno stanje. Slično funkciji *Zoom In*, *Zoom Out* također pokreće specifične animacije i zvučne efekte. Ponovno se koristi čvor *Play Sound 2D* za zvučni efekt koji prati prijelaz iz zumiranog pogleda u normalni pogled. Za resetiranje kamere Field of Viewa, čvor *Set Field of View* ponovno postavlja širi FOV, vraćajući normalan pogled iz prvog lica. Vrijednost FOV-a se povećava kako bi se simuliralo vraćanje pogleda na standardno igranje. Konačno, čvorovi *Set Relative Location and Rotation* vraćaju pušku u prvobitni položaj i orijentaciju, čime se završava proces vraćanja iz zumiranog načina ciljana.



Slika 11: Akcija za promjenu načina pucanja

Akcija za promjenu načina pucanja

Mehanika promjene načina pucanja aktivira se pritiskom na tipku V kao što je prikazano na slici 11, kako je definirano u čvoru *InputAxis*. Kada igrač pritisne ovu tipku, aktivira se niz funkcija koje omogućavaju prelazak između različitih modova pucanja, kao što su automatski način i pojedinačno ispaljivanje.

Flip-Flop logika za promjenu moda

Flip-Flop čvor omogućuje izmjenično aktiviranje dva različita moda. Prilikom prvog pritiska na tipku V, mod pucanja se postavlja na automatski način pucanja (mod A). Prilikom sljedećeg pritiska na tipku, mod pucanja prelazi na pojedinačno ispaljivanje (mod B).

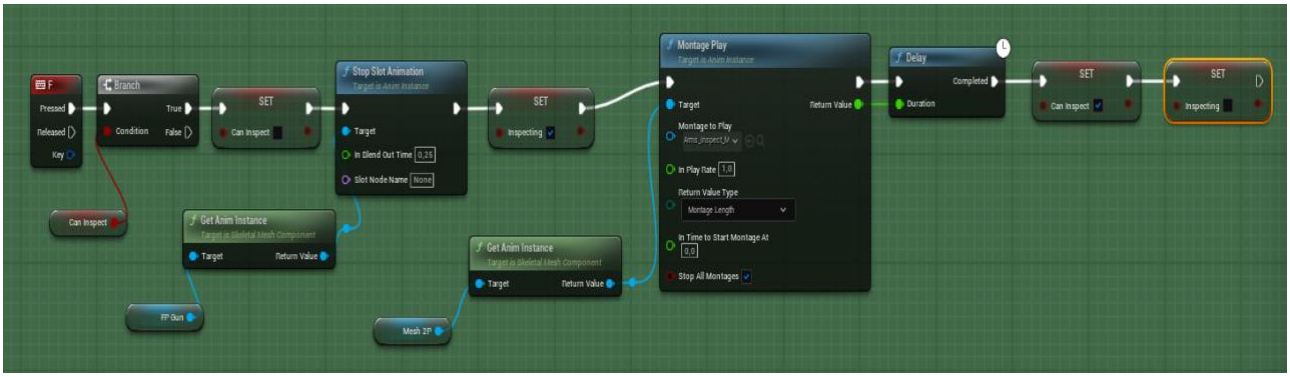
Postavljanje načina pucanja

Nakon što se Flip-Flop čvor aktivira, koristi se čvor *Set* za postavljanje varijable *Automatic Fire Mode* koja određuje je li način pucanja automatski ili pojedinačni. Za vrijednost T*rue*, oružje prelazi u automatski način pucanja, a za vrijednost *False*, oružje prelazi u način pojedinačnog ispaljivanja metaka.

Provjera stanja i uvjeta

Prije promjene načina pucanja, čvorovi *Branch* provjeravaju varijablu *Recently Fired*, kako bi se osiguralo da se način pucanja ne mijenja dok igrač aktivno puca. Ovo sprječava prekidanje trenutačnog pucanja prilikom promjene moda.

Mehanika pregledavanja oružja je estetska funkcionalnost koja omogućava igraču da pregleda svoje oružje u detalje unutar igre. Iako ova funkcija ne doprinosi direktno *gameplayu*, često se koristi kao dodatak imerzivnosti igre i može biti vizualno privlačna igračima. Na slici 12 je prikazan *Blueprint* koji implementira mehaniku pregledavanja oružja.



Slika 12: Blueprint mehanike pregledavanja oružja

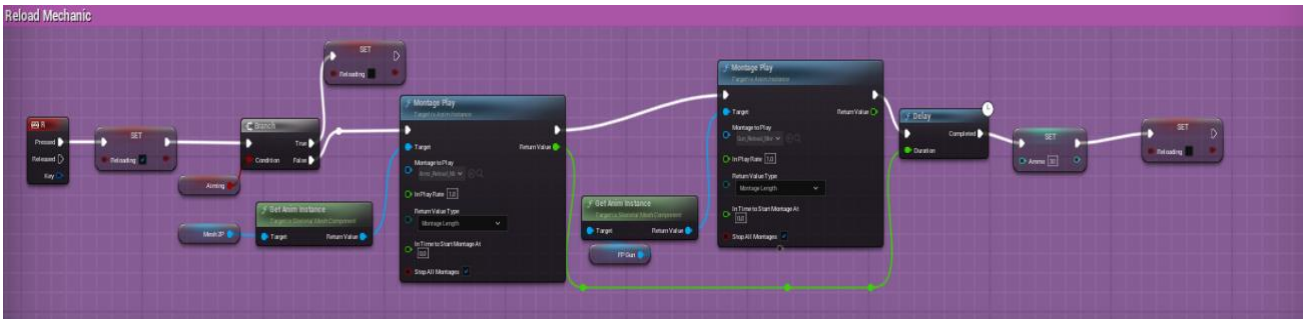
Za aktivaciju inspekcije oružja, koristi se input za pregledavanje (slika 12). Mehanika pregledavanja oružja je definirana u čvoru Input, te se aktivira tipkom E, koji detektira kada igrač pritisne ili otpusti ovu tipku. Čvor *Branch* se koristi za provjeru inspekcijskih uvjeta, vrijednost True varijable *Can Inspect* aktivira inspekciju. Spomenuti čvor omogućuje igraču pregled oružja samo u trenucima kada je to dozvoljeno (npr. kada ne puca ili nije u borbi).

Animacija pregledavanja

Za zaustavljanje trenutne animacije koristi se čvor *Stop Slot Animation*. To osigurava glatki prijelaz na novu animaciju pregleda oružja. Kod postavljanja inspekcijskog moda, varijabla *Inspecting* postavlja se na *True* kada animacija pregleda oružja započne. To omogućava praćenje stanja inspekcije tijekom cijele animacije. Čvor koji pokreće specifičnu animaciju pregleda oružja zove se Montage Play čvor. Animacija može uključivati rotaciju oružja, bliže detalje oružja ili bilo koju drugu akciju koju programer želi prikazati. Korištenje ovog čvora omogućuje izvođenje kompleksnih animacija koje su unaprijed definirane.

Vremenska kontrola i resetiranje

Nakon što animacija pregleda oružja završi, čvor *Delay* određuje koliko dugo će inspekcija trajati te se nakon isteka vremena varijabla *Inspecting* postavlja na *False*. Ovime je omogućena ponovna aktivacija mehanike pregledavanja oružja u budućnosti.



Slika 13: Blueprint za reload

Aktivacija ponovnog punjenja

Mehanika ponovnog punjenja aktivira se pritiskom na tipku R (slika 13). Ova radnja detektira se u čvoru *Input*, koji signalizira početak procesa ponovnog punjenja. Pritiskom tipke za ponovno punjenje, varijabla *Reloading* se postavlja na *True* i koristi se za praćenje stanja ponovnog punjenja, čime se osigurava da igrač ne može obaviti druge radnje (poput pucanja) dok traje proces ponovnog punjenja.

Provjera uvjeta za ponovno punjenje

Čvor *Branch* se koristi za provjeru uvjeta ponovnog punjenja prije početka animacije punjenja. Ako je igrač u načinu nišanjenja, vrijednost varijable *Aiming* je postavljena na *False,* može se pokrenuti animacija punjenja. Ovime je onemogućeno istovremeno nišanjenje i punjenje.

Animacija ponovnog punjenja

Čvor *Montage Play* ako bi se pokrenula animacija ponovnog punjenja koja može uključivati vađenje novog spremnika, umetanje metaka, te sve potrebne radnje za ponovno punjenje oružja.
U slučaju da oružje ima više animacija ponovnog punjenja (npr. kada je spremnik prazan i kada nije), *Blueprint* omogućuje pokretanje specifične animacije ovisno o situaciji.

Vremenska kontrola i resetiranje stanja

Po završetku animacije ponovnog punjenja, čvor Delay određuje trajanje cijelog procesa. Vrijednost trajanja ovog čvora odgovara vremenu potrebnom za ponovno punjenje oružja.
Završetkom procesa ponovnog punjenja, varijabla *Reloading* se vraća na *False*, što omogućava igraču da ponovno puca ili obavlja druge radnje. Također, varijabla *Aiming* se resetira, čime se omogućava igraču da ponovno cilja nakon što završi proces ponovnog punjenja.

Bliski napadi su čest dodatak borbenim mehanikama u FPS igrama, omogućujući igračima da koriste oružje za udarce u neposrednoj blizini kada su bez streljiva ili kada žele brzo eliminirati neprijatelja. Borbena mehanika obično uključuje animacije udarca i odgovarajuće zvučne efekte kako bi se napad učinio uvjerljivim. Slika 14 prikazuje implementaciju mehanike bliskog napada

Slika 14: Implementacija mehanike bliskog napada

Aktivacija bliskog napada

Mehanika bliskog napada aktivira se pritiskom na srednji gumb miša. Radnja se detektira pomoću *InputAxis* čvora, koji započinje animaciju bliskog napada kada igrač pritisne odgovarajuću tipku.

Pokretanje animacije bliskog napada

Nakon aktivacije, koristi se čvor *Montage Play* za pokretanje animacije za bliski napad. Animacija se odigrava na skeletnom modelu ruku ili puške igrača, ovisno o tome kako je mehanika konfigurirana. Prvi čvor *Montage Play* čvor pokreće animaciju bliskog napada ruku igrača (Arms_Melee_Attack). Animacija uključuje pokrete ruku koji simuliraju udarac ili sličnu akciju u bliskom kontaktu. Pri animaciji puške (Gun_Melee_Attack), drugi čvor *Montage Play* pokreće animaciju bliskog napada za pušku igrača, koja može uključivati udarac kundakom ili drugim dijelom oružja.

Pokretanje animacija: Oba čvora *Montage Play* koriste se za pokretanje specifičnih animacija za različite dijelove modela. Na ovaj način, igračev bliski napad može uključivati pokrete ruku i puške, što doprinosi realizmu.

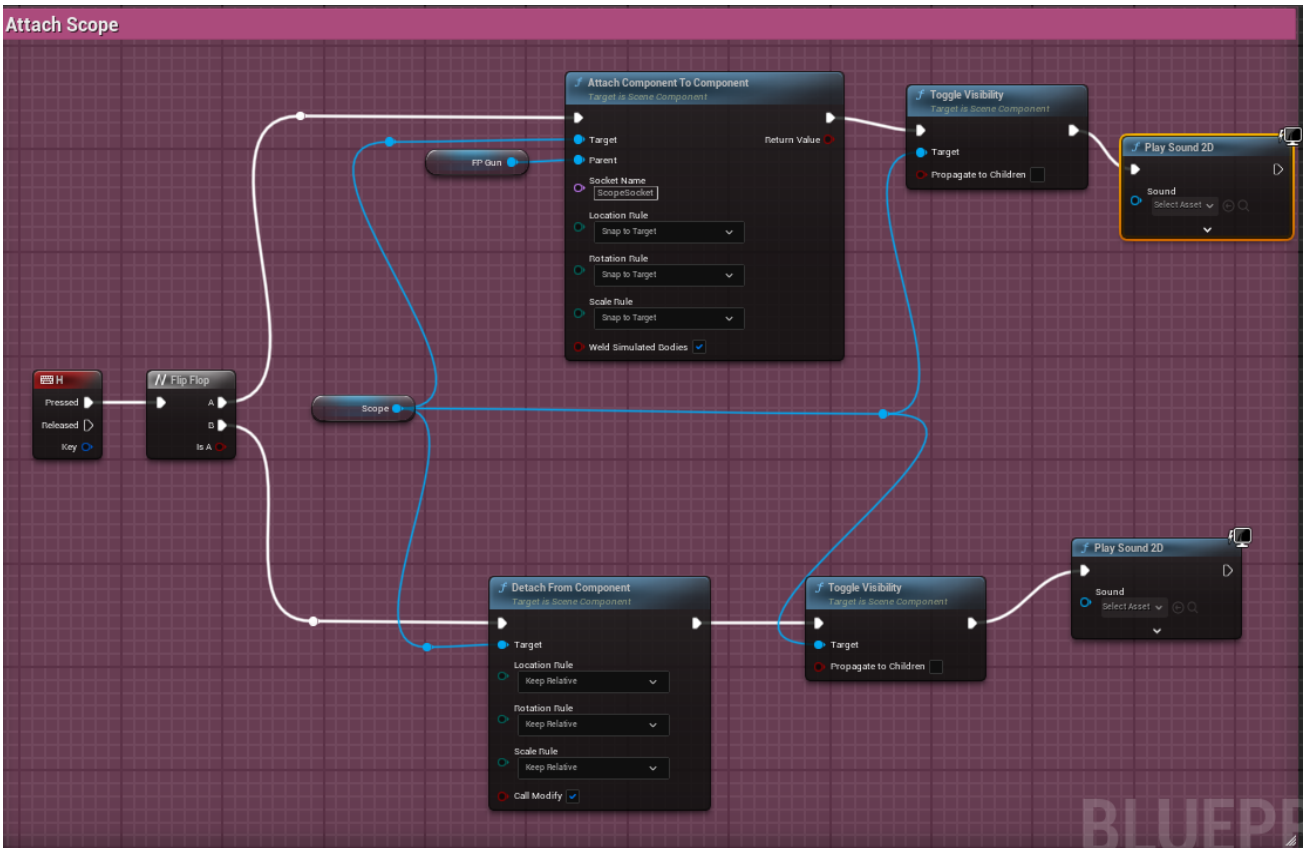
Zvučni efekti bliskog napada

Pokretanjem animacije bliskog napada koristi se čvor *Play Sound 2D* za reproduciranje odgovarajućeg zvučnog efekta. Zvučni efekt može biti udarac, zvuk kontakta oružja s objektom ili bilo koji drugi zvuk koji pojačava osjećaj udarca.

Postavljanje inspekcijske varijable

Završetkom bliskog napada postavlja se varijabla *Can Inspect* na *False*. Varijabla osigurava da igrač ne može pregledavati oružje tijekom izvođenja bliskog napada, čime se sprječava preklapanje mehanika inspekcije i napada.

U modernim FPS igrama, mogućnost dodavanja i uklanjanja dodataka poput optičkog nišana na oružje omogućuje igračima prilagodbu njihovog stila igranja. Mehanika pridodavanja nišana, prikazana na slici 15, koristi *Blueprint* sustav unutar *Unreal Enginea* za dinamičko postavljanje i uklanjanje nišana na oružje.



Slika 15: Mehanika pridodavanja nišana

Aktivacija mehanike (H Tipka)

Mehanika pridodavanja nišana na oružje aktivira se pritiskom na tipku H, nakon čega se čvor *Flip-Flop* prebacuje između dvije različite funkcije - dodavanje nišana na oružje ili njegovo uklanjanje. Čvor omogućava naizmjenično izvršavanje dviju različitih akcija ovisno o prethodnom stanju. Prvi pritisak na H dodaje a drugi uklanja nišan.

Dodavanje nišana na oružje

Ako je prvi uvjet zadovoljen (prvi pritisak na H), koristi se čvor *Attach Component To Component* za pričvršćenje nišana na unaprijed definirani *socket* na oružju nazvan *ScopeSocket*.

Promjena vidljivosti nišana

Čvor *Toggle Visibility* osigurava vidljivost pričvršćenog nišana. Vidljivost je ključna da igrač može jasno vidjeti nišan na oružju.

Zvuk dodavanja nišana

Čvor *Play Sound 2D* reproducira zvučni efekt koji signalizira uspješno dodavanje nišana. Ovaj zvuk pomaže igraču da dobije povratnu informaciju o dodavanju dodatka na oružje.

Uklanjanje nišana s oružja

Drugi pritisak na tipku H pokreće proces uklanjanja nišana s oružja. Čvor *Detach From Component* koristi se za odvajanje nišana od oružja. Ova funkcija osigurava da nišan više nije pričvršćen za *socket* i može se ukloniti iz trenutne pozicije na oružju.

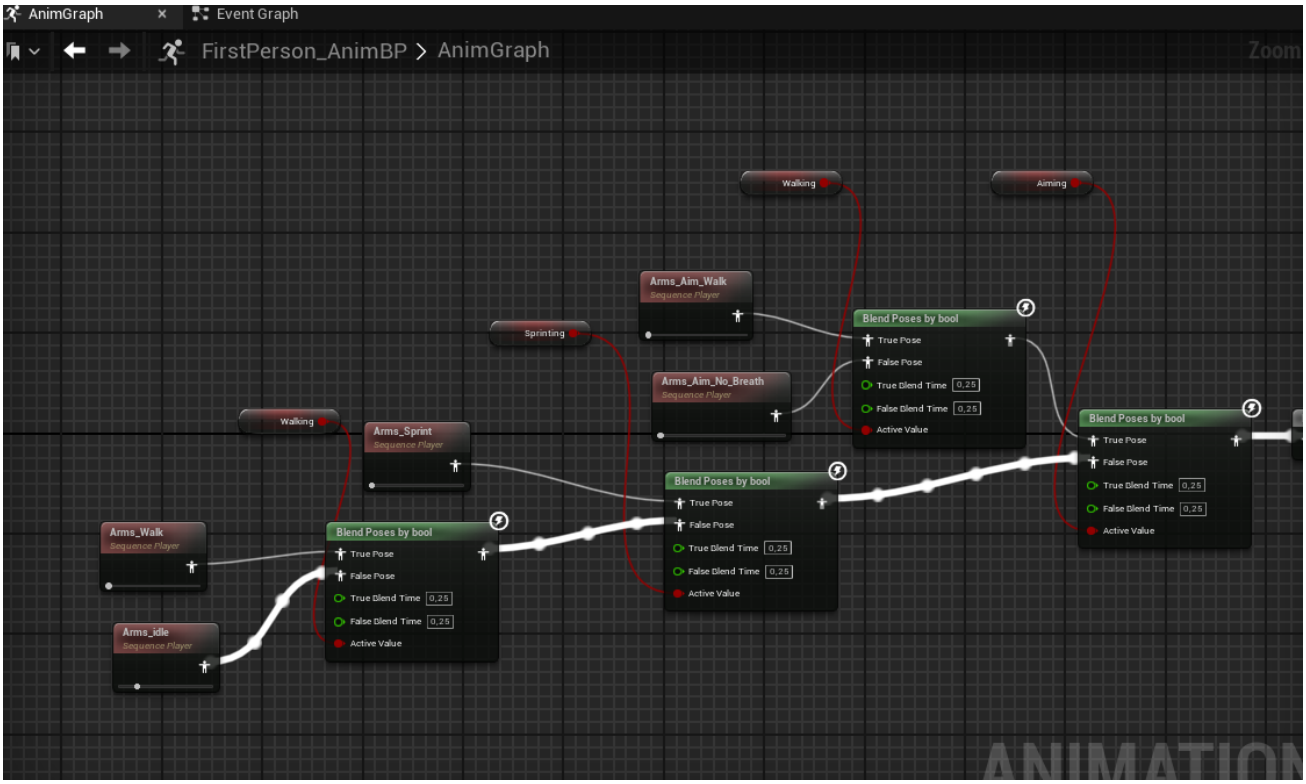
Skrivena vidljivost nišana

Nakon uklanjanja nišana, čvor *Toggle Visibility* onemogućuje vidljivost nišana igraču.

6. Animacije

Postavljanjem igrača i njegova kontrolera može se uočiti da sve radnje izgledaju kruto i neprirodno. Za stvaranje realističnije slike i ugodnijeg iskustva igranja potrebno je dodati animacije. Animacije korištene u ovome projektu preuzete su sa Unreal Marketplacea. Mogu se podijeliti u nekoliko glavnih kategorija: animacije lika (glavni pokreti lika, kao što su hodanje, trčanje, skakanje i interakcije s okolinom), animacije oružja (animacije rukovanja oružjem, kao što su pucanje, ponovno punjenje, nišanjenje i bliski napadi) te animacije neprijatelja (AI; upravljaju ponašanjem neprijatelja, uključujući hodanje, napadanje, primanje štete i smrtne animacije).

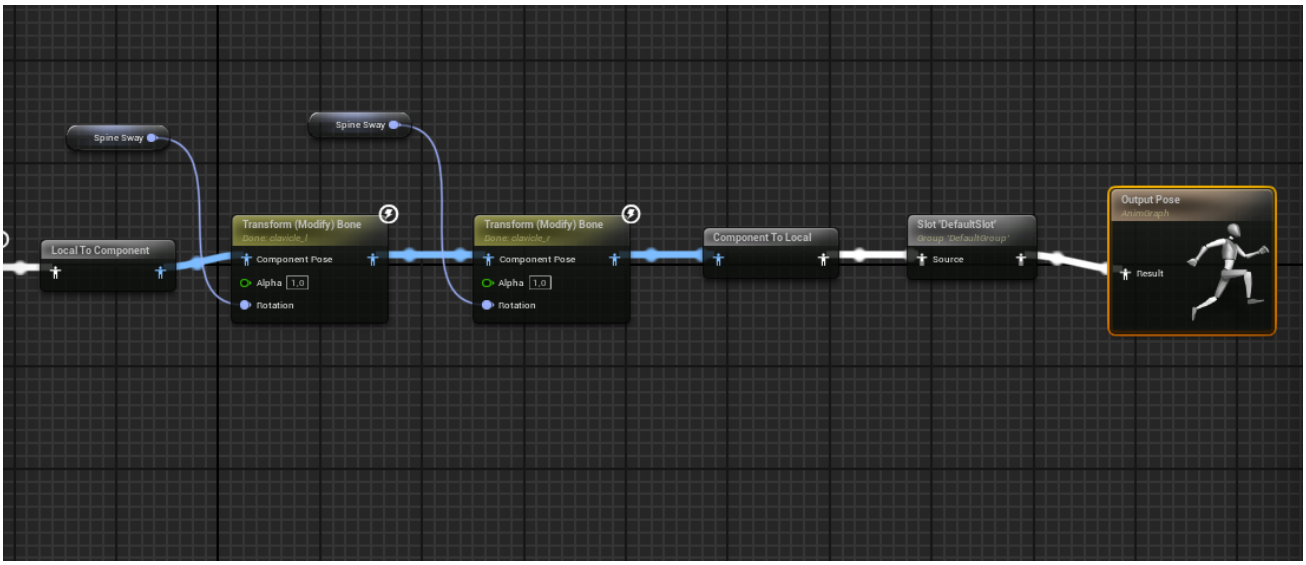
U Unreal Engineu, animacije su definirane kroz *Animation Blueprints*, što omogućuje povezivanje raznih animacija s likom i oružjem. Svaki model lika i oružja ima pripadajući skeleton mesh koji definira kosti i zglobove za animacije. Korištenjem *Blend Space* alata unutar *Unreal Enginea*, moguće je spojiti različite animacije u jednu funkcionalnu cjelinu. Na primjer, animacija hodanja može se neprimjetno spojiti s animacijom trčanja ovisno o brzini lika.



Slika 16: Animacije

U Unreal Engineu, animacije za likove i oružja kontroliraju se pomoću *Anim Graph* unutar *Animation Blueprinta*. Ovaj graf omogućuje povezivanje različitih animacija ovisno o stanju igrača (trčanje, hodanje, pucanje itd.). Slika 16 prikazuje kako su animacije za hodanje, trčanje i nišanjenje povezane putem *Blend Poses by Bool* čvorova.

- **Animacije hodanja i trčanja**: Kada igrač aktivira određeni input (npr. hodanje ili sprintanje), *Anim Graph* određuje koja će se animacija igrati. Na primjer, animacija *Arms_Walk* se aktivira kada igrač hoda, dok se *Arms_Sprint* aktivira kada igrač trči. Ove animacije su povezane pomoću čvorova *Blend Poses by Bool* koji omogućavaju glatki prijelaz između animacija ovisno o stanju lika.
- **Animacije nišanjenja**: Kada igrač koristi nišanjenje, aktiviraju se specifične animacije ruku (*Arms_Aim_Walk* i *Arms_Aim_No_Breath*) koje prikazuju držanje oružja u stanju pripravnosti. Pomoću *Blend Poses by Bool*, animacije se preklapaju sa standardnim animacijama hodanja ili trčanja, ovisno o tome što igrač radi dok cilja.



Slika 17: Dinamička prilagodba animacija

Slika 17 prokazuje struktura upravljanja modifikacijom kostiju kako bi se osigurale dinamičke prilagodbe animacija na temelju položaja lika u svijetu.

- **Transform (Modify) Bone**: Ovi čvorovi omogućuju manipulaciju specifičnim kostima unutar skeleta lika, poput rotacije kralježnice (engl. *Spine Sway*). Na ovaj način,

prilagođava se položaj gornjeg dijela tijela kako bi odgovarao stanju lika (na primjer, prilikom ciljanja).

- **Local to Component i Component to Local**: Ovi čvorovi koriste se za pretvorbu pozicija između lokalnog i komponentnog prostora, omogućavajući modifikacije kostiju na osnovu globalnog položaja lika u svijetu igre. To je važno za precizno kontroliranje animacija u stvarnom vremenu.
- **Slot 'DefaultSlot'**: Ovaj čvor koristi se za prikazivanje različitih animacijskih montaža (engl. *montage*). Na taj način se omogućuje upravljanje animacijama kao što su udarci, pucanje i druge radnje koje se izvršavaju preko animacijskih montaža.

Output Pose

- **Output Pose**: Na kraju svakog *AnimGrapha*, *Output Pose* čvor kombinira sve animacijske podatke i prikazuje konačnu animaciju lika. Sve modifikacije kostiju i prijelazi između animacija završavaju u ovom čvoru, čime se osigurava da animacija teče glatko i prikazuje ispravne pokrete u svakom trenutku.

7. Inteligentni protivnik

'

Protivnik, u ovome slučaju zombiji, su jedan od najvažnijih faktora koji stvara doživljaj igranja ugodnijim, izazovnijim te poticajnijim sa ispravnom izvedbom. Za protivnika je potrebno da reagira na igračevu prisutnost te da ga prati i pokušava ubiti.

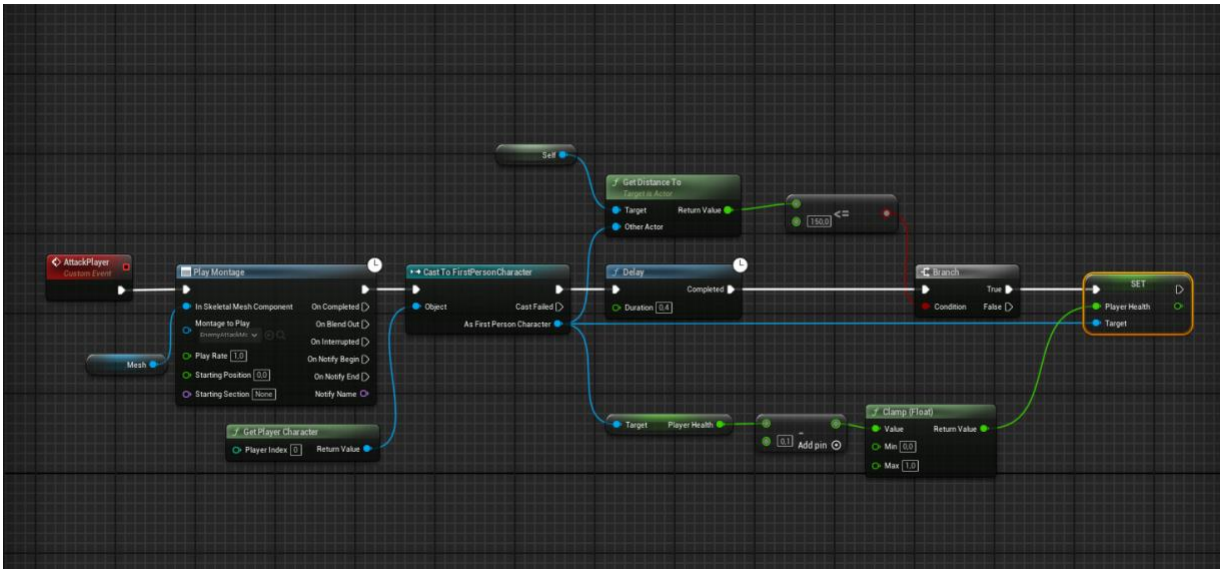
Praćenje igrača

Zombiji su programirani da detektiraju i prate igrača kada se nađu unutar određenog radijusa. Ovo se postiže pomoću sustava *NavMesh* (zeleno područje, slika 18) koji omogućuje neprijateljima navigaciju kroz 3D prostor igre.

Slika 18: Praćenje igrača

Napadanje igrača

Kada se neprijatelji nađu unutar dometa napada, prelaze iz stanja praćenja u stanje napada. Prijelaz uključuje aktiviranje animacija napada što su udarci rukama u ovom slučaju.



Slika 19: Napadanje igrača

Custom Event "AttackPlayer"

Napad neprijatelja započinje aktivacijom prilagođenog događaja pod nazivom *AttackPlayer* (slika 19). Događaj može biti pozvan unutar *Behavior Tree* sustava neprijatelja, što omogućava dinamičnu aktivaciju napada na temelju uvjeta u igri (npr. kada je igrač u blizini).

Play Montage

Pokretanjem napada, koristi se čvor *Play Montage* za odigravanje unaprijed definirana animacije napada neprijatelja. Animacija kontrolira vizualne pokrete neprijatelja, kao što su udarci ili zgrabi koji napad čine realističnim.

Animacija za napad

Unutar čvora *Play Montage* postavljena je specifična animacija napada (EnemyAttackMC) koja se reproducira pomoću skeletne mreže (*Skeletal Mesh Component*) pridružene neprijatelju.

Get Distance To

Pokretanjem animacije napada, čvor *Get Distance To* izračunava udaljenost između neprijatelja i igrača te uspoređuje poziciju neprijatelja i igrača.

Čvor Branch za provjeru uvjeta

Čvor *Branch* koristi se za provjeru udaljenosti. Ako je igrač unutar dometa napada (u ovom slučaju manje od 150 jedinica), logika nastavlja dalje i igrač prima štetu. Ako je igrač izvan dometa, napad neće uzrokovati štetu.

Cast To FirstPersonCharacter

Neprijatelj može utjecati na igrača preko čvora *Cast To FirstPersonCharacter* koji omogućava pristup varijablama igrača, uključujući razinu zdravlja.

Postavljanje zdravlja (Player Health)

Uspješnim pogotkom igrača, vrijednost igračevog zdravlja se smanjuje preko čvora *Set*. U ovom slučaju, zdravlje se smanjuje za 0,1 jedinica.

Clamp (Float)

Čvor *Clamp* osigurava da vrijednost zdravlja ne padne ispod nule ili ne prijeđe maksimalnu vrijednost 1.

Čvor Delay

Čvor *Delay* koji omogućava vremensku pauzu između animacije napada i stvarnog smanjenja zdravlja. U ovom primjeru, pauza je postavljena na 0,4 sekunde, što omogućava da animacija udarca završi prije nego što se primijeni šteta.

Zaključak

Kroz ovaj završni rad, ostvarena je izrada FPS survival igre koristeći Unreal Engine,. Fokus igre bio je na preživljavanju u post-apokaliptičnom okruženju, borbi protiv horda zombija vođenih umjetnom inteligencijom, te efikasnom upravljanju resursima. Implementacija različitih mehanika, kao što su kontrola lika, interakcije s okolinom, animacije i ponašanje neprijatelja, pokazala se kao izazovan, ali istovremeno i vrlo poučan proces.

Korištenjem Unreal Enginea demonstrirane su mogućnosti ovog pogonskog alata, posebno u području animacija, dinamičkog osvjetljenja, i fizičke simulacije. Također, implementacija umjetne inteligencije kroz *Behavior Tree* sustave omogućila je stvaranje neprijatelja koji dinamično reagiraju na igrača, čineći iskustvo igranja intenzivnijim i nepredvidivim.

S obzirom da je glavna poanta bila programiranje, izgled likova, okoline te zvuk su uvezeni korištenjem Unreal Engine tržišta. Daljnji razvoj ove igre mogao bi uključivati proširenje svijeta igre, dodavanje novih vrsta neprijatelja i dodatnih funkcionalnosti koje bi obogatile iskustvo igranja.

Ovaj projekt je pružio dublji uvid u proces razvoja videoigara i potvrdio je koliko su koordinacija i kreativnost važni elementi u stvaranju imerzivnih i inovativnih igara. Unreal Engine se pokazao kao izuzetno moćan alat, koji omogućuje brzi razvoj kompleksnih i vizualno privlačnih igara, te se može koristiti kao platforma za buduće projekte.

Literatura

[1] „Izrada platformske igre u Unreal Engine razvojnom okruženju“
Dohvaćeno iz: https://repozitorij.oss.unist.hr/en/islandora/object/ossst:1798
(posjećeno 20.08.2024.)

[2] Službena stranica Unreal Engine
Dohvaćeno iz: https://www.unrealengine.com/en-US (posjećeno 20.08.2024.)

[3] „Alati i platforme za razvoj računalnih igara“
Dohvaćeno iz: https://repozitorij.unizg.hr/islandora/object/foi:6379
(posjećeno 21.08.2024.)

[4] „Usporedba okruženja za izradu videoigara“
Dohvaćeno iz: https://zir.nsk.hr/islandora/object/unipu:7489
(posjećeno 21.06.2024.)

[5] „The best survival games on PC“
Dohvaćeno iz: https://www.pcgamer.com/the-best-survival-games-on-pc/
(posjećeno 23.08.2024.)

[6] „Learning Unreal Engine Game Development”
Dohvaćeno iz: https://tinyurl.com/27mbere5 (posjećeno 23.08.2024.)

[7] “Unreal Engine: Game Development from A to Z”
Dohvaćeno iz: https://tinyurl.com/2cdbh357 (posjećeno 23.08.2024.)

[8] „uniDOOM- rekreacija DOOM igre u Unity pogonskom alatu”
Dohvaćeno iz: https://repozitorij.oss.unist.hr/en/islandora/object/ossst%3A2159
(posjećeno 07.09.2024.)

Popis slika