Izrada igre iz prvog lica u programskom alatu Unreal Engine 4

Filipović, Filip

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:953721

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-04-19**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Filipović

IZRADA IGRE IZ PRVOG LICA U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Filipović

Matični broj: 45922/17–R

Studij: Informacijski sustavi

IZRADA IGRE IZ PRVOG LICA U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI/DIPLOMSKI RAD

Mentor/Mentorica:

Dr. sc. Mladen Konecki

Varaždin, rujan 2020.

Filip Filipović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski 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

Tema završnog rada je izrada računalne igre, točnije pucačine u prvom licu u programskom alatu *Unreal Engine 4*. U radu ćemo se doteći žanra igara u prvom licu (i njihovog porijekla) i opisa samog alata *Unreal Engine* kao i njegovih verzija. Nakon toga slijedi detaljan opis izrađene igre pri čemu posebnu važnost posvećujemo odabranim sistemima i algoritmima koje igra koristi, a to su: kretanje i kamera, mogućnost uzimanja oružja, odabir oružja, pucanje iz oružja, AI neprijatelja, detekcija kolizije, upravljanje valovima neprijatelja i korisničko sučelje. Cijeli rad okončan je zaključkom gdje je opisano moje iskustvo izrade prve 3D igre.

Ključne riječi: računalna igra; igra iz prvog lica; Unreal Engine 4; algoritmi; programiranje;

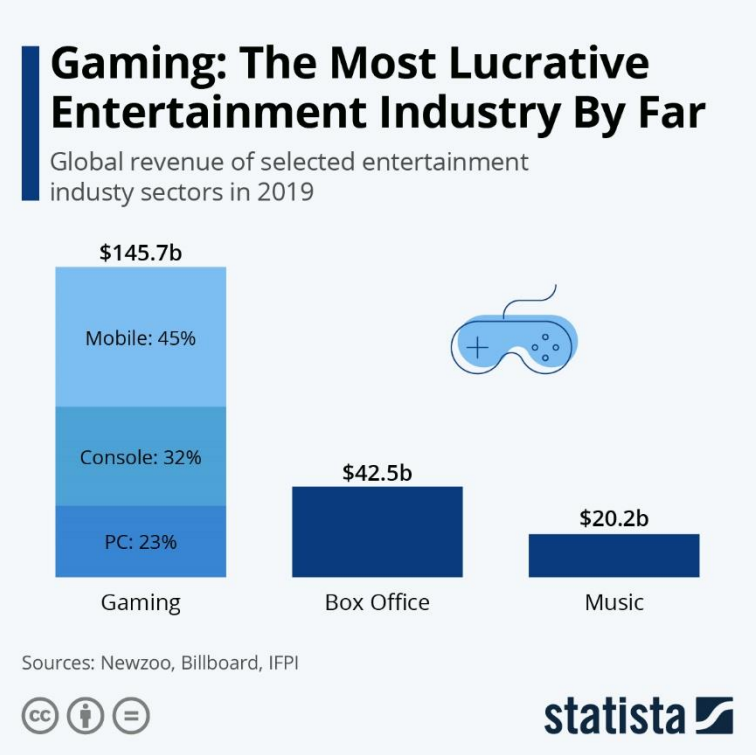
Sadržaj

Sadržaj iii
1. Uvod 1
2. Igre (pucačine) iz prvog lica 2
2.1. Podrijetlo FPS igara 2
2.2. Rano doba FPS igara 2
2.3. Porast popularnosti 3
2.4. Druga polovica 90-ih godina i početak 21. stoljeća 4
2.5. Današnje doba FPS igara 5
3. Unreal Engine 6
4. Izrada pucačine u prvom licu 8
4.1. Kontrole, kretanje i kamera 9
4.2. Oružja 12
4.3. Neprijatelji (AI) 19
4.4. Kolizija i sistem valova neprijatelja 23
4.4.1. Kolizija između objekata 23
4.4.2. Sistem valova neprijatelja 25
4.5. Korisničko sučelje 26
5. Zaključak 28
Popis literature 29
Popis slika 30
Izvori slika 31

1. Uvod

Videoigre danas su svakodnevica mnoge djece i odraslih diljem svijeta. Kao osoba koja je odrasla uz videoigre, smatram da su iste vrlo korisne i odlične za razvijanje motorike i brzine (posebice igre pucačina iz prvog lica) unatoč tome što određene igre sadrže eksplicitni sadržaj i nisu namijenjene svima. Iako mnogi videoigre vežu uz nasilje, valja napomenuti kako postoje i igre edukativne svrhe kao i igre mozgalice (eng. *puzzle games*) gdje je potrebno razmišljati da bi se postigla određena razina.

Činjenica da je industrija videoigara unazad nekoliko godina najunosnija što se tiče prihoda u sektoru zabavnog sadržaja (uz glazbu, film i ostale) govori o tome koliko su videoigre rasprostranjene, značajne i koliki utjecaj imaju na ljude. [4]



Slika 1: Najunosnija zabavna industrija do sada

Također je zanimljiva činjenica da danas mnogi zarađuju igrajući videoigre, bilo putem prijenosa uživo (eng. *streaming*) ili putem popularne mreže YouTube. Uz navedene načine zarade, *eSports* scena je iz dana u dan sve popularnija, a na njoj uz strateške igre, prevladavaju i pucačine u prvom licu. Prema danim informacijama, vrlo je jednostavno zaključiti da su igre iz prvog lica današnji standard u svijetu videoigara.

2. Igre (pucačine) iz prvog lica

Pojam igre iz prvog lica najčešće podrazumijeva igru pucačine iz prvog lica (eng. *First-person shooter - FPS*) koja je žanr igara gdje je fokus stavljen na borbu oružjem, a igrač svu akciju doživljava u očima lika kojeg kontrolira. Pucačina u prvom licu ne mora imati većih razlika od ostalih vrsta pucačine (poput pucačina u trećem licu), stoga se pucačina u prvom licu smatra potkategorijom akcijskih igara.

2.1. Podrijetlo FPS igara

Cijeli žanr započeo je igrom *Maze* (kasnije i *Maze War*) za koju se smatra da je prva igra takve vrste čiji nas razvoj vraća u davnu 1973. godinu, a razvili su je G. Thompson, S. Colley i H. Palmer, tadašnji srednjoškolci u NASA-inom obrazovnom programu. Godinu dana kasnije razvijena je i simulacija leta svemirskim brodom koja se igrala u načinu s više igrača (eng. *multiplayer*), točnije 32 igrača, a ujedno je bila i vrsta pucačine u prvom licu. Kretanje u prvim FPS igrama bilo je ograničeno, a svodilo se na kretanje unaprijed i unazad te okret za 90 stupnjeva što bi činilo kretanje po koordinatnim osima. [5]

2.2. Rano doba FPS igara

Iako se *Maze* smatra pretkom FPS žanra, *MIDI Maze* je prvi FPS koji je prikazao prvo mrežno sučelje za više igrača sa *deathmatch* načinom igranja. *Deathmatch* predstavlja način igre gdje svatko igra za sebe, a sama igra objavljena je 1987. godine za Atari ST, 35. godina starom Atari računalu. Dizajn igre bio je u stilu labirinta, a dizajn likova sličan igri Pac-Man. *MIDI* dolazi od naziva sučelja kojeg igra koristi za mrežnu komunikaciju, a *Maze* kao svojevrsna referenca na originalni *Maze*. [5]

Slika 2: MIDI Maze

2.3. Porast popularnosti

Wolfenstein 3D, igra kompanije *id Software*, zaslužan je za današnju popularnost FPS igara. Za razliku od prethodnika, nudio je slobodno kretanje i kameru (za razliku od okretanja po 90 stupnjeva) te osim popularnosti, mnogi smatraju da je zaslužan i za razvoj cijelog FPS žanra. Predstavljen je u *shareware* izdanju, što označava vrstu softvera koji se dijeli besplatno i potiče korisnike na dijeljenje kopija. Zanimljiva je činjenica da je *Wolfenstein 3D* zabranjen u Njemačkoj zbog upotrebe nacističke ikonografije pa je verzija na konzoli SNES imala drugačije modele gdje su napadačke pse zamijenili divovskim štakorima. [5]

Slika 3: Wolfenstein 3D

Slična igra *Wolfenstein 3D*-u bila je *Doom*, objavljena 1993. u istom *shareware* izdanju kao i *Wolfenstein 3D*. *Doom* je predstavio poboljšane teksture i efekte koji su poboljšali i samu atmosferu igre, a uz *Wolfenstein 3D*, smatra se najvažnijom FPS igrom koja je ikad napravljena. Za razliku od godinu dana mlađeg suparnika, *Doom* je bio vrlo utjecajan ne samo na FPS žanr, već općenito na videoigre. Označio je prekretnicu *multiplayer* igara koje su danas sastavni dio FPS žanra. Dizajn igre svodi se na krvavo nasilje, mračni humor i slike vezane uz pakao što je *Doom*-u pridonijelo velik broj pohvala kritičara, a popularni sinonim mu je simulator ubijanja. [5]

Slika 4: *Doom*

2.4. Druga polovica 90-ih godina i početak 21. stoljeća

Razdoblje nakon *Doom*-a označilo je napredak u 3D grafici FPS igara i načinu igranja za više igrača. Prva takva igra bila je *Quake* koji je bio jedan od prvih FPS-ova gdje su igrači organizirali klanove što je potaknulo organiziranje LAN partija gdje bi se igrači okupljali na fizički istom mjestu i igrali jedni protiv drugih. *Quake* je ubrzo nakon izlaska dobio i svoj nastavak, a dvije godine nakon nastavka, izašao je *Quake III Arena*, igra koja je doživjela veliku popularnost, a specifična je po brzom i dinamičnom stilu igre. [5]

Na temelju *Quake*-ove grafičke tehnologije, *Valve* je 1998. objavio FPS igru pod nazivom *Half-Life* koji je postao komercijalni uspjeh bez presedana. *Half-Life* se isticao od ostalih FPS igara u svoje vrijeme. Karakterizirala ga je odlično napisana priča (koju konkurencija u ono vrijeme nije imala), odlično napravljeni neprijatelji i njihova umjetna inteligencija, odabir oružja i pažnja prema detaljima koju je *Valve* posvetio. Prema popularnom portalu *GameSpot*, *Half-Life* je prepoznat kao „jedna od najboljih igara svih vremena“. [5]

Half-Life je 1999. godine dobio modifikaciju pod imenom *Counter-Strike* koja je, što se tiče multiplayer segmenta FPS igara, definirala novu vrstu igranja. Tako se je popularnom *deathmatch* i *capture the flag* načinu (koji se pojavio i u *Quakeu*) pridružio i način pod nazivom *search and destroy* gdje je cilj timu terorista postaviti bombu, a timu anti-terorista postavljenu bombu spriječiti od eksplozije.

Slika 5: *Half-Life*

2.5. Današnje doba FPS igara

U današnje vrijeme postoji veliki broj igara koje su nastale kao nastavak igara iz prošlosti. *Half-Life* je dobio svoj nastavak koji je bio manje utjecajan, no generalno je bio kvalitetniji i impresivniji. *Counter-Strike* je dobio nekoliko nastavaka, od kojih je *Global Offensive* i dan danas standard *multiplayer* segmenta FPS igara i eSports scene. Osim FPS igara koje se igraju na ograničenim mapama (poput *Counter-Strikea*), pojavile su se i igre koje imaju tzv. otvoreni svijet (eng. *open world*) u kojima igrač nije „zarobljen“ u četiri zida, već je slobodan i može istraživati svijet oko sebe. Primjer takvih igara su *Far Cry*, *Fallout* i *Borderlands* koje su primarno zamišljene za način igranja jedne osobe.

Kako se igre otvorenog svijeta vežu uz *singleplayer* segment FPS igara, od 2017. godine aktualna vrsta FPS igara postala je *Battle Royale*, gdje do 100 ljudi iskače na istu mapu i bori se za opstanak. U takvom načinu igre pobjeđuje osoba koja ostane zadnja živa, a primjeri takvih igara su *Apex Legends*, *PlayerUnknown's Battlegrounds* i *Call of Duty: Modern Warfare*.

Prema danim podacima, možemo samo zaključiti da su FPS igre danas veoma popularne i jedne od vodećih u cijelog industriji videoigara. Uz popularnost, lako je i zaključiti da postoji jako puno vrsta FPS igara, što također ukazuje na mogućnosti koje žanr nudi.

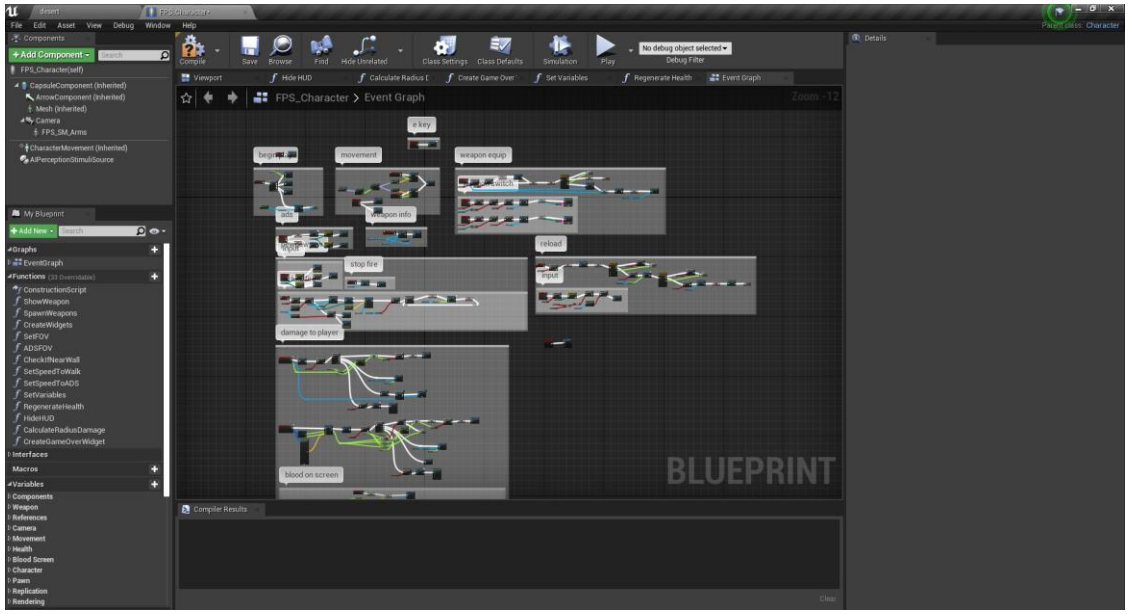
3. Unreal Engine

Unreal Engine je alat za izradu i razvoj videoigara (eng. *game engine*) razvijen od strane *Epic Games*-a. Prvi puta je prikazan 1998. godine u FPS igri po imenu *Unreal* (iz koje je kasnije proizašao serijal igara *Unreal Tournament*). U današnje vrijeme, *Unreal Engine* se koristi u svim žanrovima videoigara, od FPS-ova do MMORPG-ova što ukazuje na njegovu kompleksnost i mogućnosti koje nudi. Napisan je u programskom jeziku C++, a karakterizira ga mogućnost izrade videoigara za više platforma (eng. *cross-platform*). Neke od platforma koje su podržane od strane *Unreal Enginea* su Windows, macOS, Linux, iOS, Android, PlayStation 4 i 5, Xbox One, Nintento Switch, Oculus Rift i ostale VR platforme.

Unreal Engine 4, četvrti po redu, izašao je 2014. godine. Iako je pri izlasku alata bila potrebna licenca za izradu videoigara, danas ista ne postoji, već *Epic Games* koristi tzv. *royalty* model za komercijalnu upotrebu alata. Prva igra napravljena u *Unreal Engine 4* bila je *Daylight* za koju je tvrtka *Zombie Studios* dobila raniji pristup (eng. *early access*) samom alatu. Što se tiče izrade igara u *Unreal Engine 4*, konkretno znanje jezika C++ nije potrebno (ali je preporučljivo) zbog novog načina izrade skripti preko shematskih planova (eng. *blueprint*) gdje je moguće napraviti igru bez pisanja klasičnog koda. U ranijim verzijama *Unreal Enginea*, jezik programiranja bio je *UnrealScript* baziran na Javi, no zbog boljih performansa i jednostavnijeg otklanjanja pogrešaka (eng. *debugging*), C++ je prevladao. Neke od popularnijih igara koje pokreće *Unreal Engine 4* su *Fortnite* (koja je ujedno i najpopularnija *Epic Games*-ova igra), *Borderlands*, *Minecraft: Dungeons*, *Kingdom Hearts III* i druge.

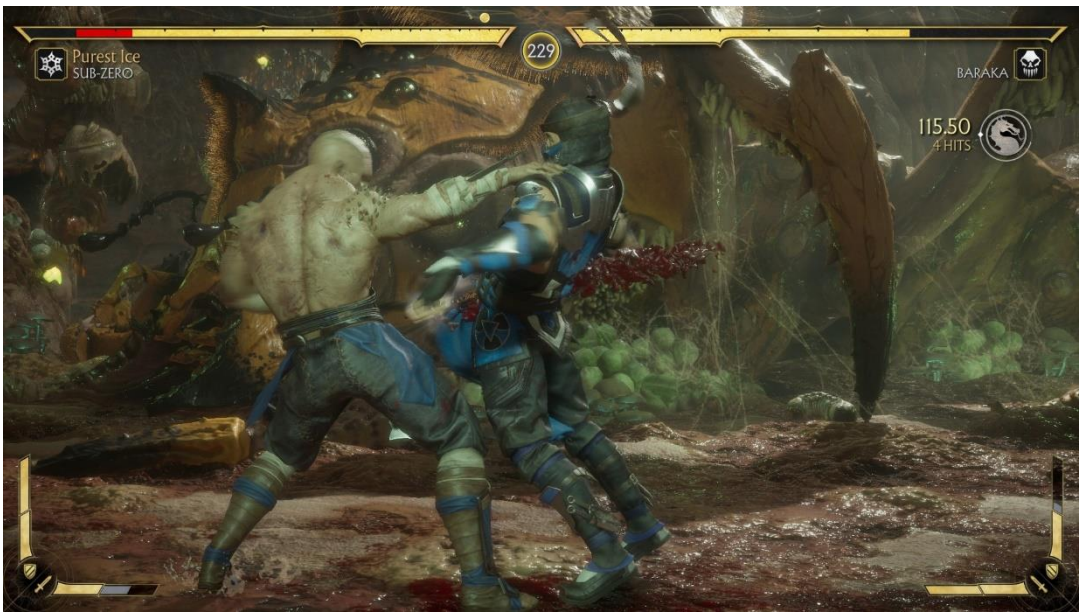
Unreal Engine 5 nasljednik je četvrte verzije alata, a očekivano vrijeme izlaska je kraj 2021. godine. Primarna namjena mu je izrada videoigara za konzole sljedeće generacije (koje doduše i *Unreal Engine 4* podržava). Do sada su prikazane dvije temeljne tehnologije koje *Unreal Engine 5* donosi, *Nanite* i *Lumen*, koje omogućuju ljepši i efikasniji prikaz detalja i osvjetljenja. [7]

Osim za izradu videoigara, kompanije poput *Disney*-a koriste *Unreal Engine* u filmskoj industriji. [6]



Slika 6: Izgled *Unreal Engine 4* korisničkog sučelja

Zanimljiva je činjenica da su neke igre poput *Mortal Kombat 11*, koja je izašla 2019. godine, napravljene u *Unreal Engine 3*. Doduše, radi se o prilagođenim varijantama alata, ali to je samo jedan od dokaza jačine *Unreal Engine*-a koja nije povezana samo uz zadnju verziju alata.



Slika 7: Mortal Kombat 11

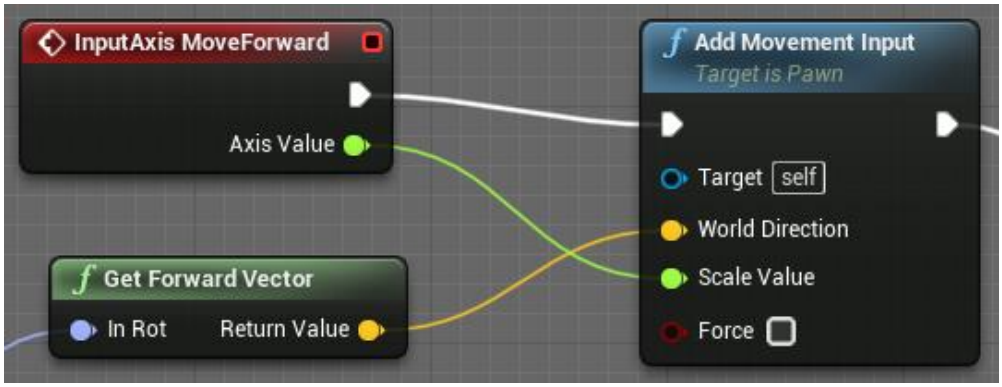
4. Izrada pucačine u prvom licu

Iako je već navedeno, cijela igra izrađena je u alatu *Unreal Engine 4* koristeći kombinaciju mnoštva *blueprint*-ova koji na kraju čine cjelinu pa ćemo redom (kako objašnjavamo dijelove igre) objašnjavati i same *blueprint*-ove te varijable, funkcije i sve ostalo što sadrže. Svi modeli koji se koriste u igri (ruke, oružja, neprijatelji) preuzeti su ili sa službenog *Unreal Marketplace*-a ili sa *Adobe*-ove stranice *Mixamo*.

Igra ispituje igračevu izdržljivost kojeg neprijatelji (zombiji) napadaju u valovima. Sama ideja slična je ideji nekada popularnih *Call of Duty 2 Zombie* servera gdje su nezaraženi igrači ubijali zaražene igrače i tako se spasili od infekcije koja bi u jednu ruku označavala kraj igre. U izrađenoj igri, važan je sistem bodovanja u kojem igrač bodove dobiva ubijanjem neprijatelja i pogađanjem neprijatelja u glavu (eng. *headshot*). Bodove igrač može trošiti na najbitniju stvar koja mu služi za opstanak – oružja. Naravno, kako igrač ne bi konstantno trošio bodove na oružja, na kraju same igre, prilikom igračeve smrti, igra ispiše trenutni broj bodova koji je pokazatelj igračevog iskustva u igri, gdje veći broj označava bolji zgoditak.

Blueprint-ovi u *Unreal Engine 4* način su izrade igara bez striktnog pisanja koda. Umjesto koda, sadrže tzv. čvorove (eng. *node*), sitne segmente od kojih svaki ima svoju ulogu. Većinskim dijelom, to su obične funkcije koje se koriste u programiranju gdje se piše kod, stoga se svaki *blueprint* može interpretirati u smislu kao da je napisan u kodu. Jedan čvor može predstavljati bilo već ugrađenu funkciju (npr. matematičke interpolacije) ili neku našu funkciju koju smo mi kreirali. Kako ćemo objašnjavati *blueprint*-ove, objašnjavati ćemo i čvorove koje smo koristili kako bi postigli željeni rezultat.

Prije samog početka, potrebno je objasniti vrste čvorova. Razlikujemo dvije vrste čvorova: izvršne i vrijednosne. Izvršni čvorovi, kao što im samo ime govori, se izvršavaju te mogu vraćati neku vrijednost prilikom izvršenja, dok vrijednosni čvorovi služe kao *getter*-i vrijednosti. Osim što ih možemo logički razlikovati, možemo ih razlikovati i vizualno prema gornjim strelicama koje sadrže samo izvršni čvorovi te se vežu linijama bijele boje. Međusobno spojeni čvorovi izvore se slijedno, a vrijednosti koje i jedni i drugi čvorovi imaju sa lijeve strane označavaju unos (eng. *input*), a s desne strane izlaz (eng. *output*).

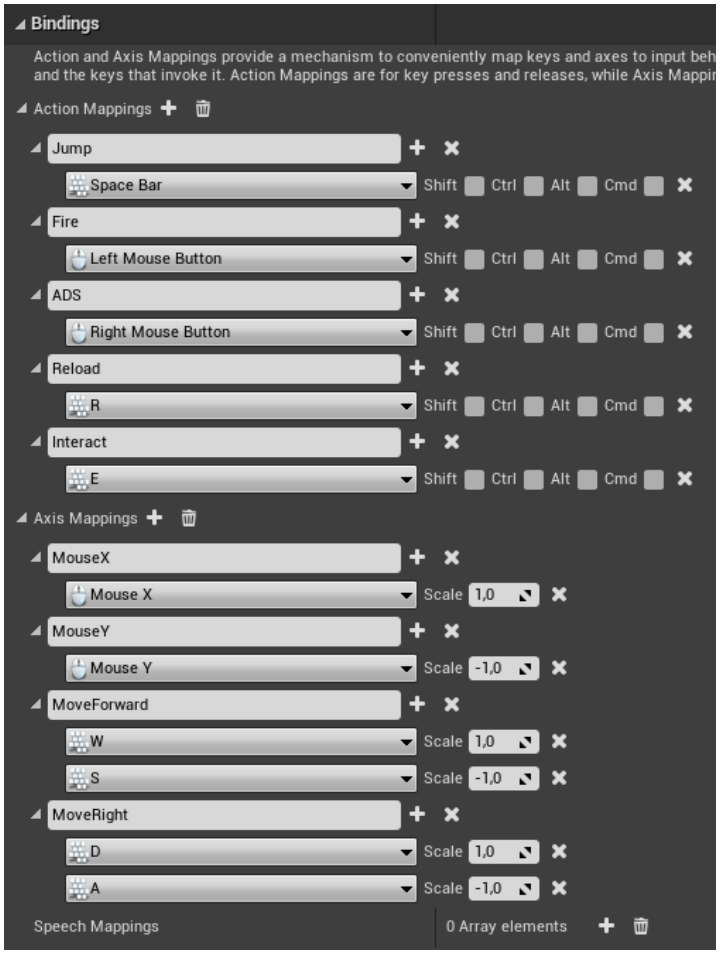


Slika 8: Izgled čvorova

4.1. Kontrole, kretanje i kamera

Da bismo uopće mogli upravljati našim likom, na samom početku potrebno je definirati kontrole koje igrač može koristit. Najjednostavnije rješenje navedenog problema je definiranje kontroli u postavkama projekta pod sekcijom *Input* čime dobivamo globalno definirane kontrole čije pritiske možemo osluškivati diljem cijelog projekta. S obzirom da radimo FPS igru, potrebno je definirati sljedeće:

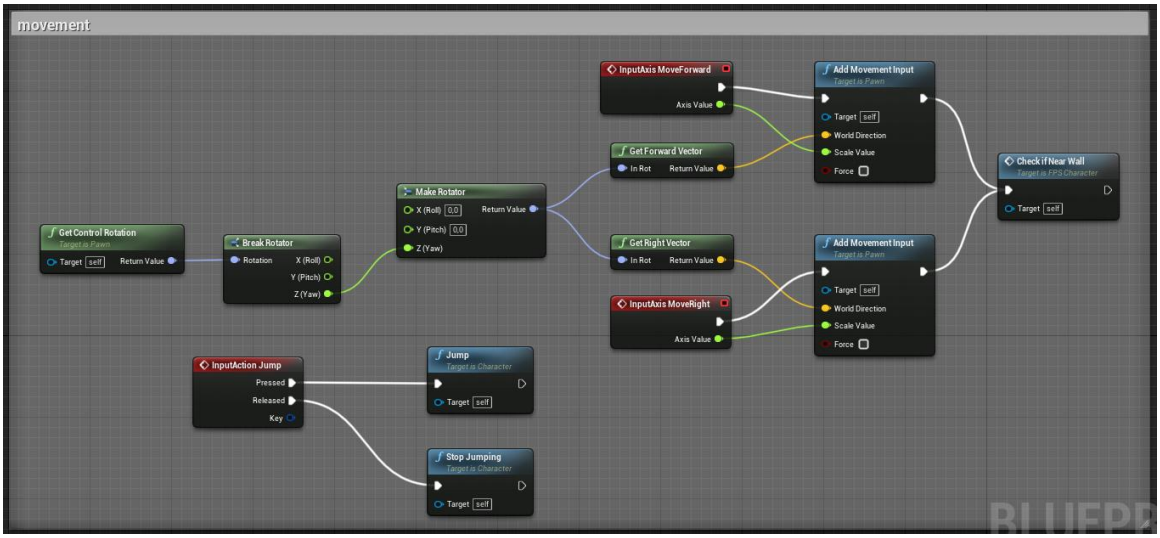
- kontrole kretanja (popularno korišteno WASD za kretanje te SPACE za skok) gdje je definirano samo kretanje unaprijed i udesno, pa kontrole kretanja unazad i ulijevo imaju negativan predznak ispred vrijednosti,
- kontrole kamere (također definicija kretanja po koordinatnim osima),
- kontrole usko vezane uz puške (pucanje i nišanjenje (eng. *aim down sights – ADS*)) gdje lijevi klik miša označava pucanje, a desni nišanjenje,
- te ostale kontrole (interakcija sa oružjem tj. kupnja oružja i ponovno punjenje oružja (eng. *reload*)).



Slika 9: Definiranje kontroli za upravljanje likom

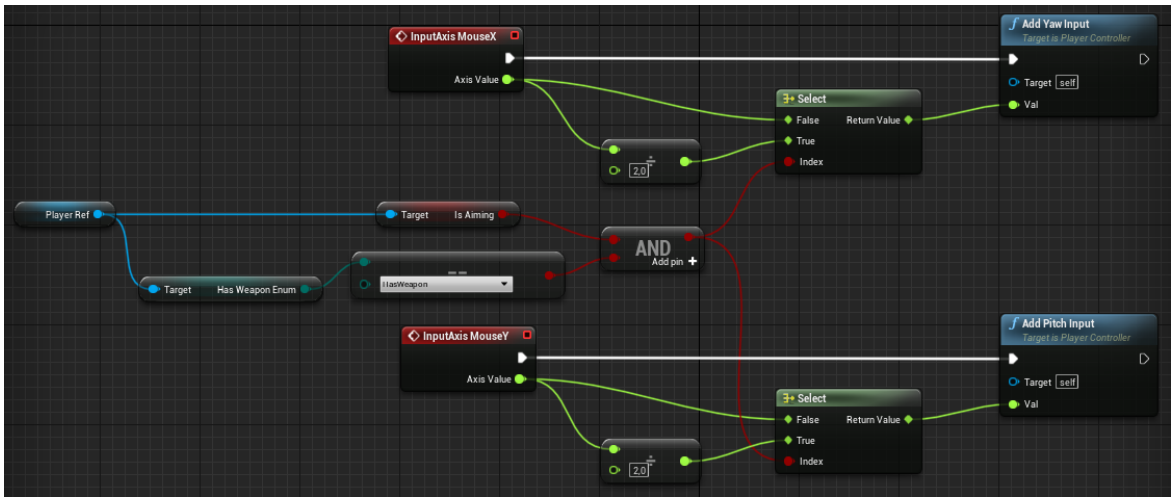
Kako su opisano samo definicije tipaka koje pokreću određenu kontrolu, praksa je nešto drugačija. *Unreal Engine* koristi glavne osi trodimenzionalnog prostora – X, Y i Z. Tri glavne koordinatne osi označavaju lokaciju objekta, dok rotacija sadrži drugačije oznake: *roll* koja je jednaka rotaciji X osi, *pitch* koja je jednaka rotaciji Y osi te *yaw* koja je jednaka rotaciji Z osi. Da bi postigli željeno kretanje, potrebno je saznati vektor kretanja našeg lika te na njega dodati vrijednost kontrole koju pritisnemo. Prvo saznajemo rotaciju našeg lika preko funkcije *GetControlRotation*, no s obzirom da ista sadrži rotaciju sve tri osi, a nama je potrebna samo rotacija Z osi, dobivenu rotaciju razlamamo (eng. *break*) koristeći čvor koji se koristi kako bi iz cijele strukture izvadili samo željeni tj. potrebni dio. Kako funkcije *GetForwardVector* i *GetRightVector* za unos traže rotaciju sve tri osi, potrebno je napraviti novu rotaciju koja je jednaka dobivenoj rotaciji Z osi. Nakon dobivenih vektora, prilikom pritiska tipke W ili S program dobije signal te se aktivira čvor po imenu *InputAxis MoveForward*, a obratno za A ili D se aktivira čvor *InputAxis MoveRight* koji na trenutni vektor kretanja dodaje novi ovisno o pritisnutoj tipki. Za kraj sekcije kretanja, dolazi naša funkcija *CheckIfNearWall* koja provjerava je li igrač blizu zida te ukoliko je, mogućnost pucanja je blokirana.. Kontrola za skok nešto je

jednostavnijeg oblika zato što koristimo ugrađene funkcije *Jump* i *StopJumping* koje same odrađuju potrebni posao.



Slika 10: Kretanje lika

Kontroliranje kamere slično je kontroliranju igrača gdje se za dobivenu vrijednost pomiče rotacija Y, odnosno Z osi. Vrijednost je duplo manja ukoliko igrač nišani sa oružjem o čemu ćemo pričati kasnije.



Slika 11: Upravljanje kamerom

4.2. Oružja

Oružja su najbitniji aspekt FPS igara, jer bez njih riječ pucačina u nazivu žanra gubi smisao. U izrađenoj igri postoji 6 oružja od kojih je svako specifično po svojim postavkama, pa razlikujemo:

- pištolj (eng. *pistol*) - klasično oružje svake FPS igre, puca brzinom pritiska klika na mišu,
- automatska puška (eng. *assault rifle*) - kao što i sam naziv govori, puca automatski brzinom koju odredimo u samoj varijabli puške,
- pumperica (eng. *shotgun*) – puška koja puca više metaka u isto vrijeme, uz sitno vrijeme odgode između pucnjeva,
- snajper (eng. *sniper rifle*) – koristi jače metke, ima mogućnost jakog zumiranja, te radi veliku štetu neprijateljima,
- bacač granata (eng. *grenade launcher*) – za razliku od prethodnih pušaka, radi štetu svima u okrugu granate koju baci,
- bacač raketa (eng. rocket launcher) – sličan *grenade launcheru*, ali ima veći doseg i radi veću štetu.

Osim što je svako oružje ima svoja svojstva, glavna podjela oružja svodi se na vrstu metaka koje ispaljuje. U svijetu FPS igara postoje dvije glavne vrste metaka, pa tako i u izrađenoj igri, a to su *hitscan* metci i projektili. Glavna razlika između njih je ta što *hitscan* metci prilikom pucanja odmah pogađaju ciljanu metu, dok projektili imaju svoju putanju i vrijeme leta. Prva tri oružja, pištolj, automatska puška i pumperica, ispaljuju metke *hitscan* vrste, dok druga tri oružja ispaljuju projektile.

S obzirom na to da sva oružja imaju slične i/ili iste varijable i funkcije koje koriste, za početak smo napravili *blueprint* roditeljsku klasu svim oružjima. U klasi oružja, po imenu *BP_WeaponBase* možemo pronaći sljedeće funkcije:

- *WeaponFire* – funkcija koja služi kao ručica za ispucavanje metka, poziva se pritiskom lijevog klika miša,
- *WeaponReload* – kao što smo naveli, *reload* znači ponovno punjenje oružja; sadrži samo potrebnu matematiku za oduzimanje metaka iz šanžera odnosno ukupnog broja metaka koje igrač ima,
- *HasAmmoInMag/HasExtraAmmo* – dvije *pure* funkcije, više služe kao *getter*-i nego što imaju klasičnu ulogu funkcija,
- *CalculateShot* – funkcija koja računa finalnu destinaciju metka,

- *AddDamage* – jednostavna funkcija koja na temelju zadnjeg metka dodaje štetu neprijatelju kojeg je metak pogodio.

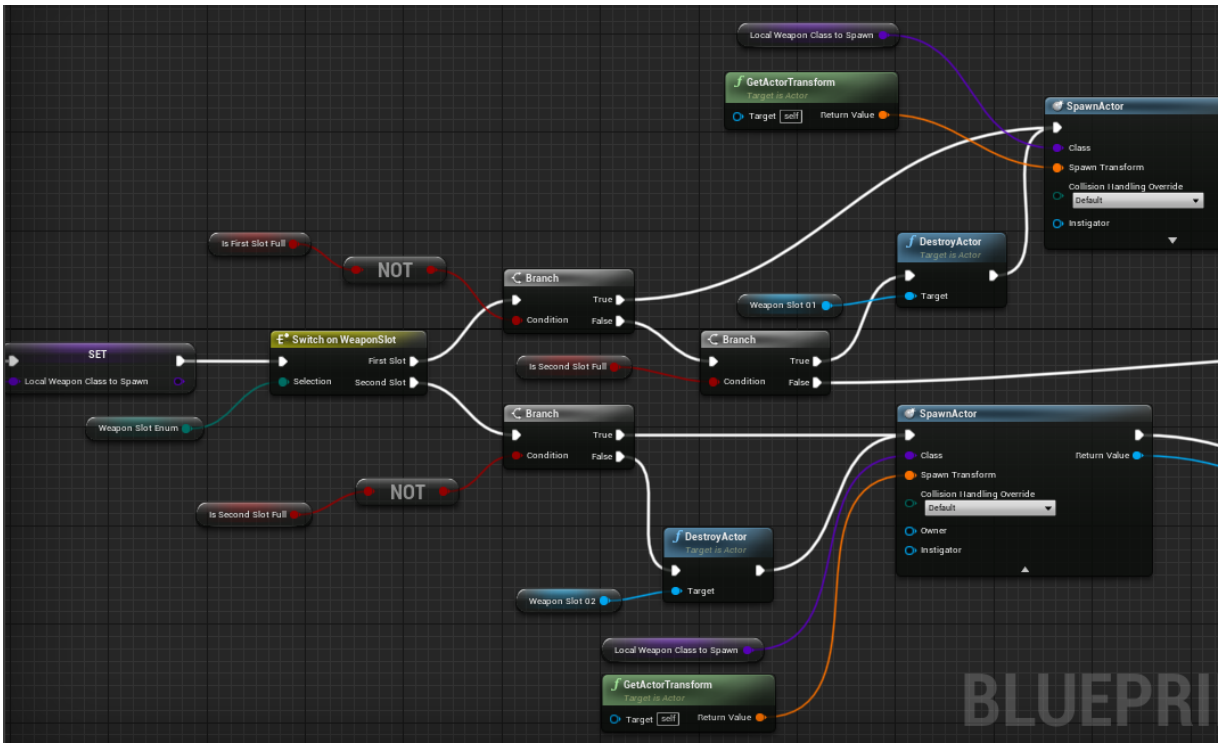
Uz funkcije, postoji i nekoliko varijabli koje su podijeljene po kategorijama za koje su vezane, a to su:

- *Ammo* – kategorija vezana uz broj metaka:
  - *CurrentAmmo (int)* – trenutni broj metaka koji se nalazi u šanžeru, smanjuje se za jedan nakon svakog ispaljenog metka,
  - *MaxMagAmmo (int)* – najveća količina metaka koju šanžer oružja može imati u jednom trenutku,
  - *CurrentTotalAmmo (int)* – trenutno ukupno stanje metaka koje igrač ima,
  - *MaxTotalAmmo (int)* – maksimalni broj metaka koje oružje može imati,
  - *MultipleBullets (bool)* i *BulletCount (int)* – funkcije koje koristi pumperica, *MultipleBullets* označava uvjet u grananjima, a *BulletCount* broj metaka koje pumperica ispaljuje.
- *Recoil* – kategorija vezana uz trzanje oružja prilikom pucanja:
  - *RecoilIntensity (float)* i *RecoilPitch (float)* – vrijednosti po kojoj se računa trzanje oružja, pištolj ima najmanji intenzitet, dok snajper i bacač raketa imaju najveći.
- *Properties* – kategorija koja sadrži generalne postavke svakog oružja:
  - *SocketName (name)* – ime priključka na koje se oružje spaja na ruku, na modelu ruku napravljeno je šest priključaka, za svako oružje po jedno. Služi za precizno stavljanje oružja u ruke s obzirom na to da se oružja razlikuju u veličini,
  - *WeaponType (enum)* – enumeracija koja sadrži ime oružja, korisno za grananja i provjere gdje je potreban točan naziv oružja za neke funkcije,
  - *AmmoData (struct)* – struktura od dvije varijable, najveća količina štete koju oružje radi i veličina kruga u kojem oružje radi štetu (za eksplozivne projektile),
  - *TrailFX (particle system)* – referenca na efekt koji oružje koristi, odnosi se na efekt koji je prisutan od pucnja do pogotka metka,
  - *ImpactEffectClass (class reference)* – referenca na klasu efekta pogotka metka. Svako oružje ima svoju klasu efekta pogotka u kojoj je definirana npr. veličina rupe koju metak ostavlja u zidu, zvuk pucnja i slično,
  - *IsAutomaticWeapon (bool)* – varijabla koja služi za provjere i grananja, odnosi se samo na automatsku pušku jer je ona jedina takvog tipa,

    - *FireType (enum)* – enumeracija koja sadrži tip metka kojeg oružje ispaljuje, može poprimiti vrijednosti *HitScan* ili *Projectile*.
- *Stats* – slično prošloj kategoriji, kategorija sadrži definicije svakog svojstva:
    - *ReloadTime (float)* – vrijeme u sekundama koje je potrebno da se oružje ponovno napuni,
    - *BulletSpread (float)* – varijabla koja definira preciznost oružja (samo u slučaju kad se metak ispaljuje, a sa oružjem se ne cilja),
    - *AutomaticFireRate (float)* – brzina kojom automatska puška puca,
    - *FireRate (float)* – brzina kojom bilo koja druga puška osim automatske može pucati,
    - *KillImpulse (float)* – jačina kojom metak udara, koristi se za simuliranje fizike prilikom ubijanja neprijatelja.
- *Projectile* – kategorija vezana uz projektile, ima funkciju samo kod oružja koji koriste projektile:
    - *Projectile (class reference)* – referenca na klasu projektila kojeg oružje ispaljuje, svako oružje sa projektilima ima odgovarajuću klasu projektila.
- *ADS* – kategorija koja sadrži varijable potrebne za funkcionalno nišanjenje oružjem:
    - *IsAiming (bool)* – varijabla koja služi samo određene provjere i grananja gdje je potrebno razlikovati nišanjenje od normalnog pucanja (poput ispaljivanja metka),
    - *AimFOV (float)* – vidno polje prilikom nišanjena izraženo u brojevima, snajper ima najuže što znači da najviše približava,
    - *XOffset, YOffset, ZOffset (float)* – varijable koje služe kako bi centrirali oružje i pomakli mu mjesto priključka na kojem se nalazi.
- *UMG* – kategorija sa jednom varijablom koja označava ikonicu koja se prikazuje na sučelju.

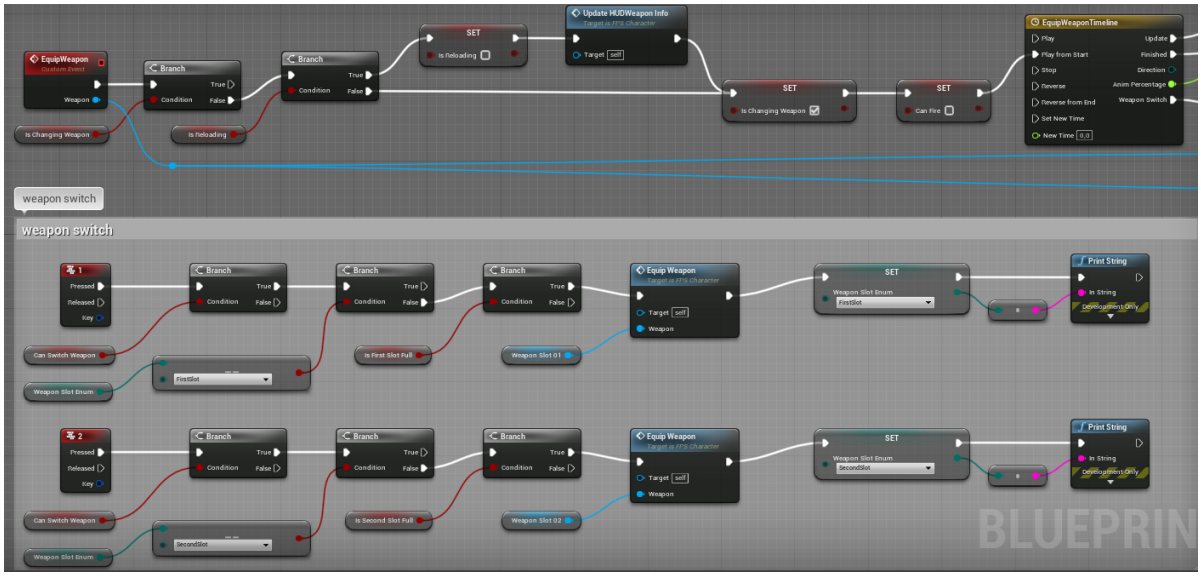
Navedene funkcije i varijable pokazuju nam složenost oružja u FPS igrama i na koliko stvari je potrebno obratiti pozornost. No, prije samog korištenja oružja, potrebno je ista uzeti. Sistem kupovine oružja zamišljen je u kombinaciji sa bodovima. Svih šest oružja se na mapi nalaze na dva mjesta, u obliku rotirajućeg modela oružja na zemlji. Svako oružje ima svoje ime i cijenu, a da bi isto bilo moguće kupiti, potrebno je približiti se rotirajućem modelu i pritisnuti ranije definiranu tipku E. Igrač u svakom trenutku ima dva utora u koje može staviti oružje, a utorima može pristupiti koristeći tipke 1 i 2. Ukoliko su oba prazna (samo na početku igre), prioritet ima prvi utor, no ukoliko je prvi utor pun, a drugi prazan, kupljeno oružje svrstava se u drugi utor. Ukoliko su oba utora puna, novo kupljeno oružje zamijenit će trenutno aktivni utor.

Slika 12: Izgled oružja za kupovinu



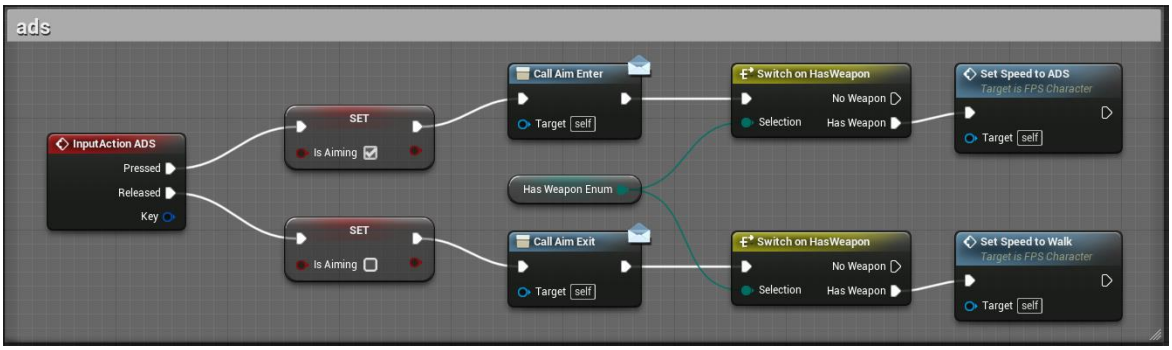
Slika 13: Algoritam izmjene oružja na utorima

Animacija promjene oružja između prvog i drugog utora zapravo i nije animacija već simulacija animacije. Naime, u trenutku pritiska gumba za promjenu utora, rotacija X osi promijeni se za 45 stupnjeva, što je dovoljno da ne vidimo ruke i oružje koje se nalazi u rukama. Trajanje „animacije“ riješeno je uz pomoć vremenske crte (eng. *timeline*) koja ima trajanje od pola sekunde.

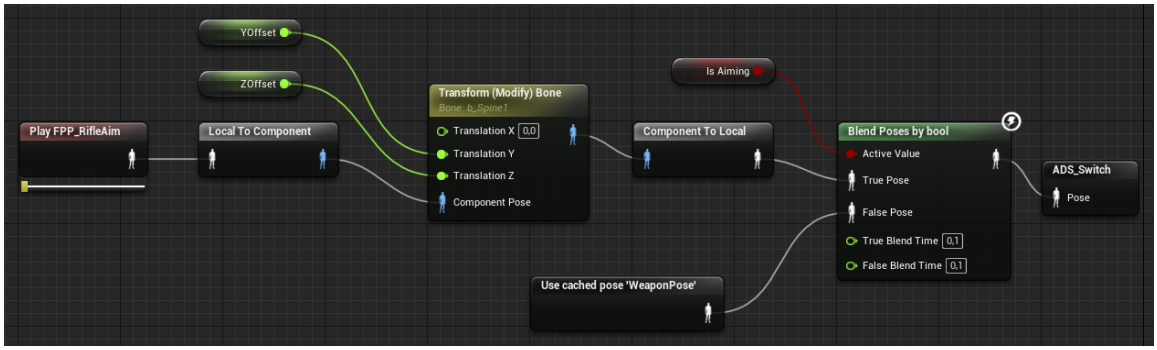


Slika 14: Algoritam promjene oružja

Nakon odabira oružja koje nam se čini privlačnim, slijedi nišanjenje i pucanje. Kod definicije kontrola, odabrali smo desni klik miša za nišanjenje i lijevi za pucanje. Nišanjenje kod FPS igara donosi nekoliko prednosti, a najveća od njih je preciznost puške koja je u našoj igri sto postotna ukoliko igrač nišani oružjem. Iako je trzanje oružja postojano prilikom nišanjena, ono je nešto manje izraženo. Samo akcija nišanjenja je zapravo transformacija kostiju ruku na način da se oružje stavlja na sredinu ekrana, i u visinu gdje pogled prolazi kroz mjesto namijenjeno nišanjenju. Osim transformacije ruku, smanjuje se i vidno polje koje daje efekt približavanja slike na ekranu.



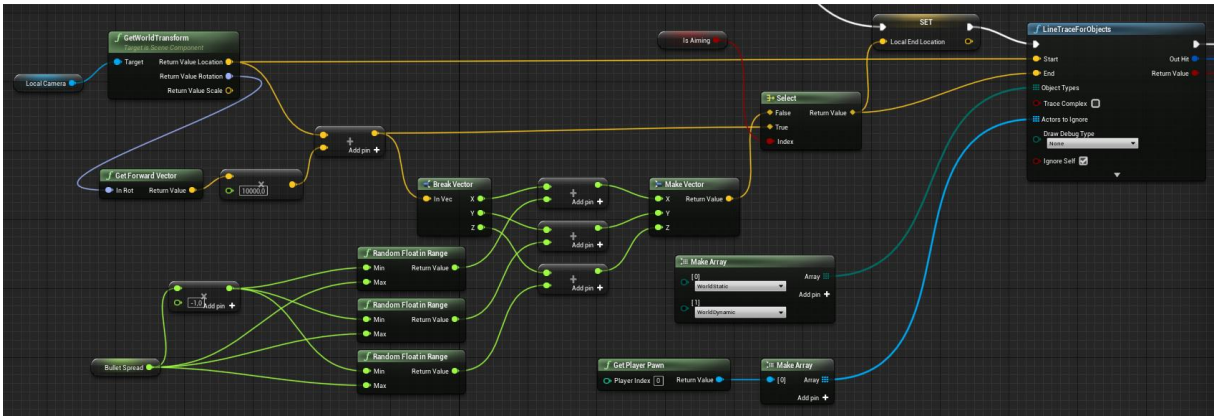
Slika 15: Osluškivanje na desni klik miša



Slika 16: Transformiranje ruke za potrebe nišanjenja

Nakon što smo naciljali metu, možemo sa uspjehom pucati. Kao što je već navedeno, postoji nekoliko vrsta oružja. Automatska puška se prema načinu pucanja izdvaja od ostalih jer ima mogućnost pucanja bez konstantnog pritiskanja lijevog klika miša. Slušatelji kontrola slušaju na dvije stavke, pritisak kontrole i otpuštanje kontrole. Zahvaljujući tome, kada otpustimo lijevi klik miša, automatska puška će prestati pucati jer radi na način petlje koja se izvršava sve dok je tipka pritisnuta i u oružju ima metaka. Brzinu pucanja kontroliramo kroz varijable koje smo ranije naveli. Da ostala oružja nemaju prilagođenu zaštitu, metke bi bilo moguće ispaljivati onom brzinom kojom igrač može pritiskati lijevi klik miša, što u stvarnosti nije baš moguće (osim pištolja za kojeg ta zaštita ne vrijedi). Nakon pritiska lijevog klika miša, metak se ispaljuje te se stvara efekt na kraju cijevi oružja koji simulira ispaljeni metak. Ovisno o vrsti metka (*hitscan* ili projektil) generira se novi efekt ili se poziva i stvara nova klasa koja predstavlja ispaljeni projektil. Kako projektil ima svoju putanju i vrijeme koje mu je potrebno da dođe do završetka, on je nakon pucnja jasno vidljiv u igri te se njegov hod može pratiti, dok se kod *hitscan* vrste metaka hod ne vidi, no njegovu završnu točku moguće je vidjeti jer svako oružje ostavlja rupu u zidu, ukoliko je zid (ili bilo koji drugi materijal osim neprijatelja) njegova završna točka. Tako se svaki ispaljeni metak sa svojom složenom strukturom sprema u strukturu pod nazivom *Hit Result* iz koje možemo izvaditi sve potrebne informacije za generiranje zvuka i efekata.

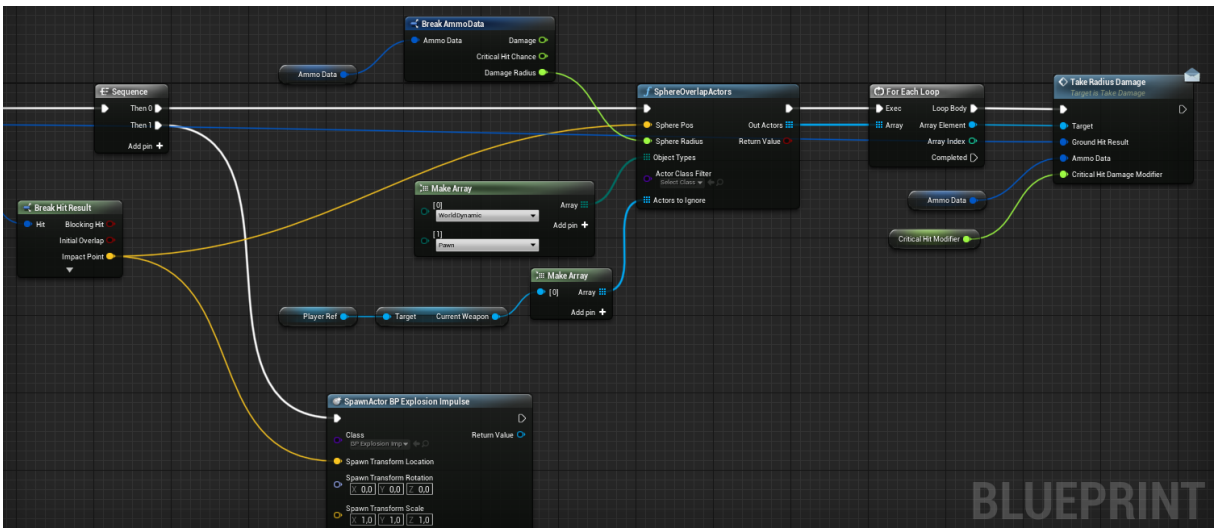
Iako funkcija *CalculateShot* prima iste argumente i za *hitscan* metke i za projektile, funkcija gdje se *CalculateShot* poziva odrađuje selekciju vrste metka, tako da ista nije potrebna i u samom računanju. Funkcija *CalculateShot* računa gdje će metak nakon ispaljivanja završiti te ugrađena funkcija *LineTraceForObjects* služi kako bi pronašli objekt kojeg je metak pogodio. Funkciji *LineTraceForObjects* šaljemo početnu lokaciju (lokaciju našeg lika) i izračunati smjer kretanja metka, vrste kolizije objekata na koje se aktivira i objekte tj. *actore* koje metak ignorira, a vraća nam ranije spomenuti *Hit Result* i *bool* vrijednost koja označava dogodila li se kolizija metka i nekog objekta ili ne. U samo računanje završne lokacije, uzeli smo u obzir preciznosti puške (*bullet spread*) koji ne postoji ukoliko sa puškom nišanimo.



Slika 17: Izračun smjera kretanja ispaljenog metka

Nakon što je metak ispaljen, igra stvara njegov objekt te efekt koji se mijenja ovisno o vrsti materijala koju pogodimo. Na samom kraju, igra dodaje štetu neprijateljima ukoliko su oni završna točka metka ili su u okrugu eksplozije koju projektil radi.

Eksplozija koju projektili naprave je simulacija nevidljive sfere, koja svakom neprijatelju s kojim prepozna koliziju napravi štetu. Količina štete ovisi o udaljenosti neprijatelja od samog centra eksplozije, a veličina sfere o svojstvu projektila koje oružje baca, koje je sadržano u strukturu *AmmoData*. Osim sfere koja čini štetu, generira se i sila koja simulira fiziku te odbacuje sve neprijatelje ovisno o udaljenosti od eksplozije.



Slika 18: Generiranje sfere i sile eksplozije

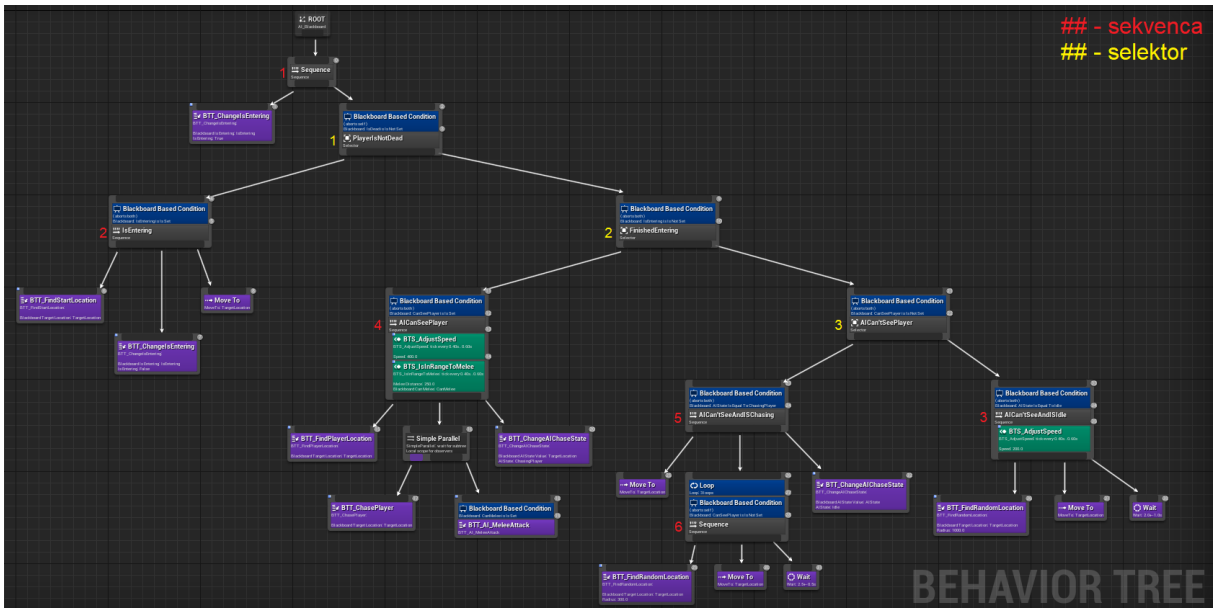
4.3. Neprijatelji (AI)

Glavni negativci u igri su zombiji, koji na valove dolaze i pokušavaju ubiti igrača svojim napadom iz blizine. Izmjene neprijateljeve animacije hodanja i trčanja odrađene su preko jednostavnog jednodimenzionalnog *blend space*-a, opcije *Unreal Engine*-a koja na temelju varijable tranzitira iz jedne animacije u drugu. Varijabla koja određuje animaciju je brzina hoda koju namještamo preko drva ponašanja AI-a o čemu ćemo malo kasnije pričati. Maksimalna brzina kretanja zombija iznosi 550, gdje nula označava mirno stanje u kojem neprijatelj stoji na mjestu, a 550 trčanje sa najvećom brzinom. Kada neprijatelj ima namještenu brzinu od 200, glavna animacija mu je animacija hodanja, a na brzini od 400 počinje trčati. Kada bi neprijatelj imao brzinu kretanja od 300 (vrijednost između hodanja i trčanja), tada bi se izvodila djelomična animacija hoda i djelomična animacija trčanja, što čini *blend space* super alatom za tranzicijske animacije. Osim animacije hoda i trčanja, neprijatelj također ima animaciju napada kojim želi oštetiti igrača.

Najbitniji dio funkcionalnosti neprijatelja je njihova umjetna inteligencija. Cijela umjetna inteligencija napravljena je koristeći drvo ponašanja (eng. *Behavior Tree*), aspekta *Unreal Engine 4* koji služi isključivo za AI. Drvo ponašanja ima i tzv. crnu ploču (eng. *Blackboard*), a ona služi za upravljanje varijablama koje koristi drvo. U crnoj ploči možemo razlikovati sljedeće varijable:

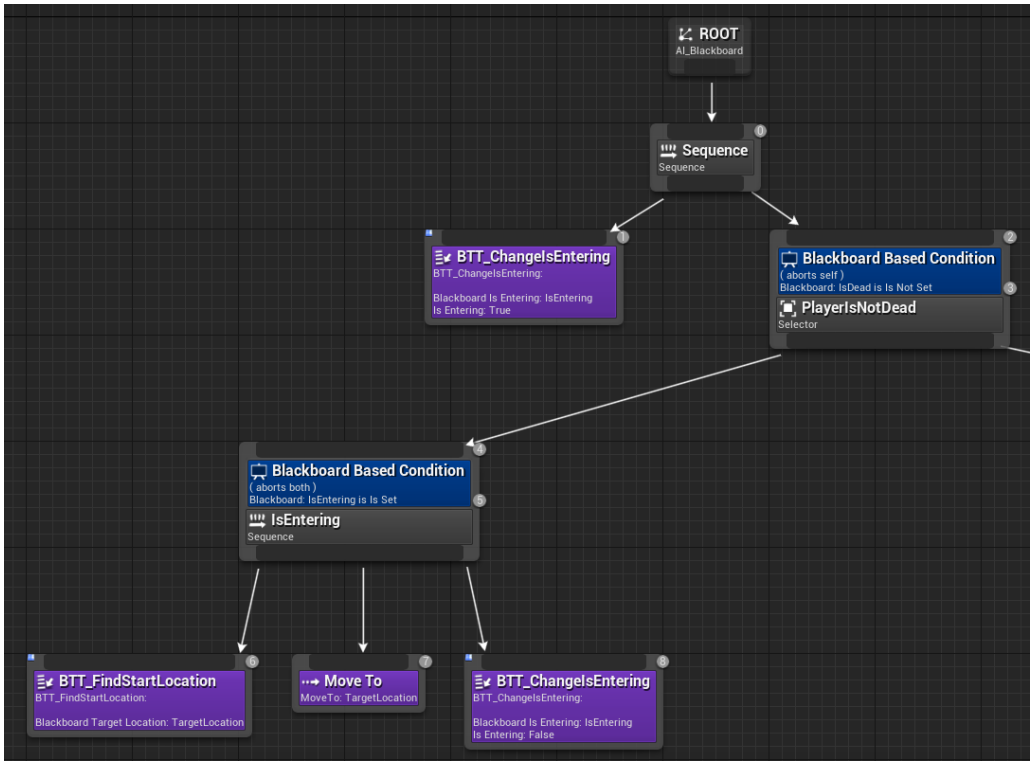
- *TargetLocation (vector)* – lokacija u obliku vektora do koje neprijatelj ima naredbu doći, mijenja se ovisno o stanju u drvu ponašanja,
- *CanSeePlayer (bool) – boolean* tip varijable koji služi kao provjera za grananja gdje je to potrebno, kada neprijatelj vidi igrača ona je postavljena na *true*, a u suprotnom njena vrijednost je *false*,
- *IsDead (bool)* – varijabla koja poprima dvije vrijednosti, ovisno o neprijateljevom životnom stanju,
- *AIState (enum)* – enumeracija koja ima dva stanja, Idle i ChasingPlayer koja označuju lovi li neprijatelj igrača ili ne,
- *CanMelee (bool)* – vrijednost koja služi kao provjera je li neprijatelj dovoljno blizu igrača da ga može napasti,
- *IsEntering (bool)* – kako se igra na valove, a neprijatelji se stvaraju izvan igrivog dijela mape, nakon njihovog stvaranja potrebno je da uđu u igrivi dio mape. Zato je varijabla *IsEntering* tu kako bi osigurala pravilno grananje u drvu ponašanja neprijatelja te se postavlja na *false* jednom kada neprijatelj uđe na svoju destinaciju.

Za početak, drvo ponašanja može imati selektore, sekvence i paralele kao načine grananja i slijeda. Osim navedenih stvari, drvo ponašanja također sadrži i određene zadatke, servise i dekoratore koji pospješuju ponašanje neprijatelja. Uz već napravljene zadatke i servise, mi možemo kreirati svoje po želji i pridodavati ih selektorima i sekvencama. Na kraju krajeva, drvo ponašanja neprijatelja izgleda ovako:



Slika 19: Drvo ponašanja neprijatelja

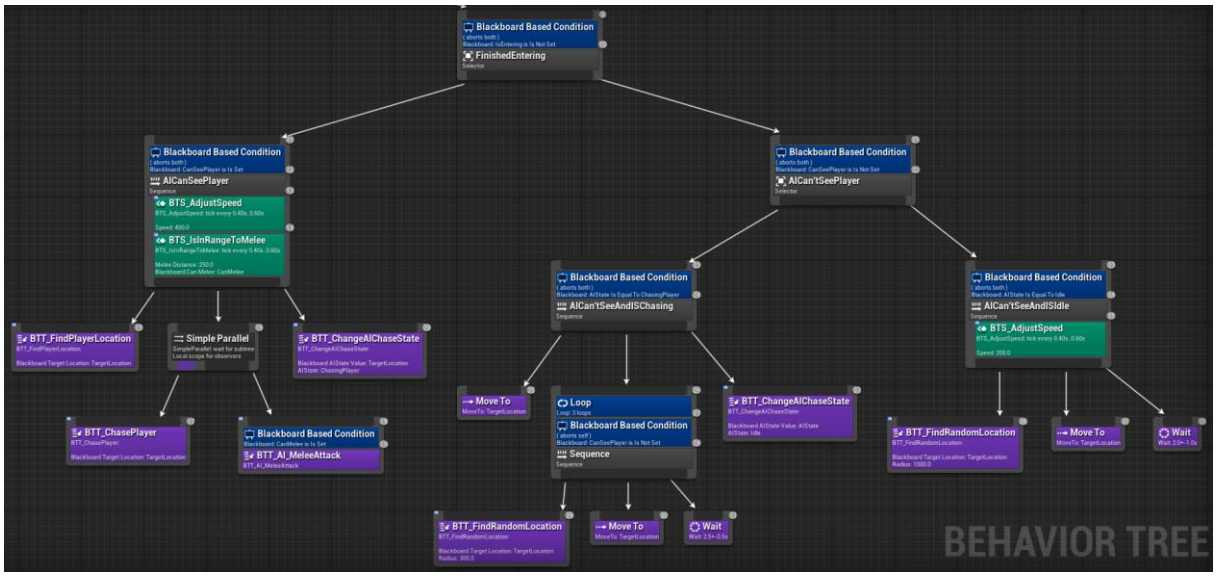
Iako teško vidljivo sa slike, objasnit ćemo redom definirano ponašanje neprijatelja. Na samom početku (*root*-u) nalazi se sekvenca (#1) koja za početak *IsEntering* varijablu postavlja na vrijednost *true* (čisto radi preventive i otklanjanja određenih grešaka). Nakon obavljenog postavljanja vrijednosti, drvo dolazi do prvog uvjetovanog čvora, koje nastavlja sa radom samo ukoliko je neprijatelj živ. U samom čvoru, nastavak tj. grananje određuje selektor (#1) koji ovisno o tome je li neprijatelj ušao u igrivi dio mape ili ne izvršava sljedeće zadatke. Ukoliko je vrijednost varijable *IsEntering* na crnoj ploči postavljena na *true*, drvo sa radom nastavlja sa lijeve strane te dolazi do druge sekvence (#2) u kojoj prvo postavlja vrijednost vektora *TargetLocation* na lokaciju unutar mape, nakon čega se neprijatelj kreće do označene lokacije i na samom kraju, sekvenca mijenja vrijednost varijable *IsEntering* u false.



Slika 20: Lijeva putanja drva ponašanja

Spomenuti selektor (#1) koji bira putanju ovisno o vrijednosti varijable *IsEntering*, veže se na dva uvjetovana čvora koji, ako varijabla *IsEntering* promijeni vrijednost, pucaju i vraćaju se ponovno na selektor gdje on mijenja svoju putanju. Ukoliko drvo poslije prvog selektora (#1) nastavi na desnu stranu, dolazi do drugog selektora (#2) koji bira putanju ovisno o tome može li neprijatelj vidjeti igrača u svojem vidnom polju, gdje lijeva strana označava *true* vrijednost, a desna *false* vrijednost. Oba uvjetovana čvora rade na istom principu kao i prvi, ukoliko vrijednost *CanSeePlayer* promijeni vrijednost, putanja ide korak u nazad te ponovno bira putanju. Putanja koju drvo odabere nakon dolaska neprijatelja unutar mape je desna, koja se aktivira kada neprijatelj ne vidi igrača. Nakon odlaska ka desnoj strani, drvo ponovno nailazi na selektor (#3) u kojem ispituje enumeraciju stanja neprijatelja. Ukoliko neprijatelj nije u stanju lovljenja igrača, tada se njegova brzina namješta na 200 (što označava animaciju hodanja), te algoritam pronalazi nasumičnu lokaciju unutar 1000 jedinica do koje neprijatelj hoda, pričeka od jedne do tri sekunde te ponovi postupak. Ako neprijatelj u međuvremenu uoči igrača, putanja se vraća do drugog selektora (#2) te odlazi na lijevu stranu nakon čega dolazi sekvenca (#4) i neprijateljeva brzina poprima vrijednost od 400, što je ekvivalent animaciji trčanja. Tada neprijatelj dobiva uputu o igračevoj lokaciji, nakon koje slijedi paralela koju najjednostavnije možemo opisati kao „dok radiš a, radi i b zadatak“. U paraleli neprijatelj lovi igrača i kada mu se dovoljno približi, napada ga. Na kraju sekvence (#4), postavlja se vrijednost enumeracije stanja neprijatelja u *ChasingPlayer* koja služi za grananja na suprotnoj strani selektora (#3). Kada igrač ode izvan vidnog polja neprijatelja, putanja drveta se vraća

na drugi selektor (#2) te opet nailazi na selektor (#3) vezan uz numeraciju neprijateljeva stanja. Ovaj puta, enumeracija je postavljena na vrijednost *ChasingPlayer* stoga slijedi izvršavanje pete sekvence (#5) u kojoj neprijatelj dolazi do posljednje poznate lokacije igrača, for petljom tri puta pronalazi nasumičnu lokaciju blizu posljednje poznate lokacije igrača, dođe do nje i pričeka jednu do dvije sekunde. Ukoliko u tom periodu neprijatelj ne ugleda igrača, stanje enumeracije postavlja se na *Idle* i igrač se vraća u početno stanje – nasumično traženje lokacije unutar 1000 jedinica.



Slika 21: Desna putanja drva ponašanja

Osim vida, neprijatelj ima još dvije percepcije, sluh i percepcija na primanje štete. Svaki puta kada igrač ispali metak iz oružja, neprijatelj dolazi do lokacije odakle zvuk dolazi te obavlja dio od pete sekvence (#5) gdje tri puta od zadnje poznate lokacije pucnja traži blisku lokaciju do koje trči. Treća percepcija, tj. treće neprijateljevo čulo je čulo na primanje štete. Kada neprijatelj primi štetu, automatski saznaje igračevu lokaciju, kreće prema njoj, a drvo ponašanja postavlja se na dio četvrte sekvence (#4).

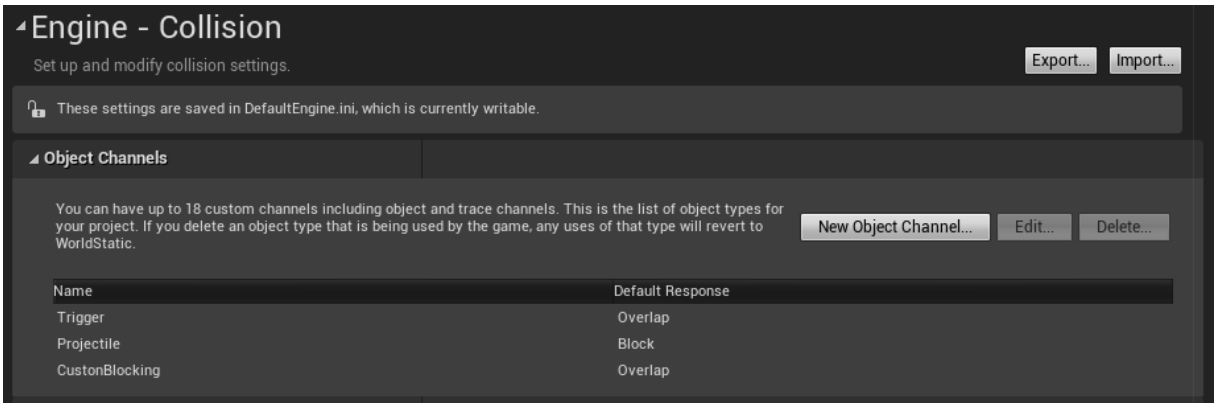
Važno je napomenuti kako se neprijatelji mogu kretati samo po posebnom sloju, zvanom *NavMeshBoundsVolume*, koji se namješta manualno, a u našem slučaju je namješten da pokrije cijeli igrivi dio mape.

Slika 22: Granice *NavMeshBoundsVolume*-a

4.4. Kolizija i sistem valova neprijatelja

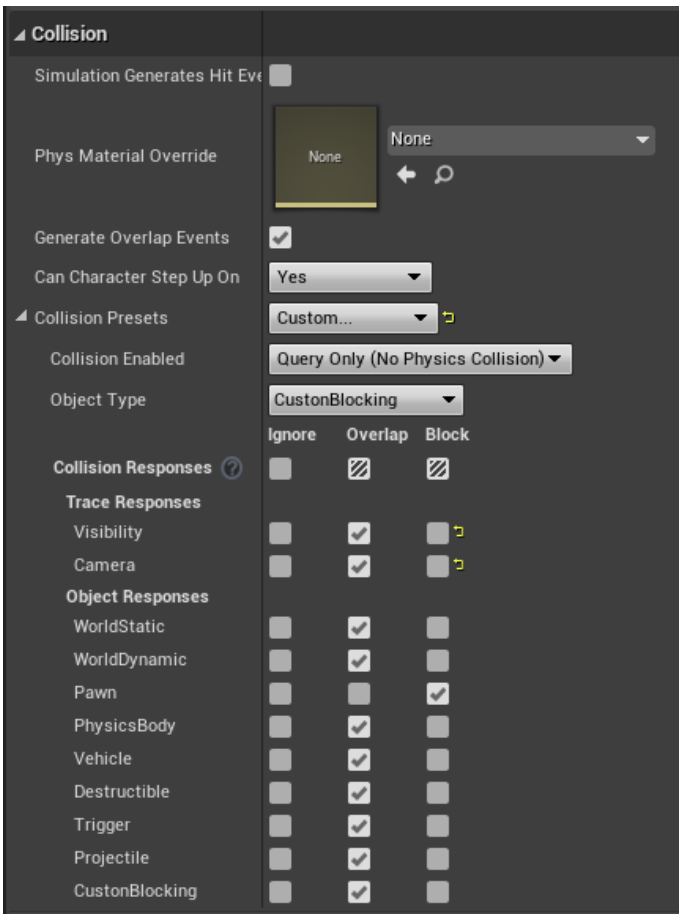
4.4.1. Kolizija između objekata

Kolizija u *Unreal Engine*-u vrlo je jednostavnog karaktera. Svaki objekt, *actor*, zapravo sve vidljivo u igri ima svoje postavke kolizije. Naravno, *Unreal Engine* dolazi sa već unaprijed postavljenim postavkama, no ukoliko nam one ne odgovaraju, uvijek možemo napraviti vlastite. Svaki vlastoručno kreirani kanal kolizije ima zadane postavke koje određuju njegovo ponašanje sa ostalim kanalima, a to može biti ignoriranje, preklapanje ili blokiranje.



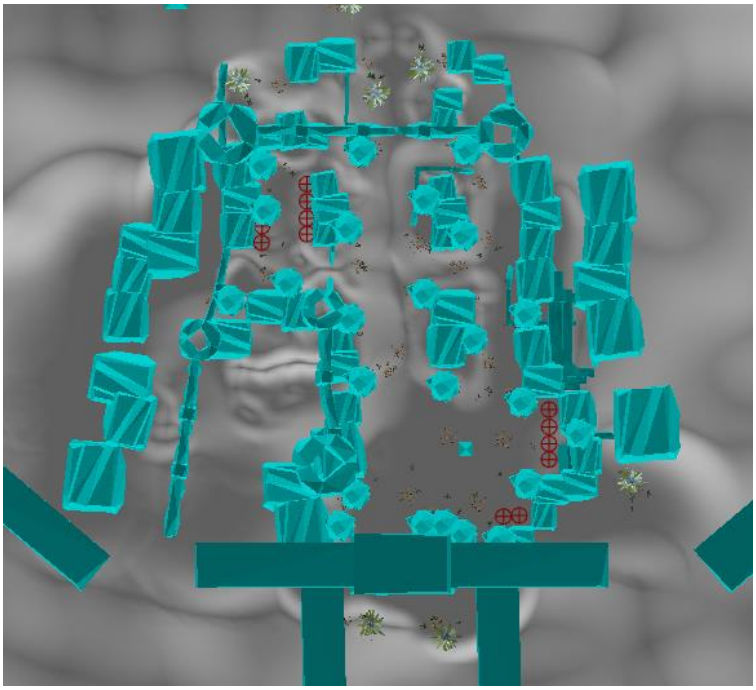
Slika 23: Vlastoručno napravljeni kanali kolizija

Na donjoj slici možemo vidjeti kako izgledaju postavke kolizije nekog objekta. Na slici se radi o nevidljivoj kutiji kroz koju igrač ne može prolaziti, a neprijatelji mogu samo jednom i to kada prvi puta uđu u njene granice kolizije. Nakon izlaska iz granica kolizije, neprijateljima za navedenu kutiju vrijedi ista kolizija kao i igraču, što bi značilo da njima kutija predstavlja *one-way* koliziju za ulaz u igrivi dio mape. Opisana nevidljiva kutija nalazi se na izlazima iz zidina pustinjskog grada te osigurava da igrač ostane isključivo u dijelu mape u kojem je igra zamišljena. Kao što se vidi i na donjoj slici, svaki objekt u igri ima namještene svoje postavke kolizije. Vrstu objekta biramo pod postavkom *Object Type*, a njegov odnos sa ostalim objektima (bilo sa likom kojeg upravljamo, neprijateljima ili nekih drugim modelom) odabiremo označavanjem kućica pored imena kanala kolizije.



Slika 24: Postavke kolizije objekta u igri

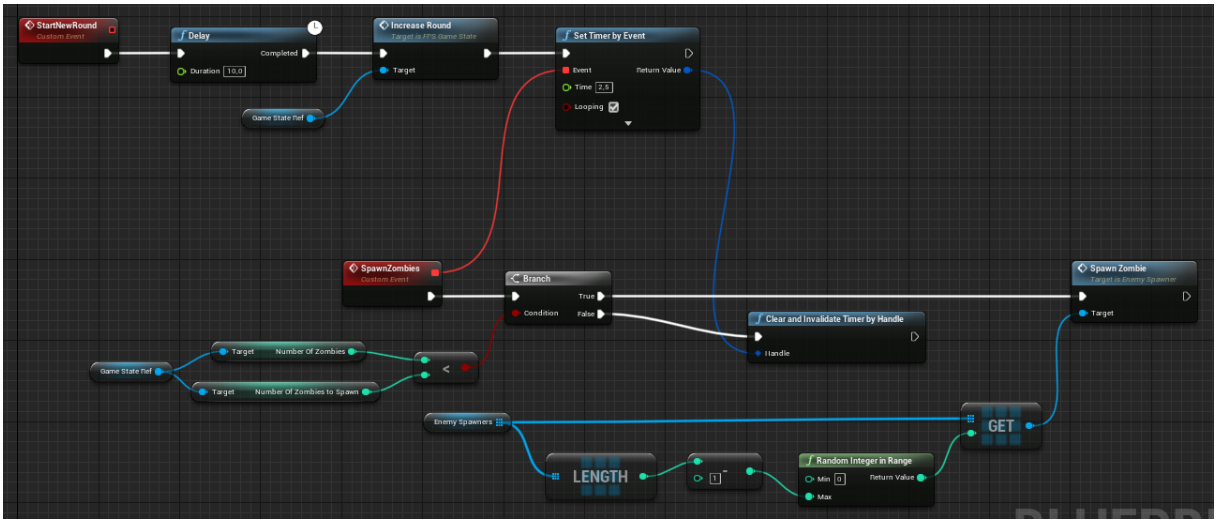
S obzirom da se na mapi nalazi puno kućica i zidina, takvi objekti najčešće imaju zadanu postavku kolizije koja blokira sve ostale kanale, uključujući i metke i projektile.



Slika 25: Prikaz igračeve kolizije na mapi

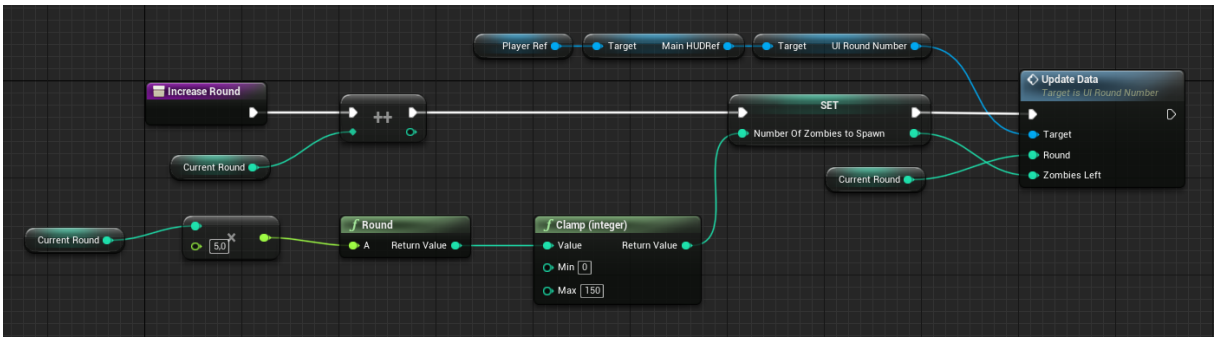
4.4.2. Sistem valova neprijatelja

U samom uvodu u igru stoji napisano kako igra testira igračevu izdržljivost stvarajući neprijatelje na valove. Svakim valom neprijatelja je sve više, pa sama igra sa svakim sljedećim valom postaje sve teža. Između svake runde igrač ima 10 sekundi slobode kako bi lakše kupio oružja i pripremio se na sljedeći val neprijatelja.



Slika 26: Algoritam za početak runde i stvaranje neprijatelja

Na mapi se nalaze četiri objekta koji stvaraju neprijatelje – na svakom izlazu sa svake strane po jedan. Kako bi stvaranje neprijatelja bilo nepredvidljivo, algoritam bira jednog od četiri stvaratelja i tako za svakog neprijatelja koji je preostao za stvaranje, što znači da ako je ostalo četiri neprijatelja za stvoriti, svakog bi neprijatelja mogao stvoriti drugi objekt. Neprijatelji se stvaraju u vremenskom intervalu od dvije i pol sekunde jedan od drugog, a provjera je li runda gotova i treba li nova krenuti odvija se nakon svakog ubijenog neprijatelja. Osim što je svakim valom sve više neprijatelja, oni postaju i sve jači (teže ih je za ubiti i rade više štete) i sve veći.



Slika 27: Algoritam za povećavanje broja runde i broja neprijatelja

4.5. Korisničko sučelje

Korisničko sučelje (eng. *User Interface – UI*) sastoji se od nekoliko elemenata koji su povezani u cjelinu. Kada pričamo o korisničkom sučelju, bitno je da ono nije prenatrpano, a opet da pokazuje sve bitne informacije. U igri možemo kategorizirati sve elemente sučelja na tri glavne kategorije:

- UI vezan uz oružja, a sastoji se od:
  - dinamičnog nišana koji se povećava prilikom hodanja i/ili skakanja odnosno padanja, prikazuje odstupanja metka od samog središta. Nije prikazan kada igrač nišani jer tada odstupanja metaka ni nema,
  - statistike oružja koje se sastoje od ikonice oružja kojeg trenutno držimo, broja metaka u šanžeru te ukupnog broja metaka koje igrač posjeduje. Uz navedene stvari, sadrži i tekst koji služi kao indikator kada ponovno punimo oružje,
  - nišana za snajper, koji je u suštini slika u PNG formatu koja se prikazuje kada nišanimo sa snajperom,

- indikatora štete koju napravimo neprijateljima. Stvara se iznad dinamičnog nišana i prikazuje točnu štetu koju napravimo, a kada se radi o pucnju koji ubija neprijatelja, tekst je ispisan crvenom bojom,
- indikatora pogotka koji za razliku od indikatora štete prikazuje samo pogodak neprijatelja. Mijenja boju u crvenu kada pucanj ubija neprijatelja.

- UI vezan uz lika, sadrži sljedeće elemente:
  - indikator igračevog zdravlja u obliku trake za napredak, treperi crveno kada igrač ima manje od 25% zdravlja,
  - indikator igračevog zdravlja u obliku krvi na ekranu, pojavljuje se kada igrač ima manje od 50% zdravlja,
  - indikator smjera štete koji je prikazan svaki puta kada igrač primi štetu.
- UI vezan uz informacije o igri, čine ga:
  - informacije o trenutnoj rundi i broju živih neprijatelja koje treba ubiti kako bi započela nova runda,
  - broj trenutnih bodova koje igrač ima,
  - informacije o oružju za kupovinu,
  - statistika cijele igre, pokazuje se kada igrač umre.

Slika 28: Prikaz elemenata korisničkog sučelja

5. Zaključak

Iako je ovo bilo moje prvo iskustvo sa izradom 3D igre, smatram da je krajnji produkt zadovoljavajuće razine. Tijekom izrade igre, štošta se dalo naučiti, no ono što je ostavilo najveći utjecaj na mene je kompleksnost 3D FPS igara. Sljedeći korak u cijeloj izradi bilo bi dodavanje komponente igranja za više igrača, no to ćemo ostaviti za drugi put.

Osim što je ovo bilo moje prvo iskustvo izrade 3D igre, također sam se i prvi puta susreo sa jednim ovako velikim alatom za izradu videoigara, koji je bez dvojbe trenutno najjači (uz *Unity*) u industriji videoigara. Samo njegovo djelovanje koje se provodi od 1998. godine i četiri izdane verzije govore o tome koliko se *Epic Games* zalaže za svoj *engine* tako što ga drže ažurnim i vjerodostojnim današnjim standardima. Smatram da će *Unreal Engine 5* podići standarde sa svojim mogućnostima koji su namijenjeni primarno za sljedeću generaciju videoigara. Iako danas sve velike kompanije koje se bave izradom igara (npr. *Rockstar Games*, *Blizzard* i dr.) imaju svoje *engine*, valja napomenuti kako su neki od njih vlastite varijacije *Unreal Engine*-a.

Unatoč tome što 3D igre, a posebice FPS igre zahtijevaju određeno znanje matematike, smatram da *Unreal Engine* svojim gotovim funkcijama dosta olakšava samo računanje te svojim ugrađenim sustavom kolizije olakšava manipuliranje modelima.

Vjerujem da u današnjem svijetu nedostaje više igara edukacijskog žanra s obzirom da većina današnje djece provodi slobodno vrijeme uz videoigre, no mislim da su i igre iz prvog lica korisne za razvijanje motorike i brzine koje igra testira.

Popis literature

[1] Luka Funda, Želimir Mikulić, Milan Hrga (2018) Osnovni elementi razvojnih alata Unity i Unreal Engine za stvaranje računalnih igara. Preuzeto 20. kolovoza 2020. s https://hrcak.srce.hr/198595

[2] Unreal Engine (2020.) Unreal Engine 4 Documentation. Preuzeto 20. kolovoza 2020. s https://docs.unrealengine.com/en-US/index.html

[3] DevSquad (2017.) Creating A First Person Shooter. Preuzeto 21. kolovoza s https://www.youtube.com/watch?v=DywBqQtTHMo&list=PLL0cLF8gjBprG6487lxgSq-aEo6ZXLDLg

[4] Niall McCarthy (2020.) Gaming: The Most Lucrative Entertainment Industry By Far. Preuzeto 10. rujna 2020. s https://www.statista.com/chart/22392/global-revenue-of-selected-entertainment-industry-sectors/

[5] First-person shooter (2020.) Wikipedia. Preuzeto 10. rujna 2020. s https://en.wikipedia.org/wiki/First-person_shooter

[6] Unreal Engine (2020.) Wikipedia. Preuzeto 10. rujna 2020. s https://en.wikipedia.org/wiki/Unreal_Engine

[7] Unreal Engine (2020.) A first look at Unreal Engine 5. Preuzeto 11. rujna 2020. https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

Popis slika

Slika 1: Najunosnija zabavna industrija do sada 1
Slika 2: MIDI Maze 2
Slika 3: Wolfenstein 3D 3
Slika 4: *Doom* 4
Slika 5: *Half-Life* 5
Slika 6: Izgled *Unreal Engine 4* korisničkog sučelja 7
Slika 7: Mortal Kombat 11 7
Slika 8: Izgled čvorova 9
Slika 9: Definiranje kontroli za upravljanje likom 10
Slika 10: Kretanje lika 11
Slika 11: Upravljanje kamerom 11
Slika 12: Izgled oružja za kupovinu 15
Slika 13: Algoritam izmjene oružja na utorima 15
Slika 14: Algoritam promjene oružja 16
Slika 15: Osluškivanje na desni klik miša 16
Slika 16: Transformiranje ruke za potrebe nišanjenja 17
Slika 17: Izračun smjera kretanja ispaljenog metka 18
Slika 18: Generiranje sfere i sile eksplozije 18
Slika 19: Drvo ponašanja neprijatelja 20
Slika 20: Lijeva putanja drva ponašanja 21
Slika 21: Desna putanja drva ponašanja 22
Slika 22: Granice *NavMeshBoundsVolume*-a 23
Slika 23: Vlastoručno napravljeni kanali kolizija 23
Slika 24: Postavke kolizije objekta u igri 24
Slika 25: Prikaz igračeve kolizije na mapi 25
Slika 26: Algoritam za početak runde i stvaranje neprijatelja 25
Slika 27: Algoritam za povećavanje broja runde i broja neprijatelja 26
Slika 28: Prikaz elemenata korisničkog sučelja 27

Izvori slika

Slika 1: https://cdn.statcdn.com/Infographic/images/normal/22392.jpeg

Slika 2: https://en.wikipedia.org/wiki/MIDI_Maze#/media/File:ST_Midi_Maze.png

Slika 3: https://vignette.wikia.nocookie.net/wl6/images/d/d8/Tw3d60e5l7.png/revision/latest?cb=20190425123030

Slika 4: https://media.playstation.com/is/image/SCEA/doom-1993-screenshot-02-ps4-us-26july2019?$native_nt$

Slika 5: https://steamcdn-a.akamaihd.net/steam/apps/70/0000002347.1920x1080.jpg?t=1591048039

Slika 7: https://assets.pcmag.com/media/images/553737-mortal-kombat-11-for-pc.jpg