IZRADA PLATFORMSKE IGRE U UNREAL ENGINE RAZVOJNOM OKRUŽENJU

Mihaljević, Duje

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:757963

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-05**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

DUJE MIHALJEVIĆ

Z A V R Š N I R A D

IZRADA PLATFORMSKE IGRE U UNREAL ENGINE RAZVOJNOM OKRUŽENJU

Split, rujan 2022.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Objektno orijentirano programiranje

Z A V R Š N I R A D

Kandidat: Duje Mihaljević

Naslov rada: Izrada platformske igre u Unreal Engine razvojnom okruženju

Mentor: Ljiljana Despalatović, viši pred.

Split, rujan 2022.

Sadržaj

Sažetak **1**

1 Uvod **2**

2 Općenito o Unreal Engineu **3**

3 Upoznavanje s razvojnim okruženjem **4**

3.1 Glavni prozor 5

3.2 Postavke projekta 12

3.3 *Blueprint* 13

3.3.1 Vizualni skriptni sustav *blueprinta* 14

4 Implementacija igre **15**

4.1 Glavni lik 15

4.2 Mijenjanje nivoa 26

4.3 Animacije glavnog lika 27

4.4 Pomična prepreka 31

4.5 Neprijatelj 33

4.6 Nivo 40

5 Zaključak **42**

Literatura **43**

Sažetak

Rad opisuje izradu platformske video igre. Igra je napravljena u Unreal Engineu koristeći razne alate i pomagala koja razvojno okruženje nudi. Kôd je pisan u C++ programskom jeziku jer ga koristi razvojno okruženje, ali i zato što je to objektno orijentirani programski jezik koji pruža visoku brzinu izvođenja programa i kontrolu korištenja memorije. S obzirom da razvojno okruženje nudi veliku količinu alata bit će spomenuti i opisani samo oni alati koji su korišteni prilikom izrade završnog rada.

U igri glavnu ulogu ima ženski lik. Ona je zatočena u tamnici i pokušava se izvući iz nje zaobilazeći prepreke i sakupljajući kipiće koji joj daju živote. Na kraju svakog nivoa nauči dodatnu moć sve dok se ne sukobi sa svojim neprijateljem.

Ključne riječi: Platformska video igra, platformer, Unreal Engine , C++

Summary

Development of platformer video game using Unreal Engine development environment

Paper describes platformer video game development. The game was developed in Unreal Engine by using different tools and assets that development environment offers. Code was written in C++ programming language because it is used by development environment but also because its an object-oriented programming language that offers high speed of program executing and memory management. Because of the large number of tools that this development environment offers, only tools that were used in the game development will be mentioned and described.

In the game the main character is female. She is trapped in a dungeon and she is trying to get out of it by overcoming obstacles and collecting statues that give her lives. At the end of each level she learns a new power until she meets her enemy.

Keywords: Platformer video game, platformer, Unreal Engine, C++

1. Uvod

Igra je širok pojam što je vidljivo prema više definicija Hrvatske enciklopedije [1], ali jednostavno rečeno to je oblik aktivnosti tijekom koje bi se osoba koja se bavi njome trebala zabaviti. Primijećena je kod mnogih sisavaca, a kao pojam se javlja i kroz povijest pa je lako pretpostaviti da igra među ljudima postoji otkad i sâm čovjek. Najčešće se veže uz djecu i mnoga istraživanja pokazuju važnost igre pri djetetovu razvoju i odrastanju. Postoje mnoge vrste igara i mogu se podijeliti na razne načine, a jedna od njih je video igra.

U današnje vrijeme izrada računalnih igara je jako velika i unosna industrija. Ona i dalje raste na većini platformi i vjerojatno neće stati rasti još dosta vremena. Nudi brojne mogućnosti zaposlenja, kako programerima, tako i nekim ne tehničkim strukama kao što su vizualni umjetnici, glazbenici, pisci. Tijekom godina nastali su razni tipovi igara kao što su pucačke (iz prvog ili trećeg lica), strateške (vojne, ekonomske, u stvarnom vremenu ili na poteze), sportske (nogomet, košarka, tenis), igre igranja uloga i mnoge druge. Naravno postoje i neke igre koje imaju odlike više različitih žanrova stvarajući poseban ugođaj.

Platformske igre nastaju početkom 1980-ih godina iako su igre s nekim elementima platformera postojale i ranije. Mogu se svrstati kao podžanr akcijskih igara. U većini platformskih igara je cilj proći prepreke i doći do kraja nivoa. Različite igre kao prepreku postavljaju različite stvari. Neke imaju neprijatelje, praznine gdje ne smijete upasti, ograde koje mogu naštetiti liku kojeg igrač kontrolira ili jednostavno od igrača zahtijevaju točno tajmiranje skokova ili drugih oblika kretanja (klizanje, puzanje, brže trčanje). Najčešće postoje i stvari koje se mogu skupljati pa tako igrač dobiva određene bodove. Neki od prvih naslova su Donkey Kong, Sonic the Hedgehog i Super Mario. Iz tih igara se razvio žanr i one su postavile temelje za mnoge druge koje će nastati. S razvojem tehnologije igre dobivaju više mogućnosti što se tiče igrivosti, grafike, veličine nivoa. Iako je tijekom godina popularnost platformskih igara opala, žanr ponovo dobiva na popularnosti s razvojem mobilnih igara i nastankom novih konzola, primarno konzola koje se drže u rukama (engl. *handheld consoles*).

Rad je podijeljen u tri poglavlja. U prvom poglavlju se kratko opisuje Unreal Engine i njegova povijest. Iduće poglavlje opisuje i objašnjava korištenje raznih alata i pomagala uz slike kako bi bilo lakše shvatiti o čemu se radi. Poglavlje implementacija igre se bavi opisom izrade raznih klasa i funkcionalnosti prikazanih unutar igre.

2. Općenito o Unreal Engineu

Unreal Engine je na službenoj stranici [2] opisan kao besplatan alat koji se najčešće povezuje uz izradu video igara, ali se tijekom godina razvio i trenutno nudi širok spektar alata u koje spadaju mogućnost izrade raznih vizualizacija, izrada filmskih i televizijskih sadržaja, trening, simulaciju i druge primjene. Unreal Engine dolazi spreman sa svim mogućnostima i alatima te izvornim kôdom.

Razvila ga je tvrtka Epic Games, a prvo je predstavljen s izradom pucačke igre Unreal. Ovaj pogonski alat je u početku bio primarno zamišljen kao podloga za pucačke igre, ali s vremenom njegova upotreba postaje sve šira. Unreal Engine je napisan u C++ jeziku te nudi podršku na različitim platformama.

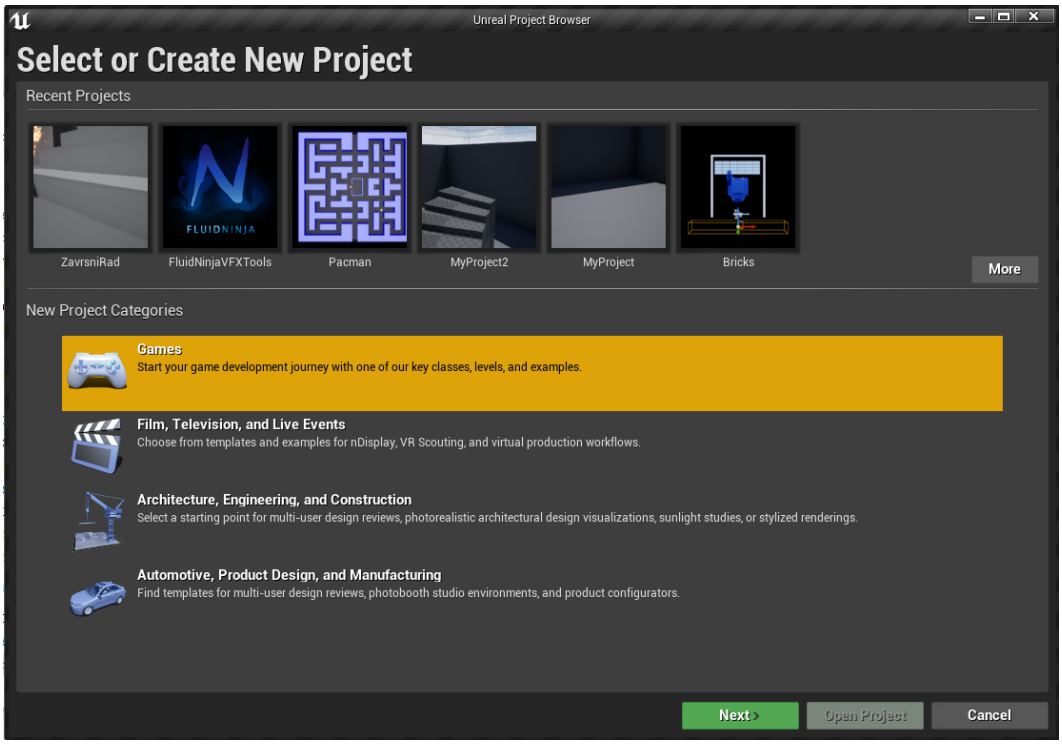
Najnovija verzija Unreal Enginea je 5, a ovaj rad je napravljen u verziji 4 jer potpuno izdanje verzije 5 nije izašlo kada je započet završni rad.

Na internetskoj stranici tvrtke Epic Games ili njihovoj aplikaciji ima poseban dio zvan tržnica (engl. *marketplace*) za Unreal Engine gdje se mogu naći mnogobrojni dodatci koje su napravili drugi ljudi. Postoje različiti projekti koje se može dodati unutar projekta u izradi ili jednostavnije stvari kao objekti za ukrasiti okolinu unutar igre, različiti likovi, animacije, kreatori terena gdje će se odvijati radnja igre, materijali koje se postavlja na objekte ili zvukovi i glazba kako bi igra imala i zvučnu komponentu. Dodatci se kupuju, ali neki mogu biti besplatni. Rad je napravljen upotrebom besplatnih dodataka.

Sve to moguće je kreirati samostalno unutar Unreal Enginea ili uvesti iz nekih drugih programa za uređivanje i izradu zvučnih ili vizualnih sadržaja.

3. Upoznavanje s razvojnim okruženjem

Na slici 1 je prikazan početni zaslon kada se pokrene pogonski alat. Na vrhu se nalaze projekti na kojima je nedavno rađeno, a ispod su predlošci za stvaranje novog projekta raspoređeni po nekoliko kategorija ovisno za što će se koristiti Unreal Engine.



Slika 1: Početni zaslon

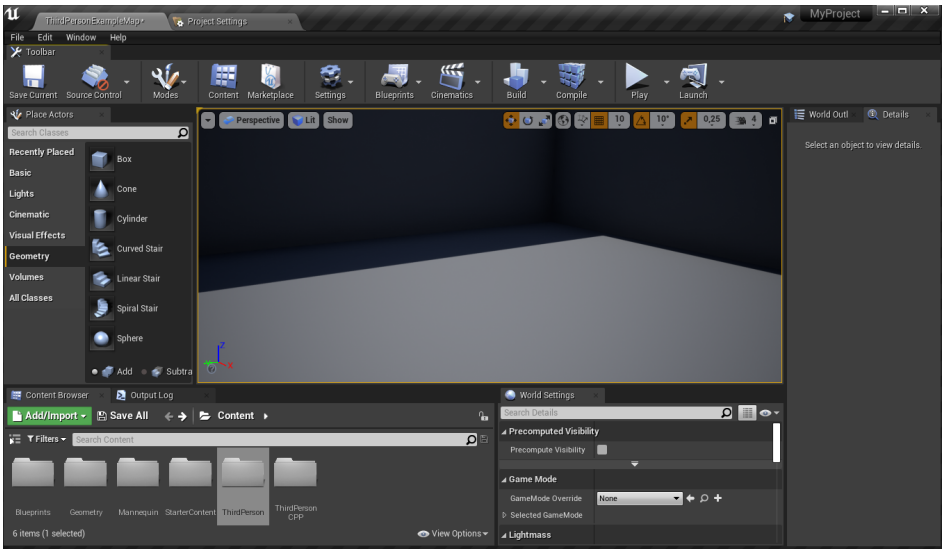
Najbliža ovom radu je naravno kategorija igre (engl. *games*). Odabirom te kategorije nude se različiti predlošci koji u projekt dodaju razne biblioteke i dodatke za lakšu izradu projekta. Također mogu dodati nekakve početne klase za koje se pretpostavlja da će ih projekt koristiti. Može se startati s potpuno čistim (engl. *blank*) projektom. Na slici 2 se vidi prikaz predloška u kategoriji igre. Ovaj rad je napravljen pomoću predloška trećeg lica (engl. *third person*).

Slika 2: Šablone u kategoriji igre

Prema dokumentaciji [3] predložak stvara lika kojeg igrač može kontrolirati različitim upravljačima, a prati ga kamera pozicionirana iza i malo iznad lika. Ovakva kamera stavlja naglasak na lika i često se koristi u akcijskim i avanturističkim igrama. Unutar projekta se dobije jednostavni nivo s nekoliko objekata po kojima se može kretati.

3.1. Glavni prozor

Izgled glavnog prozora može se vidjeti na slici 3. Taj izgled nije konačan. Korisnik može prema svojim željama i preferencijama pomicati različite prozore na mjesta gdje mu više odgovara te im mijenjati veličinu ili ih u potpunosti ugasiti i dodati neke druge.

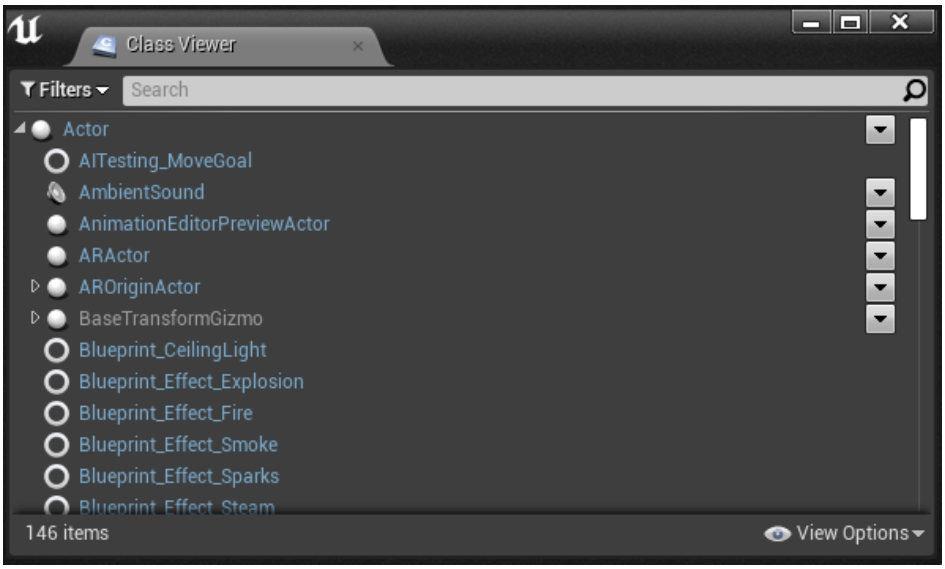


Slika 3: Glavni prozor

Pretraživač sadržaja (engl. *content browser*) nalazi se u donjem dijelu slike 3. U dokumentaciji [3] je definiran kao alat koji ima mogućnost pretraživanja cijelog sadržaja unu-

tar projekta, bez obzira je li datoteka sačuvana ili ne. Budući da ima pristup svemu služi kao prostor za uređivanje sadržaja mapa te operacije nad datotekama kao što su brisanje, otvaranje, premještanje ili kopiranje. Da bi pristup datotekama bio lakši ima tražilicu koja pretražuje prema nazivima, etiketama, putanjama i tipovima. U jednom trenutku mogu biti otvorena četiri pretraživača sadržaja radi lakšeg pristupa različitim dijelovima projekta. Alat pruža mogućnost stvaranja privatnih i lokalnih kolekcija kako bi njihov sadržaj bio iskoristiv u kasnijim projektima, a stvarajući djeljive kolekcije daje mogućnost da se projekt dijeli s drugim Unreal Engine korisnicima.

Alat preglednik klasa (engl. *class viewer*) pruža korisniku mogućnost da pregleda klase unutar projekta. Osim pregleda iz njega se mogu kreirati nove C++ i *blueprint* klase nasljeđujući odabranu klasu.



Slika 4: Izgled preglednika klasa

Alat je prikazan na slici 4. Na vrhu se vidi opcija filteri (engl. *filters*). Klase je moguće filtrirati prema 3 izbora. Samo sudionici (engl. *actors only*) izbor filtrira klase tako da prikazuje samo djecu sudionik (engl. *actor*) klase. Samo objekti za postavljanje (engl. *placable only*) filtrira sudionik klase koje se mogu postaviti u nivoe. Samo *blueprint* baze (engl. *blueprints bases only*) prikazuje one *blueprinte* od kojih se kreiraju novi. Pokraj nje se nalazi tražilica. Pretraga oboji dijelove imena klasa koji se podudaraju s upisanim pojmom. Ako se ime neke klase podudara, ali ime njezinog roditelja nije, ime roditelja bit će zamagljeno. Moguće je upisati više pojmova u tražilicu, ali tada klase koje se podudaraju s traženim pojmom neće imati označen dio imena. Opcije pregleda (engl. *view options*) nude odabir prikaza samo klasa trenutnog korisnika unutar razvojne mape (engl. *developer folder*) ili svih klasa. Može

se proširiti ili suziti sve klase kao i pojedinačnu klasu pritiskom na trokutiće lijevo od naziva. Pritiskom miša na klasu ona se učita te je se može ubaciti u prozor za prikaz. Kada je klasa učitana desnim klikom ili pritiskom miša na strelice u desnom dijelu uređivača otvaraju se opcije za uređivanje pojedine klase ili izrade nove *blueprint* klase prema odabranoj klasi.

Globalni odabir dodataka (engl. *global asset picker*) je još jedan alat za pronalazak potrebne datoteke, ali nije pogodan pregledavanju jer su sve datoteke u njemu naslagane u dugoj listi.

Uređivač nivoa (engl. *level editor*) je alat za pregled i izradu nivoa. Ubacivanjem sudionika (engl. *actors*) i uređivanjem njihovih svojstava nastaje prostor u kojem se odvija radnja igre.

Unutar Unreal Enginea [3] sudionici su svi objekti koji su ubačeni u nivo, odnosno objekti koji imaju 3D poziciju, rotaciju i mjerljive podatke, a to mogu biti izvori svjetla, izvori zvuka, likovi ili nepomični objekti.

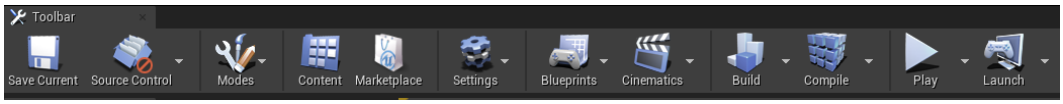
Traka prozora (engl. *tab bar*) nalazi se na vrhu s imenom trenutno otvorenog nivoa. U njoj se nalaze prozori drugih uređivača kako bi im se moglo lakše pristupiti, a ime prozora se podudara s imenom objekta koji se uređuje u njemu. Primjer otvorenih prozora vidljiv je na slici 5.

Slika 5: Izgled trake prozora

Traka izbornika (engl. *menu bar*) pruža pristup alatima i naredbama koje se koriste prilikom izrade nivoa. Može se pristupiti naredbama kao što je spremanje datoteka, kopiranje ili otvaranje različitih alata , a njen izgled je prikazan na slici 6.

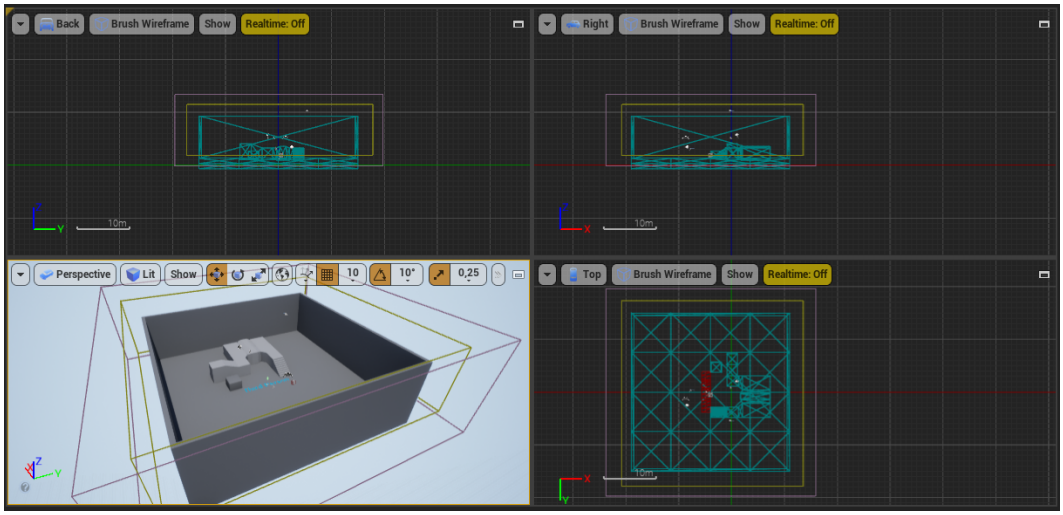
Slika 6: Izgled trake izbornika

Alatna traka (engl. *toolbar*) ima razne ikonice koje predstavljaju grupu naredbi, nudeći tako brz pristup često korištenim alatima i operacijama. Neki od ponuđenih izbora su spremanje trenutno otvorenog nivoa, pristup tržnici gdje se kupuju dodatci, paljenje igre ili gradnja (engl. *build*) projekta što je vidljivo iz slike 7.



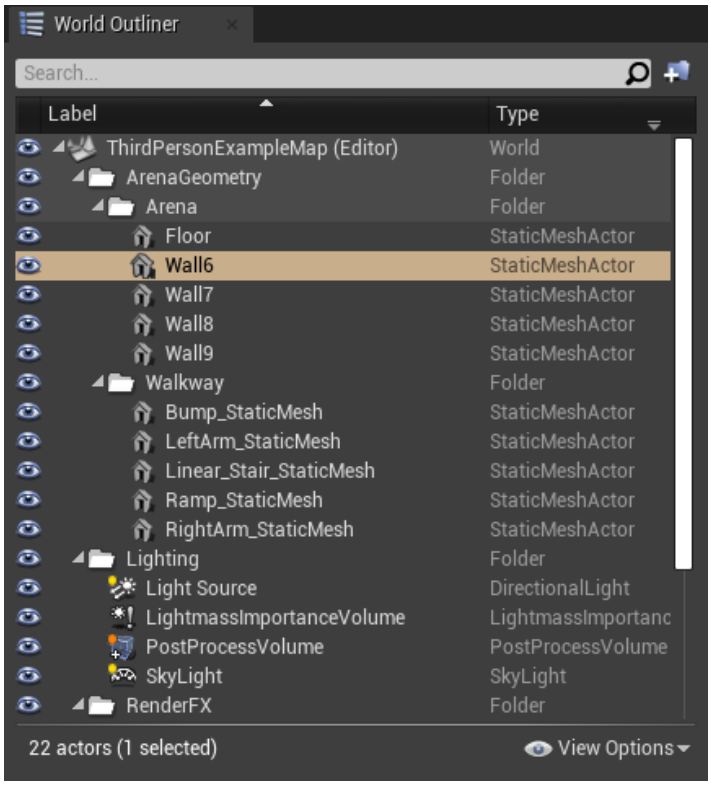
Slika 7: Izgled alatne trake

Prostor za prikaz (engl. *viewport*) služi pregledu nivoa te je moguće direktno u njega ubacivati stvari i mijenjati im svojstva. Može se sastojati od više prozora i svakom prozoru je moguće dodijeliti različitu perspektivu odnosno način prikaza. Prikazi koji nisu potrebni se mogu smanjiti. Na slici 8 je prikazan izgled s četiri otvorena prozora gdje jedan od prozora prikazuje izgled nivoa iz ptičje perspektive, jedan sa stražnje strane, jedan s desne strane, a jedan je perspektiva gdje se miče kamera i prilagođava pogled.



Slika 8: Prostor za prikaz

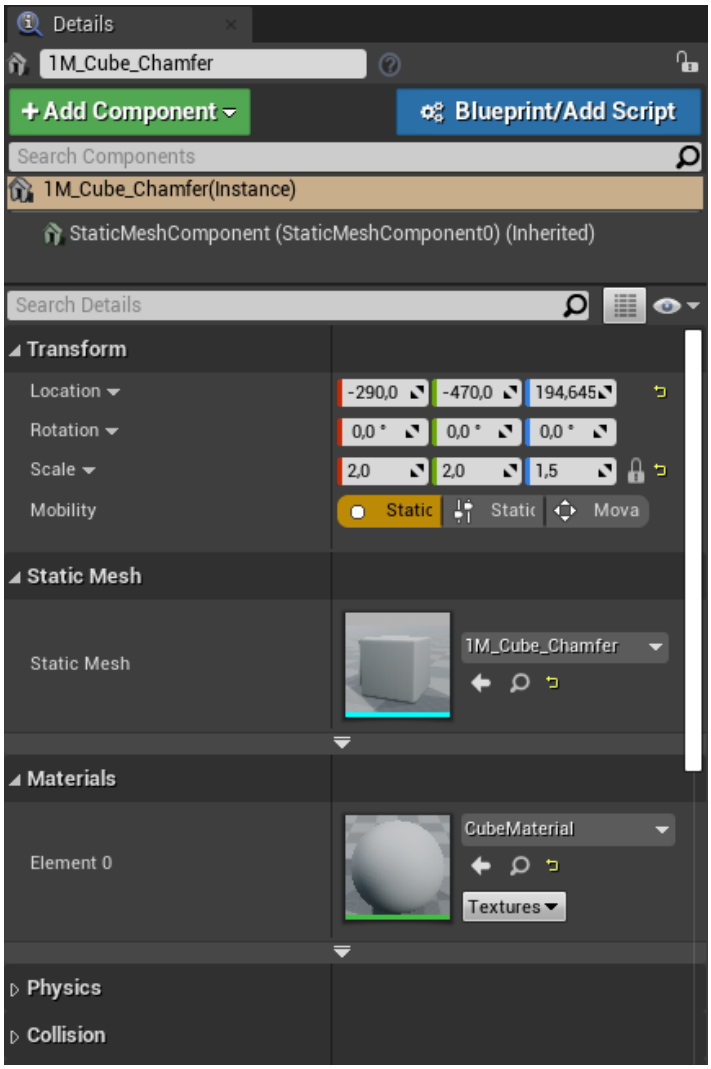
Obris svijeta (engl. *world outliner*) služi za pregled svih sudionika unutar nivoa. Da bi prikaz bio pregledniji moguće je dodavati mape i postavljati sudionike u njih. Odabirom nekog sudionika mogu se vidjeti njegovi detalji.



Slika 9: Obris svijeta

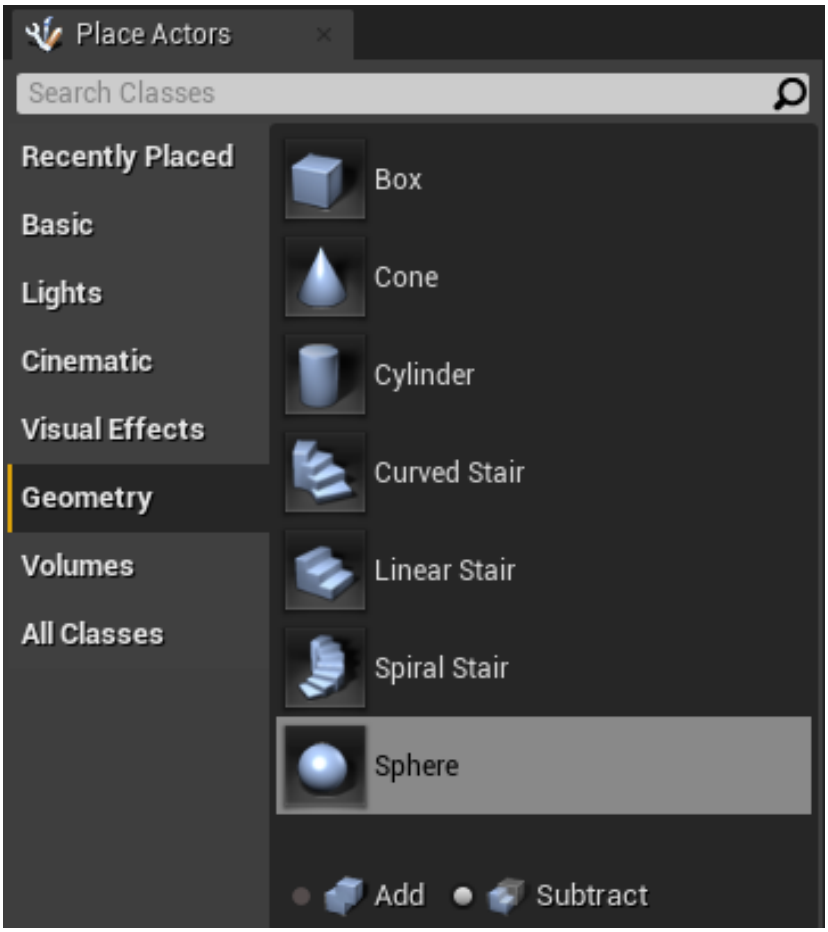
U obrisu svijeta se može pronaći gdje se nalazi neki objekt jer svijet koji nivo predstavlja može biti velik i nije uvijek lako znati gdje je što. Slika 9 prikazuje objekte koji se nalaze u početnom nivou iz predloška za igre iz trećeg lica.

Detalji (engl. *details*) su bitan dio, pogotovo kod testiranja igre. Odabirom nekog sudionika u prozoru za prikaz ili obrisu svijeta u prozoru detalji se prikažu njegovi podatci kao na slici 10. Tu je moguće mijenjati podatke koji su postavljeni u klasi ako je potrebno da se taj objekt ponaša ili izgleda drugačije. Moguće je vidjeti i mijenjati od kojeg oblika je napravljen objekt, koji materijal ima, stvari u vezi fizike, pomicati ga i još mnogo toga. Ako je to klasa mogu se uređivati javni podatkovni članovi.



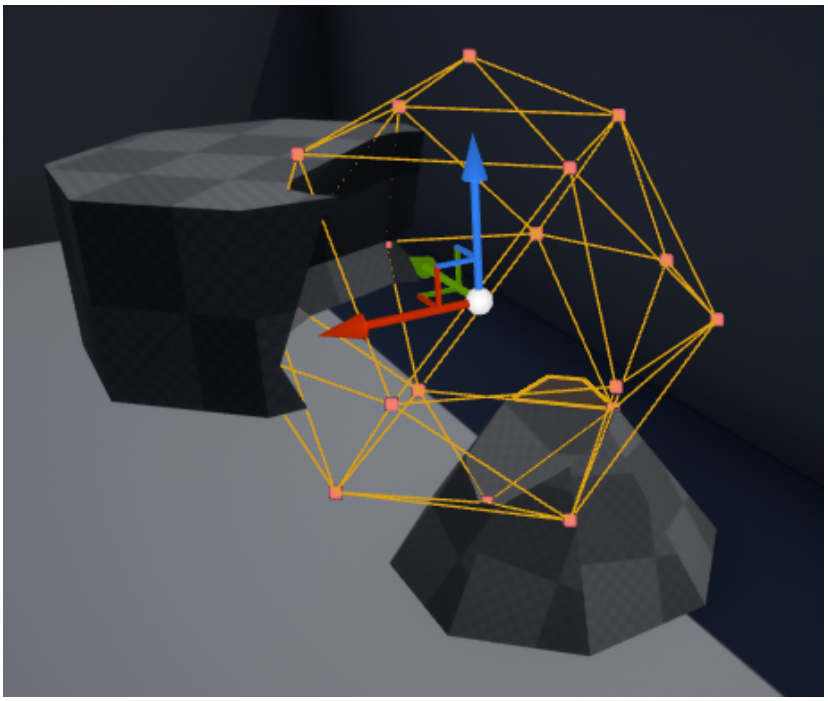
Slika 10: Prozor detalji

Jedan od standardnih prozora koji se otvore kod uređivanja nivoa je postavljanje sudionika (engl. *place actors*). Po određenim mapama podijeljene su različite klase koje se mogu povući mišem i ubaciti u nivo. U njemu se nalaze osnovne klase kao što je kocka, ravnina ili sfera što je prikazano na slici 11. Postoje specifičnije klase poput igračev početak (engl. *player start*) odakle kreće lik kojeg igrač kontrolira u tom nivou, postoje razni svjetlosni izvori, zvučni izvori i naravno svi sudionici napravljeni u trenutnom projektu.



Slika 11: Postavljanje sudionika

Ovaj alat također ima mogućnost pretraživanja kako bi se omogućilo brže i efikasnije korištenje. Na slici je prikazana jedna zanimljiva kategorija objekata koji se zovu *brush*. Dokumentacija navodi [3] da su se u prošlosti često koristili prilikom izrade nivoa, ali u današnje vrijeme ih najčešće mijenjaju efikasniji *mesh* objekti. *Brush* objekti se najčešće upotrebljavaju prilikom testiranja ili kako bi predstavljali dio nivoa napravljenog u grubo. Zanimljivo je što se takve objekte može ubacivati na dva načina: dodavanje (engl. *add*) i oduzimanje (engl. *subtract*). Na slici 12 se može vidjeti kako su dodani stožac i valjak, a od njih je oduzeta sfera.

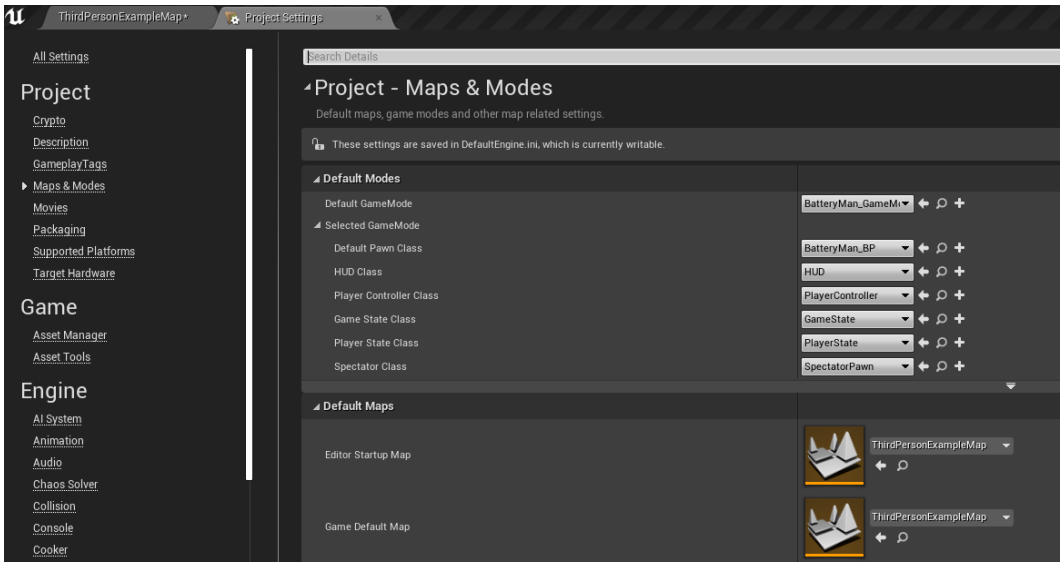


Slika 12: Dodavanje i oduzimanje *brush* objekata

Izlazni dnevnik (engl. *output log*) je koristan prozor u kojem je prilikom rada vidljiva prošlost svih akcija. Tako se na primjer pri kreiranju nivoa vidi koji sudionik je dodan ili izbrisan, koji sudionika je odabran, poništavajuće akcije (engl. *undo*). Isti prozor tijekom testiranja igre može služiti za razne ispise u konzolu pa je vidljivo jesu li se objekti sudarili, preklopili ili pozivaju li se destruktori klasa.

3.2. Postavke projekta

Preko trake izbornika odabirom uređivanja (engl. *edit*) pristupa se postavkama projekta (engl. *project settings*). Neke postavke su specifične za samu igru dok su druge općenitije i vezane su uz pogonski alat i platforme na kojima se projekt pokreće. U šabloni za igre iz trećeg lica su tako zadane postavke za lika s kojim će korisnik igrat, početna mapa kada se igra pokreće, zadana mapa u uređivaču nivoa, a vidljive su u primjeru na slici 13. Na lijevom dijelu slike nalazi se mnogo drugih izbornika koji prikazuju kompleksnost i količinu promjenjivih postavki.



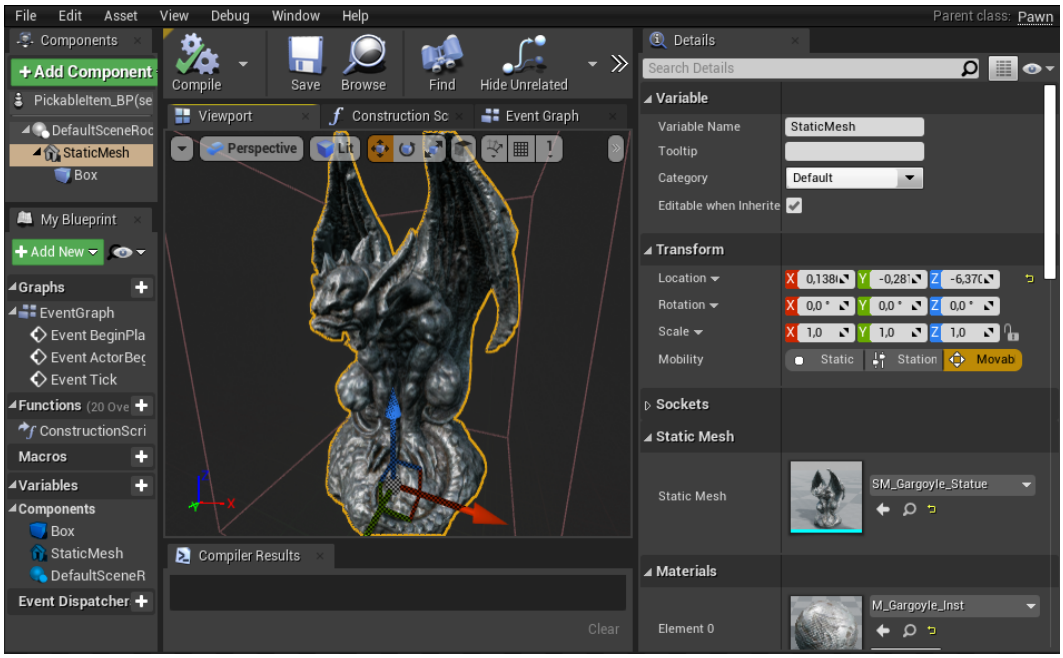
Slika 13: Postavke projekta

3.3. *Blueprint*

Blueprinti se prvi put pojavljuju u četvrtom izdanju Unreal Enginea. U dokumentaciji [3] su opisani kao način izrade klasa, metoda, događaja i skripti bez korištenja programskog jezika. S *blueprintima* se mogu raditi nove klase koje nasljeđuju drugu *blueprint* klasu ili neku C++ klasu. U *blueprintu* se može dodavati, slagati i mijenjati komponente sudionika, implementirati logiku koristeći vizualni skriptni jezik, definirati varijable, rukovati izlaznim podacima, definirati odgovore na razne događaje i okidače.

Izgled uređivača *blueprinta* ovisi o vrsti *blueprinta* koji se obrađuje u alatu. Međutim postoje neke sličnosti pa u svakom postoji traka izbornika gdje se mijenjaju pogledi, sprema datoteka, otvaraju i zatvaraju prozori. Postoje prozori koji su slični onima u prethodnim alatima kao prozor za prikaz gdje su vidljivi objekti ili skripta koja je napravljena u vizualnom skriptnom jeziku. Isto tako postoje detalji za uređivanje objekata. Alatna traka ima neke prečace za lakše korištenje. Uređivač *blueprinta* ima izgled koji je u potpunosti prilagodljiv korisnikovim potrebama. Primjer izgleda uređivača *blueprinta* je vidljiv na slici 14.

Blueprint klasa je klasa koja omogućava dodavanje funkcionalnosti postojećim klasama. Razlika između ovakve klase i C++ klase je u tome što se kod *blueprint* klase funkcionalnost ne implementira tipkajući kôd nego koristeći vizualni skriptni sustav *blueprinta*. One mogu nasljeđivati druge *blueprint* klase i C++ klase. Mogu im se dodavati varijable i funkcije kao što je običaj u objektno orijentiranom programiranju.



Slika 14: Izgled uređivača *blueprinta* za klase

Nivo *blueprint* je specijalizirana vrsta *blueprinta* koja se kreira automatski za svaki nivo u projektu. Novi nivo *blueprint* ne može biti kreiran sâm za sebe jer su oni uvijek vezani uz postojeći nivo. Kao i ostale vrste *blueprinta* može ih se uređivati te tako definirati neka svojstva nivoa.

3.3.1 Vizualni skriptni sustav *blueprinta*

Dokumentacija [3] daje objašnjenje da je to skriptni sustav za implementiranje klasa, funkcija i događaja gdje se ubacuju različiti čvorovi. Čvorovi se međusobno povezuju i tako se stvara tok programa. Čvorovi su gotove funkcionalnosti napravljene unutar Unreal Enginea, a mogu biti implementacija zbrajanja dva cijela broja, grananje, varijabla ili nešto kompliciranije kao traženje nasumične točke u prostoru unutar zadanog polumjera. Velika prednost ovog sustava je njegova fleksibilnost i jednostavnost te mogućnost da dizajnerima pruži alate koji su uobičajeno dostupni samo programerima. Bitno je naglasiti da postoji *markup* pomoću kojeg je u C++ kôdu moguće stvoriti osnovne sustave i nastaviti njihovo proširivanje unutar uređivača *blueprinta*. Kao i mnogi slični skriptni jezici koristi se za definiciju objektno orijentiranih klasa ili objekata unutar pogonskoga alata.

4. Implementacija igre

Radnja igre se odvija kroz četiri nivoa u kojima je stvoren ugođaj srednjovjekovne tamnice. Glavni lik kojim upravlja igrač je ženska osoba. Igrač sakuplja kipiće za dobivanje života. Prije nego se prijeđe pojedini nivo igrač dobije novu moć koja mu pomaže da prijeđe idući nivo. U nivou postoje razne prepreke koje treba proći zaobilaženjem, preskakanjem ili pomicanjem nekih objekata. Na zadnjem nivou se nalazi neprijatelj koji napada igrača, a pobjedom nad neprijateljem igra je završena.

4.1. Glavni lik

Funkcionalnosti glavnog lika napravljene su kroz C++ klasu `AMainCharacter` i *blueprint* klasu `MainCharacter_BP`. Klasa `AMainCharacter` nasljeđuje `ACharacter` klasu koju nudi Unreal, a *blueprint* klasa nasljeđuje `AMainCharacter`. Za početak je napravljena konstruktor metoda. U njoj se inicijaliziraju privatne i javne član varijable klase. Tu pripadaju neke jednostavne stvari kao broj života ili maksimalna brzina kojom se lik može kretati, ali i kamera koja prati glavnog lika. Zato postoji klasa `UCameraComponent` koja predstavlja kameru. U ovom slučaju može se reći da se funkcionalnost kamere napravila pomoću dvije član varijable, jedna varijabla je klasa `UCameraComponent` koja predstavlja vidno polje kamere i njene postavke. Druga je klasa `USpringArmComponent` koja održava djecu na fiksnoj udaljenosti od roditelja, ali ako nastane sudaranje ona će privući djecu i vratiti ih nakon što nestane sudaranja. Zato se u igri može primijetiti da će se kamera približiti liku ako je leđima previše blizu zida. Postavljanje kamere vidljivo je u kôdu 1.

```
CameraBoom = CreateDefaultSubobject<USpringArmComponent>
        (TEXT("CameraBoom"));
CameraBoom->SetupAttachment(RootComponent);

CameraBoom->TargetArmLength = 300.0f;
CameraBoom->bUsePawnControlRotation = true;

FollowCamera = CreateDefaultSubobject<UCameraComponent>
        (TEXT("FollowCamera"));
FollowCamera->SetupAttachment(CameraBoom,
```

```
        USpringArmComponent::SocketName);
FollowCamera->bUsePawnControlRotation = false;
```

Ispis 1: Dio konstruktora gdje je kreirana kamera

Sad se lik može postaviti unutar nivoa, ali osim gledanja u jednom smjeru nema drugih mogućnosti. Zato su iduće napravljene kretanje i rotacija. Za kretanje su implementirane metode `MoveForward` i `MoveRight`. Metode su praktički iste osim što se za kretanje naprijed uzima os X, a za kretanje desno os Y. U kôdu 2 se vidi implementacija za kretanje naprijed.

```
const FRotator Rotation = Controller->GetControlRotation();
const FRotator YawRotation(0, Rotation.Yaw, 0);
const FVector Direction = FRotationMatrix(YawRotation).
                    GetUnitAxis(EAxis::X);
AddMovementInput(Direction, Axis);
```

Ispis 2: Metoda koja definira kretanje prema naprijed

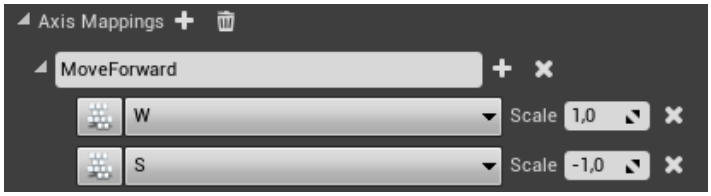
Klasa `ACharacter` ima metodu `SetupPlayerInputComponent` pa da bi se mogla implementirati mora ju se pregaziti (engl. *override*). Ta metoda dozvoljava klasi da joj se dodaju drugačije kontrole za upravljanje likom tako što se nove funkcije povežu s različitim osima i dodijeli im se ime pomoću kojeg će ih se pozvati unutar postavki projekta. U kôdu 3 se može vidjeti da su za okretanje kamere u stranu ili gore-dolje povezane osi s delegatima iz klase `APawn`. Za skok je funkcija iz klase `ACharacter`, koju je naslijedila, povezana s akcijom.

```
PlayerInputComponent->BindAxis("Turn", this,
                &APawn::AddControllerYawInput);
PlayerInputComponent->BindAxis("LookUp", this,
                &APawn::AddControllerPitchInput);

PlayerInputComponent->BindAction("Jump", IE_Pressed,
                this, &ACharacter::Jump);
```

```
PlayerInputComponent->BindAction("Jump", IE_Released,
                this, &ACharacter::StopJumping);

PlayerInputComponent->BindAxis("MoveForward", this,
                &AMainCharacter::MoveForward);
PlayerInputComponent->BindAxis("MoveRight", this,
                &AMainCharacter::MoveRight);
```

Ispis 3: Metoda koja dodaje kretanje

Da bi se iskoristile ove metode u postavkama projekta *bluepirnt* klasu se postavi kao zadanog lika. Zatim se u postavkama ode u kategoriju *Input* i dodaju nove metode. Za metodu `MoveForward` se postavi neka tipku s tipkovnice ili drugog željenog upravljača. Najčešće se za kretanje naprijed uzme W, a za kretanje unazad S. Jedina razlika je što se `Scale` za tipku S postavi na -1 što znači da će se kretanje odvijati u suprotnom smjeru. Primjer je prikazan na slici 15.



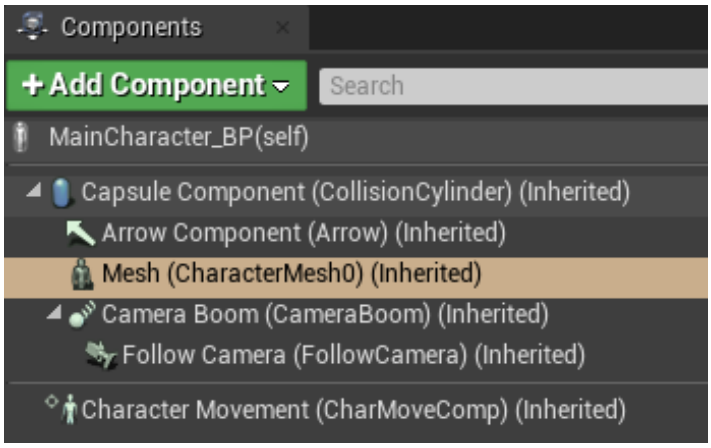
Slika 15: Korištenje funkcije za kretanje

Isti je odnos za kretanje desno i lijevo, za desno će `Scale` biti 1, a lijevo -1. Slično se postavi i za rotaciju, samo što se koriste ulazni signali s miša što je vidljivo na slici 16.

Slika 16: Korištenje funkcija za okretanje kamere

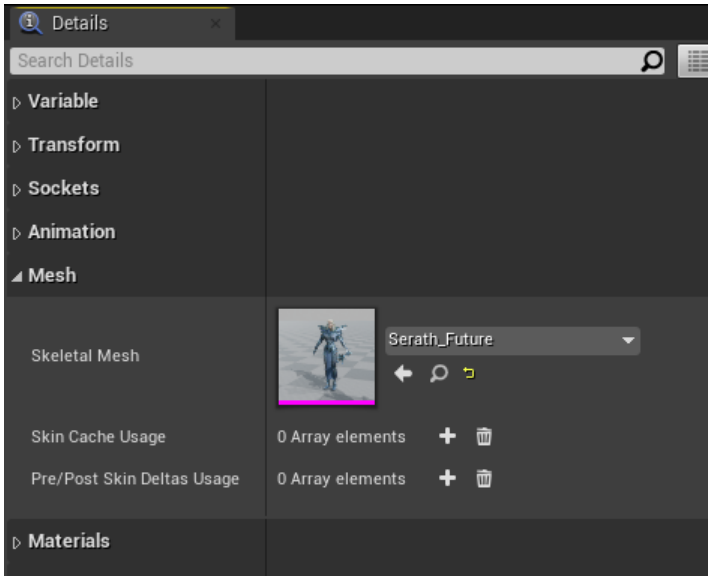
Za skok se samo postavila tipka koja će se koristiti jer je iskorištena postojeća metoda koju nudi Unreal. Da bi glavni lik imao izgled, s tržnice na kojoj se nude različiti dodatci za Unreal Engine preuzet je besplatni dodatak. Pošto klasa nasljeđuje `ACharacter` ima `Mesh` varijablu koja je pokazivač na klasu `USkeletalMeshComponent`. Ona predstavlja kostur i izgled glavnog lika. Preuzeti dodatak uvezen je u projekt i unutar *blueprinta*

dodijeljen liku. U uređivaču *blueprinta* se odabere komponenta `Mesh` kao na slici 17.



Slika 17: Odabir komponente koju želimo uređivati

Zatim se u prozoru detalja mogu vidjeti i urediti informacije o klasi. Odabere se dio koji se tiče izgleda i pronađe dodatak kao na slici 18 i sada je u igri vidljiv novi prikaz lika.



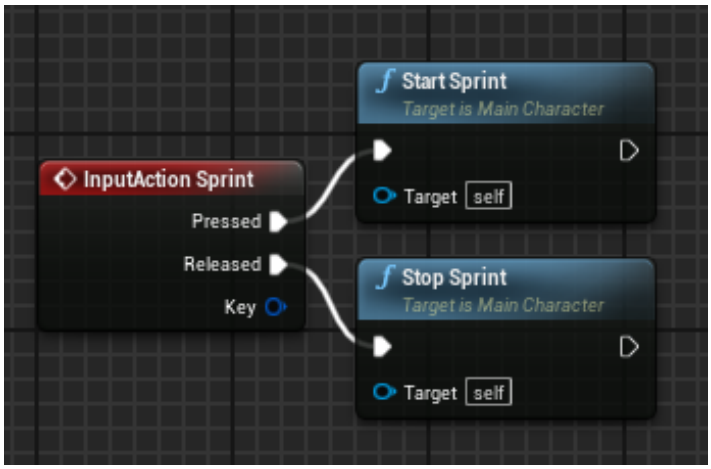
Slika 18: Dodavanje izgleda liku

Unutar klase `AMainCharacter` su implementirane funkcionalnosti bržeg trčanja i povlačenja objekata. Kada lik dobije mogućnost bržeg trčanja vrijednost njegove *boolean* varijable `CanSprint` se postavi na istinu. U postavkama projekta se doda nova ulazna vrijednost s tipkovnice i dodijeli joj se ime `Sprint` kao na slici 19.

Slika 19: Dodavanje novog ulaznog signala s tipkovnice

Sada se unutar *blueprint* klase `MainCharacter_BP`, pomoću vizualnog skriptnog sustava *blueprinta*, može napraviti akciju kada se pritisne tipka Q i pusti što je vidljivo

sa slike 20. Kada se tipka pritisne poziva se metoda `StartSprint`, a kada se pusti `StopSprint`.



Slika 20: Graf unutar *blueprint* klase

Metode `StartSprint` i `StopSprint` su implementirane unutar C++ klase. Da bi bile vidljive i iskoristive u *blueprintu*, u datoteci zaglavlja treba iskoristiti `markup` jezik koji nudi Unreal, a primjer se vidi iz kôda 4. `BlueprintCallable` znači da metode mogu biti pozvane unutar *blueprinta*.

```
UFUNCTION(BlueprintCallable)
   void StartSprint();

UFUNCTION(BlueprintCallable)
   void StopSprint();
```

Ispis 4: Definicija metoda za brzo trčanje

Obje metode su jednostavne. `StartSprint` poveća maksimalnu brzinu i ubrzanje lika te postavi *boolean* varijablu `IsSprinting` u istinu, dok `StopSprint` vraća te varijable na početne vrijednosti. I za kraj se u metodi `Tick` provjeri ima li dovoljno energije za trčanje.

```
if (!IsSprinting && Sprint < 200)
   Sprint += DeltaTime * 40;
else if (Sprint <= 5)
   StopSprint();
```

```
else if (IsSprinting)
   Sprint -= DeltaTime * 60;
```

Ispis 5: Dio Tick metode koji puni i prazni energiju

Varijabla `Sprint` sadrži količinu energije u obliku *floata*, odnosno koliko dugo se može brzo trčati. Dok lik brzo trči energija se prazni, a kad se potroši zove se `StopSprint`. Dok lik ne trči brzo energija se puni do određene granice. Metoda `Tick` je još jedna metoda koja je pregažena, a poziva se svaki vremenski okvir za nekog sudionika. Implementacija povlačenja objekta ima sličnosti s brzim trčanjem. U postavke projekta se doda tipku kojom će se započinjati povlačenje. Unutar *blueprinta* se postavi pozivanje metoda `StartPull` i `StopPull` prilikom pritiskanja i puštanja tipke. Također postoji *boolean* varijabla pomoću koje se određuje ima li lik dozvolu za vući objekte te *float* varijabla koja se prazni dok lik povlači objekt, odnosno puni dok ne povlači. Za povlačenje treba dodatni objekt. Zato je napravljena klasa `AInteractableObject`. Ona se sastoji od `Mesh` objekta koji je postavljen kao kocka. Osim njega dodana je kutija, koja unutar igre nije vidljiva, a služi da se odredi je li lik dovoljno blizu objektu. Ako lik uđe unutar te kutije aktivira se događaj preklapanja te je određeno da se pozove nova metoda `OnBeginOverlap` što je vidljivo unutar kôda 6. Taj dio kôda se nalazi u metodi `BeginPlay`.

```
GetCapsuleComponent()->OnComponentBeginOverlap.AddDynamic(this,
                        &AMainCharacter::OnBeginOverlap);
GetCapsuleComponent()->OnComponentEndOverlap.AddDynamic(this,
                        &AMainCharacter::OnEndOverlap);
```

Ispis 6: Postavljanje metoda za preklapanje

`OnBeginOverlap` provjerava s kojim objektom se preklopio lik. Unutar *blueprinta* za svaku klasu je dodan njihov nadimak i usporedbom se provjerava koja vrsta objekta je pokrenula `OnBeginOverlap`. Za povlačenje se provjeri je li to `AInteractableObject`, onda se uzme pokazivač na taj objekt i spremi u član varijablu lika kao u kôdu 7.

```
else if (OtherActor->ActorHasTag("InteractableObject")) {
```

```
    InRangeForPull = true;
    InteractableObject = Cast<AInteractableObject>(OtherActor);
}
```

Ispis 7: Postavljanje metoda za preklapanje

Kada se pritisne tipka za privlačenje unutar `Tick` metode se provjeri ima li dovoljno energije, ima li lik dozvolu za povlačenje i je li pokazivač stvarno neki objekt. Nakon toga na `Mesh` komponentu unutar Unreal Engine se može dodati sila. Uzimajući trenutne pozicije lika i objekta za povlačenje i oduzimajući ih dobije se vektor. Množeći vektor s masom objekta i nekim brojem, kako bi povlačenje bilo dovoljno brzo u igri, objekt se kreće prema liku. Kôd je prikazan na ispisu 8.

```
if (IsPulling && CanPull && this->InteractableObject != NULL)
{
    InteractableObject->Mesh->AddForce((this->GetActorLocation()
            - this->InteractableObject->GetActorLocation())
            * this->InteractableObject->Mesh->GetMass() * 5);
    InteractableObject->BoxCollision->SetWorldLocation(this->
            InteractableObject->Mesh->GetComponentLocation());
    InteractableObject->SetActorLocation(this->InteractableObject->
            Mesh->GetComponentLocation());
}
```

Ispis 8: Povlačenje objekta

Kao i u svakoj platformskoj igri napravljene su klase koje predstavljaju objekte za prikupljanje. Klasa `PickableItem_BP` je jako jednostavna pa je u potpunosti napravljena kao *blueprint* klasa. Sastoji se od izgleda i kutije za preklapanja. Unutar lika već spomenuta metoda `OnBeginOverlap` može provjeriti je li objekt `PickableItem_BP` i ako je to istina uvećava `Coins`. Kad se sakupi 100 `Coins` dobije se 1 život. Prikupljeni `PickableItem_BP` se prilikom preklapanja briše kao u kôdu 9.

```
if (OtherActor->ActorHasTag("Recharge")) {
```

```
    Coins += 1;
    if (Coins >= 100)
    {
        Coins = Coins - 100;
        ++Lives;
    }
    OtherActor->Destroy();
    GIMC->RefreshMyChar(this);
}
```

Ispis 9: Sakupljanje novčića

Klasa `LootBox` predstavlja kutiju koja se razbije i onda se stvore `PickableItem_BP`. Za razbijanje klasa ima kutiju za preklapanje s kojom glavni lik provjerava preklapanje i uništava je. Komponenta izgleda je postavljena unutar *blueprint* klase kao za glavnog lika i `PickableItem_BP`. Stvaranje `PickableItem_BP` je prikazano kôdom 10, a ta metoda se nalazi u `LootBox` klasi. `CoinSpawn` predstavlja vrstu objekta koji će se stvoriti, a njegova vrijednost je postavljena unutar *blueprint* klase `LootBox_BP`.

```
FRotator SpawnRotation = FRotator(0.0f, 0.0f, 0.0f);
FVector BoxLocation = this->GetActorLocation();
BoxLocation.Z += 100;
for (int i = 1; i < 5; ++i) {
    BoxLocation.X += (30 * i);
    GetWorld()->SpawnActor(CoinSpawn, &BoxLocation, &SpawnRotation);
}
this->Destroy();
```

Ispis 10: Stvaranje novčića i uništavanje kutije

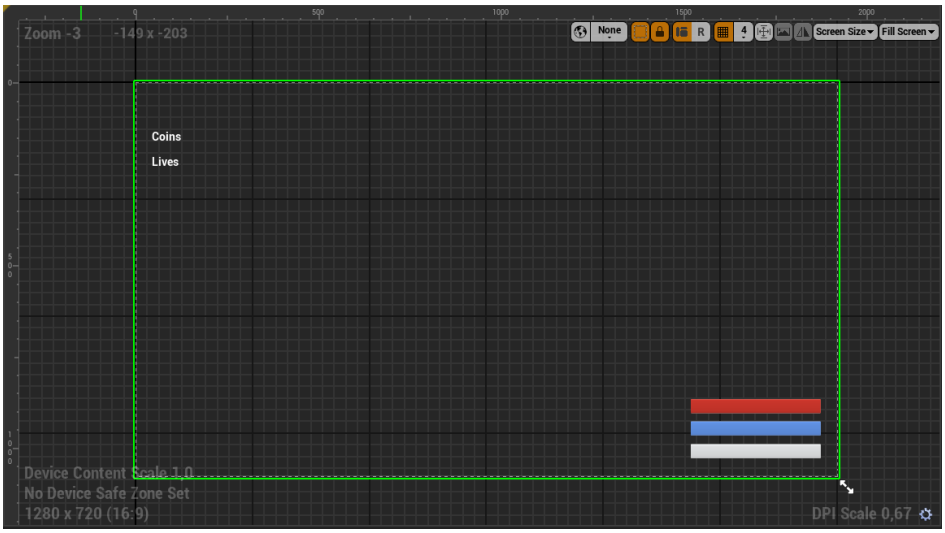
Mogućnosti kao što je brže trčanje, povlačenje objekata i dvostruki skok lik dobiva skupljanjem objekta koji nastaje iz klase `PowerUp`. Klasa ima javnu varijablu u koju je spremljena jedna od tri *enum* vrijednosti. Ovisno o vrijednosti postoje tri *blueprint* klase izvedene iz ove C++ klase. Kada se u `OnBeginOverlap` dogodi preklapanje s jednom od njih *bool*

varijable se mijenjaju kao u kôdu 11.

```
else if (OtherActor->ActorHasTag("PowerUp")) {
   APowerUp* powerUp = Cast<APowerUp>(OtherActor);
   if (powerUp->PowerUpType ==
                EPowerUpType(EPowerUpType::DoubleJump)){
      this->DoubleJump = true;
      JumpMaxCount = 2;
   }
   else if (powerUp->PowerUpType ==
                EPowerUpType(EPowerUpType::Sprint)){
      this->CanSprint = true;
   }
   else if (powerUp->PowerUpType ==
                EPowerUpType(EPowerUpType::Pull)){
         this->CanPull = true;
   }
   powerUp->CollectPowerUp();
   GIMC->RefreshMyChar(this);
}
```

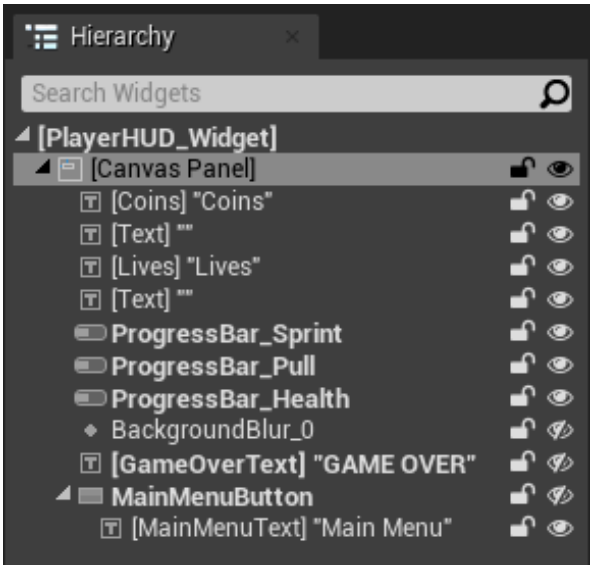
Ispis 11: Dobivanje pojačanja

Da bi se vidjelo koliko života, novčića, energije za trčanje ima lik napravljeno je vizualno sučelje koje te informacije prikazuje igraču. Takvo sučelje se može jednostavno napraviti unutar Unreal Enginea. Odabere se izrada `Widget Blueprint`. Izrada sučelja unutar ovog uređivača podijeljena je u dva dijela. U prvom planu je dizajner gdje se definira izgled sučelja. Na prozor za prikaz se ubacuju razni natpisi, botuni, trake koje se mogu puniti i prazniti. Drugi dio je graf gdje se pomoću vizualnog skriptnog jezika rade funkcije i uređuje graf događaja (engl.*event graph*). Tijekom normalnog igranja vizualno sučelje prikazuje broj novčića i života kao brojeve. Količina zdravlja (engl.*health*), energija za trčanje i energija za povlačenje su prikazane kao trake koje se pune i prazne. Ako je lik ostao bez života ekran se zamagli i otkriva se natpis `Game Over` i botun `Main Menu` koji vraća u glavni izbornik igre. Na slici 21 je prikaz sučelja unutar dizajnera.



Slika 21: Sučelje unutar dizajnera

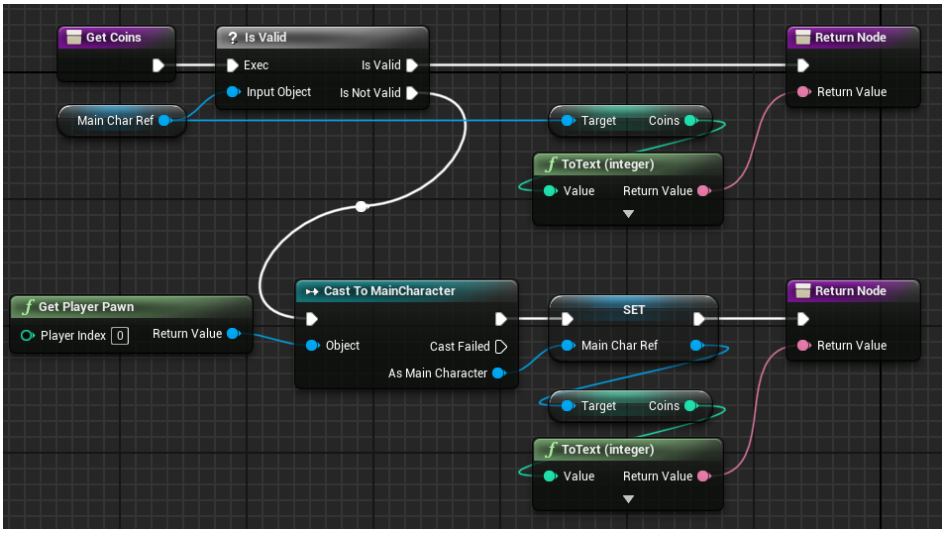
Na slici se u ovom slučaju ne vide natpis `Game Over`, botun `Main Menu` ni zamagljen ekran jer je njihova vidljivost isključena pomoću prozora hijerarhije (engl.*hierarchy*) što se vidi na slici 22 prema prekriženoj ikoni oka u desnom dijelu.



Slika 22: Prikaz hijerarhije dodataka stavljenih na sučelje

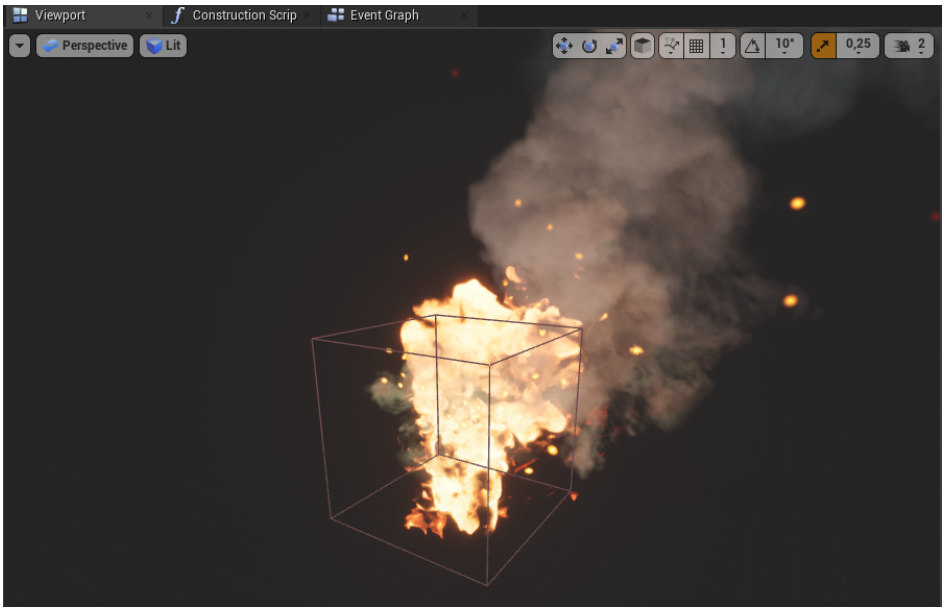
Nakon izgleda sučelja izrađene su njegove funkcionalnosti. Napravljene su funkcije za dohvaćanje vrijednosti varijabli iz klase pomoću vizualnog skriptnog jezika. Unutar graf dijela ovog uređivača odabere se izrada nove funkcije. Nadjene joj se ime i unutar prozora za prikaz se stvori prvi pravokutnik koji je u biti početak funkcije. Primjer dohvaćanja novčića se može vidjeti na slici 23. U klasi koja predstavlja sučelje je dodana referenca na lika. U ovoj funkciji se provjerava je li referenca valjana i ako nije druga funkcija dohvaća lika te mu se referenca postavlja unutar varijable. Zatim se može dohvatiti vrijednost novčića, pretvoriti je u tekst i vratiti kao povratnu vrijednost. Idući put kad se uđe u funkciju referenca

bi trebala biti valjana tako da se iz nje dohvaćaju novčići i vraćaju kao povratnu vrijednost. Da bi to funkcioniralo u dizajn dijelu se odabere tekstualni prozor koji će ispisivati novčiće i u detaljima se postavi da će sadržaj dobivati preko funkcije koja je nazvana `Get Coins`. Na sličan način su odrađene i ostale funkcije za prikaz vrijednosti na sučelju.



Slika 23: Dohvaćanje novčića za prikaz na sučelju

Da bi igra imala izazov stvorena je klasa `ADamagingElements`. Sastoji se od nekoliko pokazivača. Ima kutiju za preklapanje, `USceneComponent` koja je postavljena kao korijenska komponenta i `UParticleSystemComponent` koji služi za prikaz objekta, ali s obzirom na to da je ovaj objekt vatra ne može se iskoristiti *mesh*. Kada lik dođe unutar kutije za preklapanje, kao i dosad, u `OnBeginOverlap` se provjeri je li objekt ova vatra, odnosno `ADamagingElements`. Ako je to slučaj unutar `AMainCharacter` klase postoji *boolean* varijabla `IsInFire` kojoj se dodjeljuje vrijednost istine. Unutar `Tick` metode se cijelo vrijeme provjerava ta varijabla, a kad ona poprimi vrijednost istine pokreće metodu `Burn` koja liku smanjuje vrijednost `Health`. Ako `Health` padne ispod nule lik umire. Kada lik izađe iz kutije za preklapanje klase `ADamagingElements` zove se metoda `OnEndOverlap`, provjerava se je li prekinuto preklapanje s `ADamagingElements`. Ako nema preklapanja varijabla `IsInFire` postavlja se u laž te se više ne gubi `Health`. Izgled vatre se može vidjeti unutar prozora za prikaz *blueprint* klase `DamagingElements_BP` što je prikazano na slici 24. Kada se vrijednost `Health` spusti ispod 0, unutar `Tick` zaustavi se mogućnosti kretanja liku te mu se `Mesh` pretvori u krpenu lutku tako što se omogući fizika kostima unutar `mesh` objekta. Lik se sruši i nakon pauze od 2 sekunde, ako ima života, poziva se metodu koja ponovo pokreće nivo. Ako nema života vizualno sučelje prikazuje botun za povratak na `Main Menu` i javlja da je igra gotova.



Slika 24: Prikaz vatre

4.2. Mijenjanje nivoa

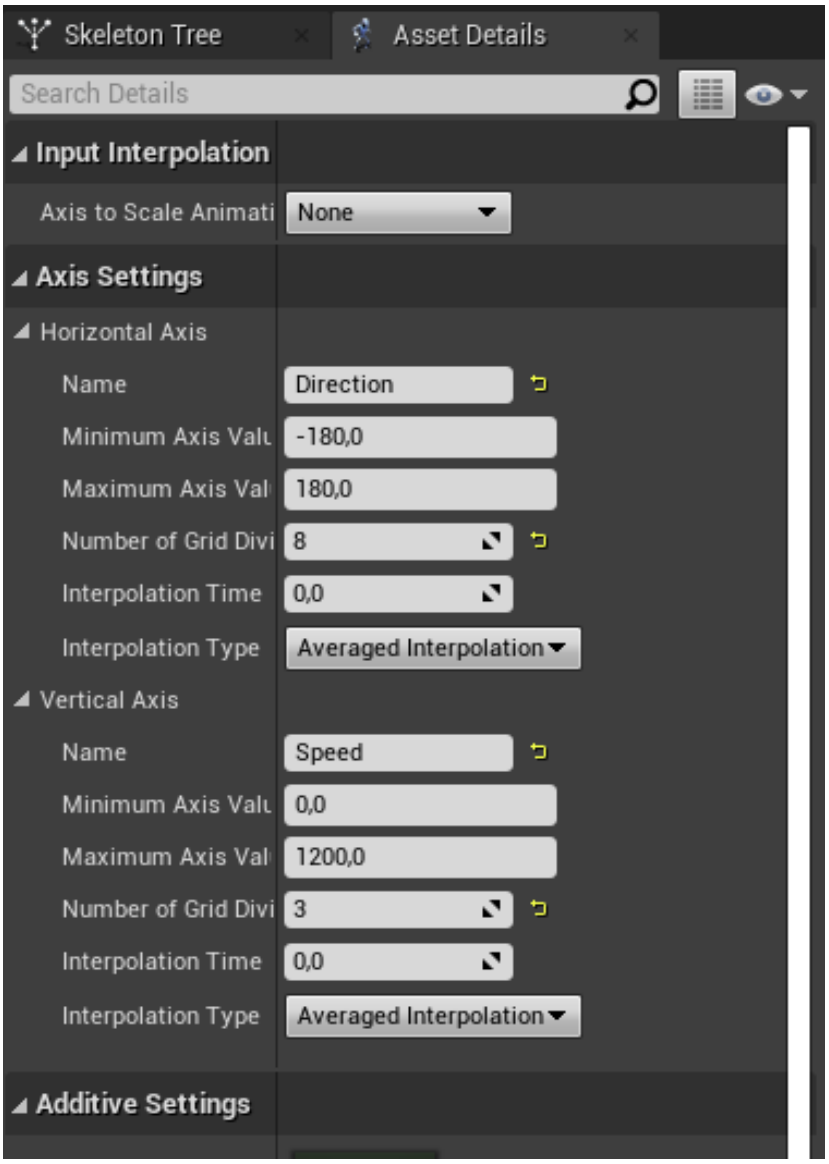
Da bi se mogli mijenjati nivoi napravljena je klasa `LevelChange`. Klasa sadrži `FString NextLevelName` u kojemu drži ime idućeg nivoa te kao i dosadašnje klase ima kutiju za preklapanje koju detektira lik u `OnBeginOverlap` metodi. Tada se poziva metoda `ChangeLevel` u kojoj se promijeni nivo prema član varijabli `NextLevelName`. Kada se mijenja nivo iskorištena je funkcija `Open Level` koja je dio Unreal Enginea. Ona automatski zatvara trenutni nivo i oslobađa memoriju. Zbog toga bi se izgubili podatci o broju novčića i života. Da bi se pamtili neki podatci mijenjajući nivoe postoji klasa `UGameInstance` koja nastaje prilikom paljenja igre i postoji dok se igra ne ugasi. Napravljena je klasa `UGameInstanceMainChar` u kojoj se bilježe neke od varijabli lika kao broj života ili mogućnost duplog skoka. Kada se nivo promijeni kreira se novi objekt klase koja predstavlja lika i vrijednosti njegovih varijabli se postave prema onima iz klase `UGameInstanceMainChar`. Prilikom skupljanja objekata dok igramo u našem liku se mijenjaju varijable, a njihove vrijednosti osvježavamo pomoću metode `RefreshMyChar` koja se nalazi u `UGameInstanceMainChar` klasi. Pošaljemo pokazivač na našeg lika i vrijednosti njegovih varijabli zapišemo u varijable `UGameInstanceMainChar`. Kada lik umre također se uništava objekt pa je ponovo potrebno učitati stvari koje su izgubljene. To se odrađuje unutar `Tick` metode. U konstruktoru postavimo član varijablu `IsUpdated` na neistinu i kada `Tick` dođe do grananja u prvom prolazu će osvježiti vrijednosti varijabli što je prikazano u kôdu 12.

```
if (!IsUpdated)
{
   UGameInstanceMainChar* GIMC = Cast<UGameInstanceMainChar>
                     (GetGameInstance());
   if (GIMC)
   {
      CanPull = GIMC->CanPull;
      DoubleJump = GIMC->CanDoubleJump;
      CanSprint = GIMC->CanSprint;
      Health = GIMC->Health;
      Coins = GIMC->Coins;
      Lives = GIMC->Lives;
      if (DoubleJump)
         JumpMaxCount = 2;
      IsUpdated = true;
   }
}
```

Ispis 12: Osvježavanje podataka u glavnom liku

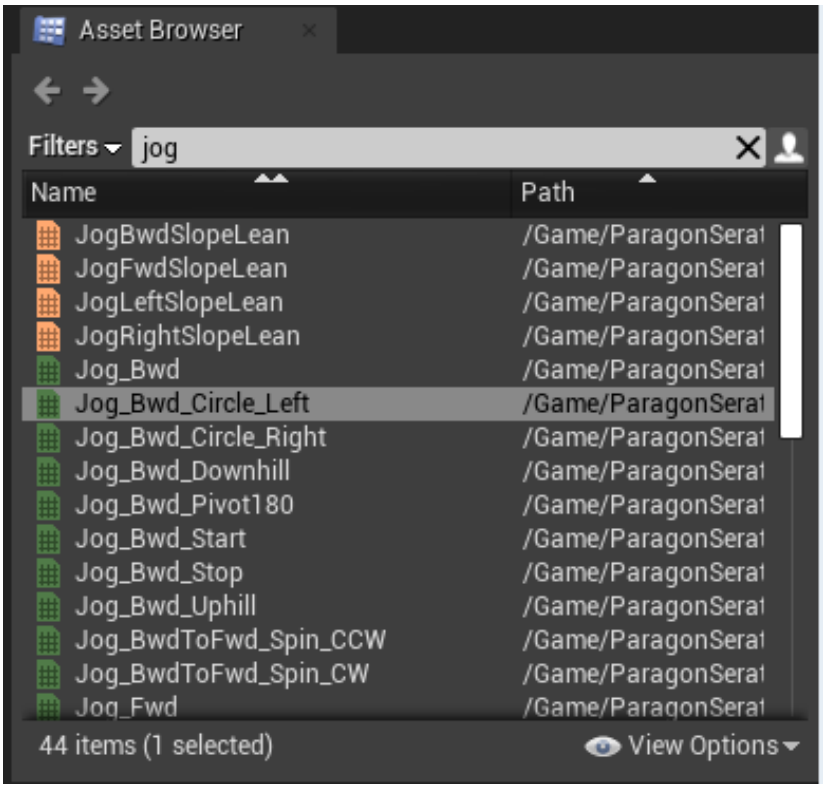
4.3. Animacije glavnog lika

Animacije su uvezene s tržnice u istom dodatku kao i izgled lika. Za početak se unutar Unreal Enginea napravi *Blend Space*. Zatim se odabere kostur lika za kojeg se radi animacija. Alat za izradu na dnu ima koordinatnu mrežu, a s lijeve strane postoje detalji pomoću kojih se određuju stvari koje se tiču koordinatne mreže. Na slici 25 se vidi da postoje horizontalna i vertikalna os.



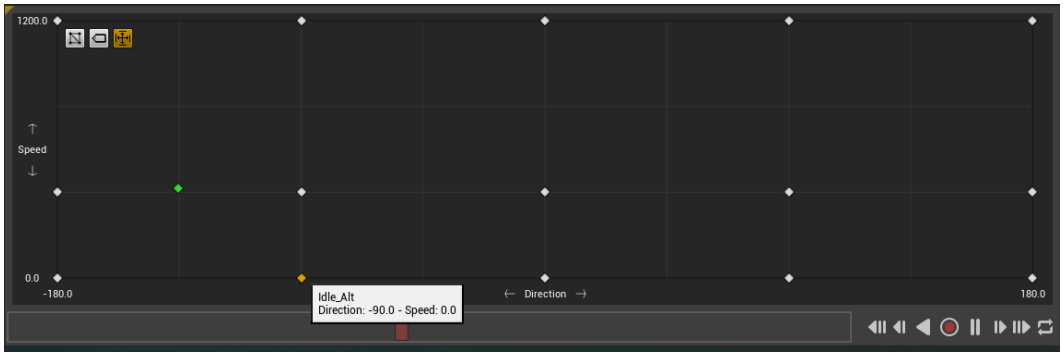
Slika 25: Detalji koordinatne mreže

Horizontalna os je nazvana usmjerenje (engl. *direction*) te joj je postavljena maksimalna vrijednost na 180, a minimalna vrijednost na -180. Os je podijeljena na 8 dijelova. Vrijednosti -180 i 180 predstavljaju stupnjeve. Vertikalna os je nazvana brzina (engl. *speed*). Maksimalna vrijednost joj je postavljena na 1200, a minimalna na 0. Vertikalna os je podijeljena na 3 dijela. U donjem desnom dijelu je popis animacija koje su uvedene što se vidi na slici 26. Dvoklikom na neku od njih otvara se novi prozor u kojem se vidi kako animacija izgleda.



Slika 26: Popis animacija

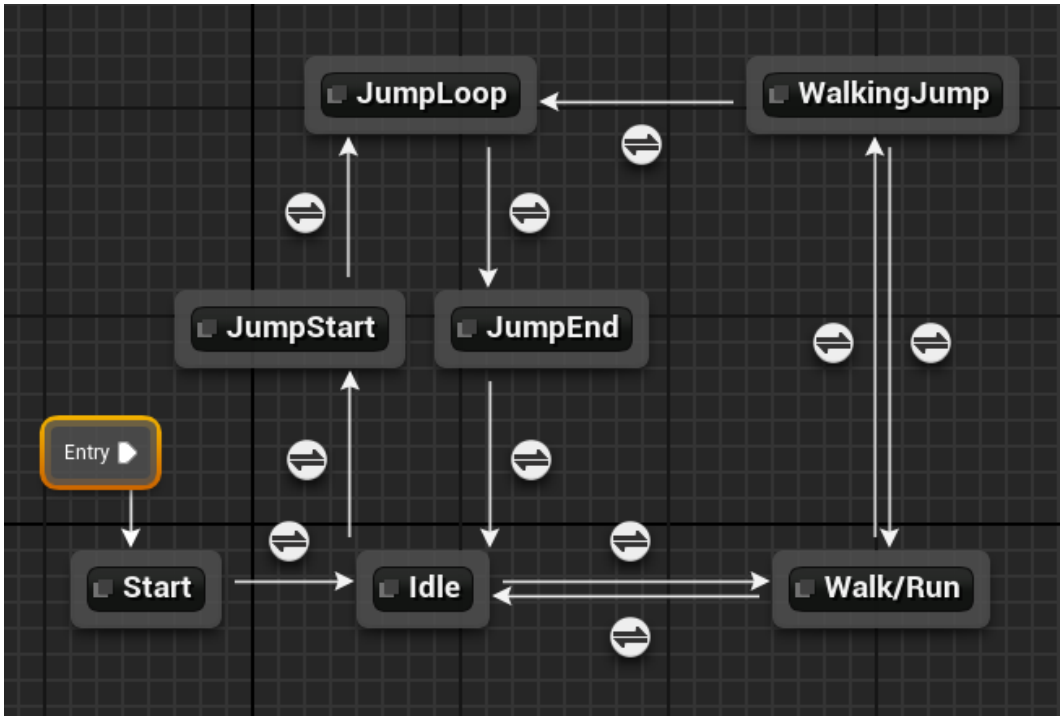
Na koordinatnoj mreži mirovanje predstavlja minimalna vrijednost vertikalne osi što je 0, odnosno to je brzina kretanja lika. Animacije mirovanja su predstavljene kao pet kvadrata na dnu koordinatne mreže sa slike 27. Među animacijama se nađe ona koja je predviđena za mirovanje, u ovom slučaju je to *Idle* i postavi je se na minimalnu vrijednost za kretanje naprijed što je vrijednost 0 na horizontalnoj osi, na vrijednosti -90 za kretanje lijevo, 90 za kretanje desno te na -180 i 180 za unatrag.



Slika 27: Koordinatna mreža

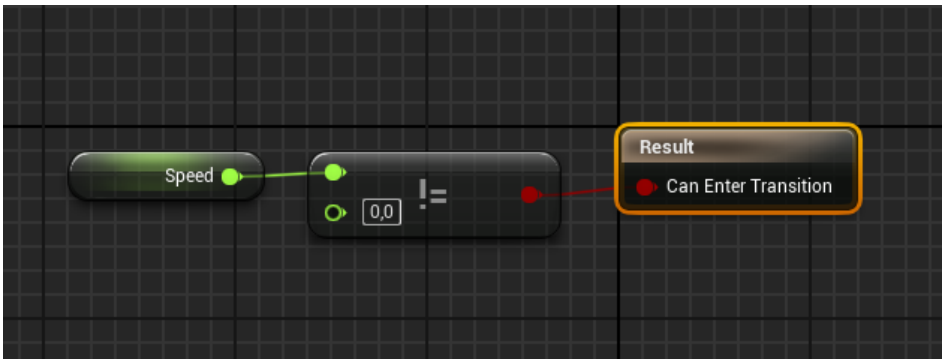
Nakon toga se odabere animacija za kretanje naprijed i postavi na vrijednost 0 horizontalne osi, a vertikalna vrijednost je 400 jer je to regularna brzina kretanja lika. Za kretanje nazad na vrijednosti horizontalne osi -180 i 180 te vertikalne 400 postavljena je animacija `Jog_Bwd`. Za kretanje normalnom brzinom je ostalo za dodati kretanje desno i lijevo. Obje

animacije idu gdje je vrijednost vertikalne osi 400, za kretanje lijevo horizontalna os je -90, a za kretanje desno 90. Sada lik ima animacije dok se kreće od stajanja do regularne brzine, a alat računanjem stvara animacije za kretanje na primjer desno i naprijed, iako takve animacije nema u ovom slučaju. Sada treba dodati četiri animacije za kretanje kada lik trči većom brzinom, odnosno za implementaciju `sprinta`. Pronađu se pripadajuće animacije i doda ih se slično kao i do sada osim što ih se stavlja gdje je vrijednost vertikalne osi 1200, a to je brzina koju lik može imati kada dobije mogućnost brzog trčanja. Da bi lik mogao prolaziti kroz ove animacije u `Blend Space` odabere se *blueprint* odjel gdje se za početak napravi potreban događaj (engl.*event*) koji će osvježavati brzinu, smjer kretanja i provjeru nalazi li se lik u padu. Nakon toga napravljen je konačni automat (engl.*state machine*) da bi se mogle mijenjati animacije iz stajanja u trčanje ili skok. Primjer se može vidjeti na slici 28. Smjer strelica nam govori iz kojeg stanja se može ući u drugo stanje i predstavlja uvjete za mijenjanje stanja. Uvjeti za ulazak u neko od stanja vezanih sa skakanjem je da se lik nalazi u zraku.



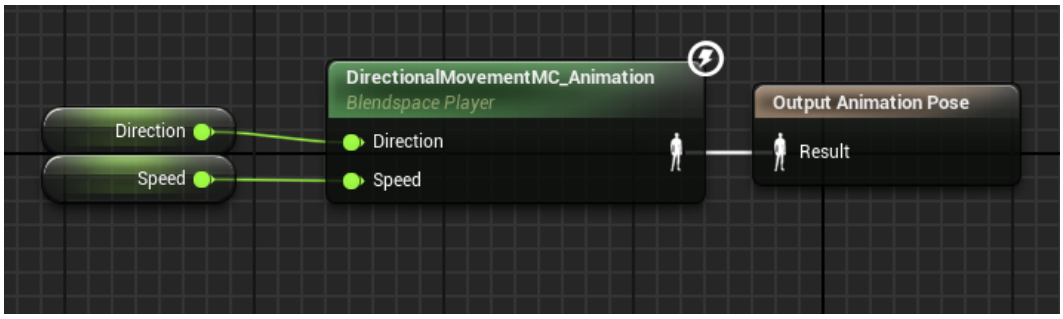
Slika 28: Konačni automat

Klikom miša na strelicu koja ide od stanja `Idle` prema stanju `Walk/Run` mogu se vidjeti njeni detalji, a ako se klikne dva puta može se urediti kako će se prelaziti iz `Idle` u `Walk/Run`. Iz slike 29 je vidljivo da se provjeri brzina kretanja i ako je različita od 0 ulazi se u čvor `Walk/Run`.



Slika 29: Uvjet prelaska između stanja

Otvaranjem stanja `Walk/Run` pomoću vizualnog skriptnog jezika uzme se `Blend Space` u koji su stavljene animacije. Potrebne ulazne varijable su `Speed` i `Direction` kako su nazvane koordinatne osi. Pošto je napravljen događaj koji osvježava vrijednosti tih varijabli može ih se samo poslati u `Blend Space` kao na slici 30 i iz njega izlazi potrebna animacija koja se prikazuje na ekranu.



Slika 30: Primjer stanja

4.4. Pomična prepreka

Klasa koja se zove `AMovingObstacle` je poslužila u više svrha. Ona može služiti da zasmeta kada se pokušava proći nekuda ili može biti dio nivoa koji pomaže liku da se približi dovoljno blizu drugom objektu za skok. Klasa ima `Mesh` varijablu da bi se mogao postaviti neki oblik. Dvije član varijable nazvane `Start` i `End` su tipa `FVector`, a predstavljaju točke u prostoru koje pomažu da se odredi gdje će se objekt u nivou kretati. Na obje je postavljena mogućnost *markup* jezika `EditAnywhere` kako bi im se moglo davati vrijednosti prilikom postavljanja u nivo. Član varijabla `Length` tipa `float` predstavlja udaljenost između `Start` i `End`. Član varijabla `Distance` tipa `float` predstavlja udaljenost koju je objekt prešao u nivou. Varijabla `Direction` služi da bi se mijenjao smjer kretanja objektu u nivou, dok `Speed` predstavlja brzinu kojom će se objekt micati u nivou.

Mogućnost pomicanja je implementirana unutar `Tick` metode koja je pregažena pošto

klasa nasljeđuje `AActor`.

```
FVector newLocation = GetActorLocation();
if (Distance < Length)
{
   newLocation.X += ((this->Start.X - this->End.X) * Speed)
                           * Direction * DeltaTime;
   newLocation.Y += ((this->Start.Y - this->End.Y) * Speed)
                           * Direction * DeltaTime;
   newLocation.Z += ((this->Start.Z - this->End.Z) * Speed)
                           * Direction * DeltaTime;
   Distance += pow(pow((this->Start.X - this->End.X), 2)
               + pow((this->Start.Y - this->End.Y), 2)
               + pow((this->Start.Z- this->End.Z), 2), 0.5)
               * Speed * DeltaTime;
   this->SetActorLocation(newLocation);
}
   else
{
   Direction *= (-1);
   Distance = 0;
}
```

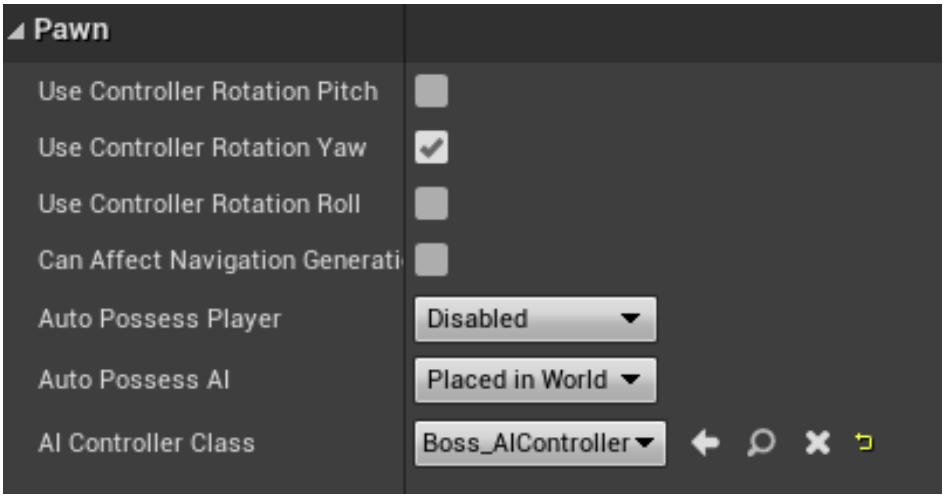
Ispis 13: Pomicanje prepreke

U kôdu 13 se prvo provjeri je li `Distance` manji od `Length`. Drugim riječima to znači da objekt još nije došao do krajnje točke kada bi se trebao početi kretati u suprotnom smjeru. Ako je prošao tu točku `Direction` se mijenja u suprotnu vrijednost, a `Distance` postavi na 0 da krene iznova. Ako objekt treba promijeniti poziciju, prije grananja uzme se trenutno mjesto gdje se nalazi te na koordinate X, Y, Z nadoda izračunate vrijednosti za koliko bi se trebao pomaknuti. Objekt se na kraju postavi na novu lokaciju.

4.5. Neprijatelj

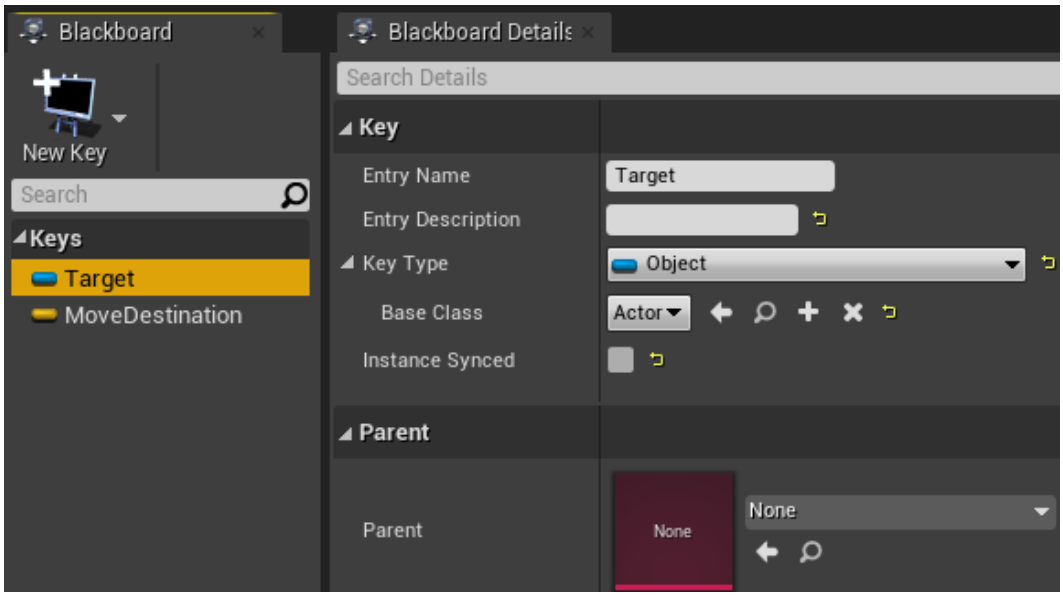
Za zadnji nivo je napravljen neprijatelj koji hvata glavnog lika i udara ga skidajući mu `Health`. C++ klasa `ABoss` sadrži član varijablu `Health` kao i glavni lik te kada neprijatelju padne vrijednost `Health` na 0 igra je završena. `SphereVision` je pokazivač na `USphereComponent` koja služi provjeri postoji li neki objekt unutar radijusa kugle. *Blueprint* klasa `Boss_BP` nasljeđuje klasu `ABoss`. Unutar Unreal Enginea su implementirane funkcionalnosti kako bi se pokazao drugi način izrade igre u ovom razvojnom okruženju.

Unreal Engine nudi pomagalo koje je se zove stablo ponašanja (engl. *behaviour tree*). Stablo ponašanja se oslanja na drugo pomagalo zvano ploča (engl. *blackboard*). Koristeći ploču koja ima ključeve s potrebnim informacijama stablo ponašanja zna donijeti odluke o izvođenju grana te se tako stvara umjetna inteligencija (engl. *artificial intelligence*) za likove koje ne kontrolira igrač. Napravljeno je stablo `Boss_BT`, ploča `Boss_BP` i *blueprint* upravljač (engl. *controller*) `Boss_AI Controller`. Unutar upravljača se napravio događaj koji pokreće stablo ponašanja `Boss_BP`. Da bi ga neprijatelj mogao koristiti u detaljima *blueprint* klase `Boss_BP` se postavi ovaj upravljač kao na slici 31.



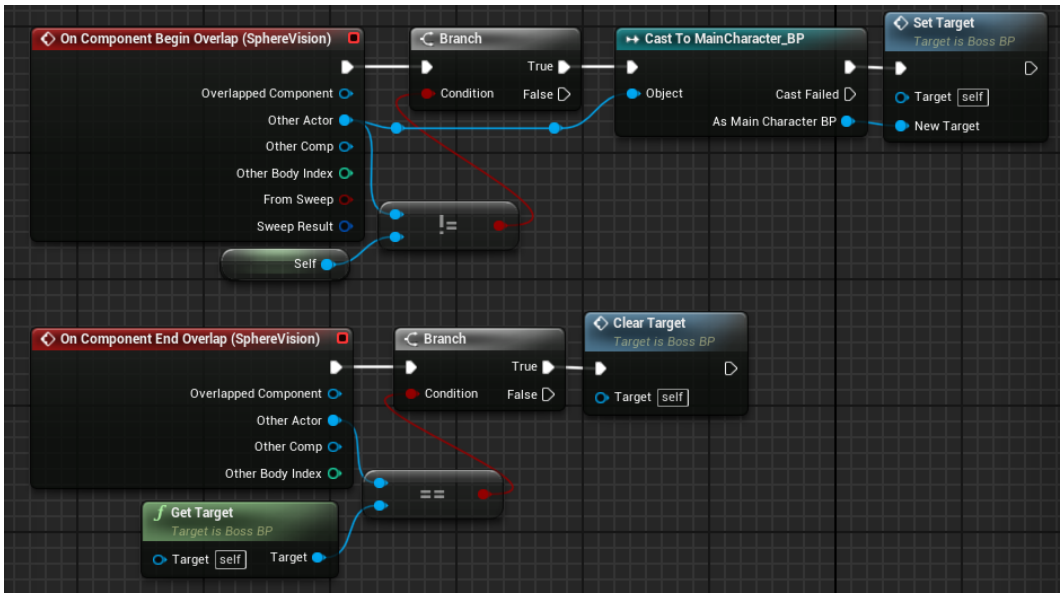
Slika 31: Postavljanje upravljača kod neprijatelja

Na ploči su napravljena dva ključa - `MoveDestination` i `Target`. `Target` će biti lik s kojim se igra kako bi neprijatelj znao koga pratiti i napadati, a vrijednost će se postavljati unutar `Boss_BP`. `MoveDestination` je vektor, a služit će da se lik nasumično pomiče dok nema postavljen `Target`. Izgled ploče i detalji ključa `Target` su prikazani na slici 32



Slika 32: Primjer ploče

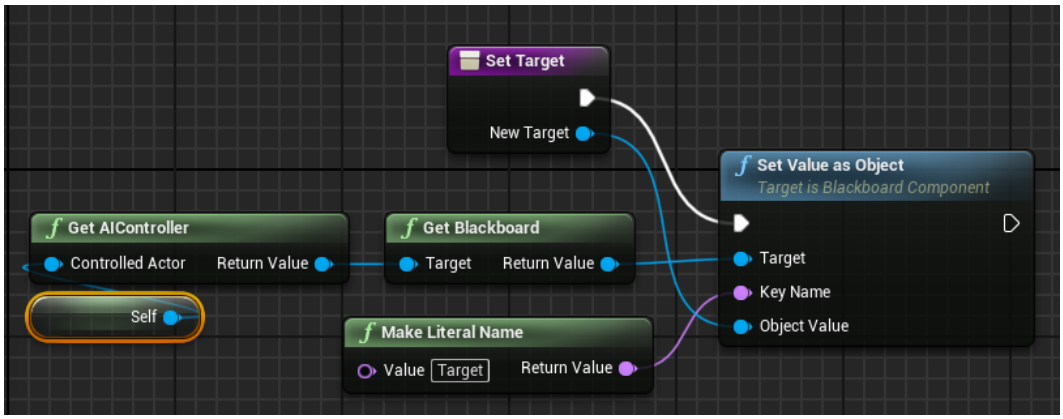
Unutar `Boss_BP` su napravljeni događaji za početak i kraj preklapanja sa `SphereVision`. Događaji imaju istu funkcionalnost kao događaji preklapanja kod glavnog lika samo su ovaj put napravljeni pomoću skriptnog vizualnog jezika kao na slici 33. Prvo se provjeri je li objekt koji se preklapa različit od samog neprijatelja, ako je različit pokuša ga se pretvoriti u pokazivač na `MainCharacter_BP` i ako uspije poziva se funkcija `SetTarget`. Ako neki objekt izađe iz `SphereVision` pokreće se `On Component End Overlap` i ako se utvrdi da je naš lik izašao iz `SphereVision` poziva se funkcija `Clear Target`.



Slika 33: Događaji preklapanja

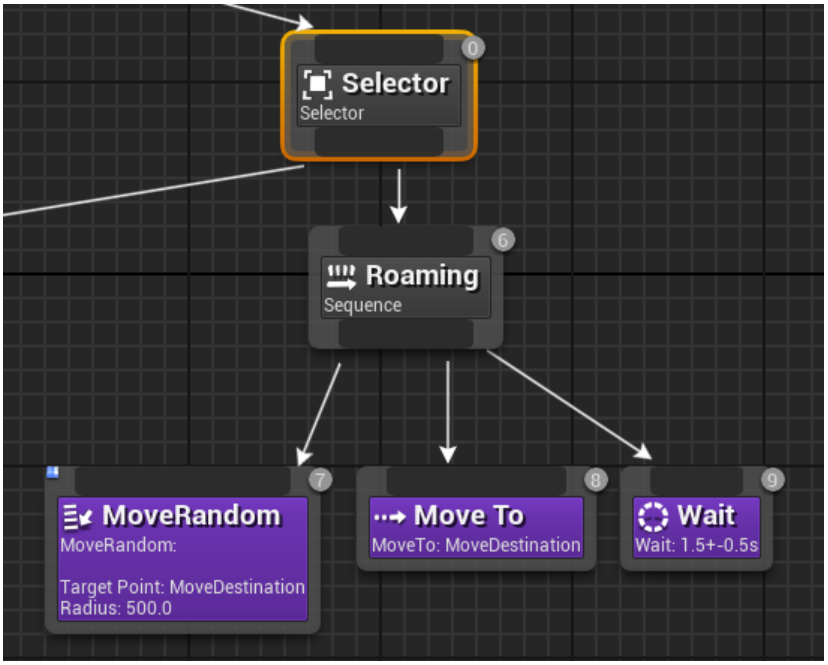
Pomoću funkcije `SetTarget` ključ `Target` s ploče se postavi na vrijednost lika kojim

igrač upravlja. Funkcija `ClearTarget` služi da se ključu `Target` makne postavljena vrijednost, a funkcija `GetTarget` dobavlja vrijednost ključa `Target` s ploče. Sve tri funckije napravljene su skriptnim vizualnim jezikom, a primjer `SetTarget` se nalazi na slici 34. Funkcija prima parametar `NewTarget`, nabavi referencu na ploču i na vrijednost `Target` ključa postavi vrijednost parametra `NewTarget`.



Slika 34: Postavljanje mete

Unutar stabla ponašanja postavimo čvor selektor (engl. *selector*). Unreal Engine kaže da selektor pokreće svoju djecu s lijeva na desno i prestane ih pokretati ako je jedno od djece uspješno u pokretanju. Ako je neko od djece uspješno i selektor je uspješan, ako se niti jedno dijete nije uspjelo pokrenuti selektor je neuspješan.

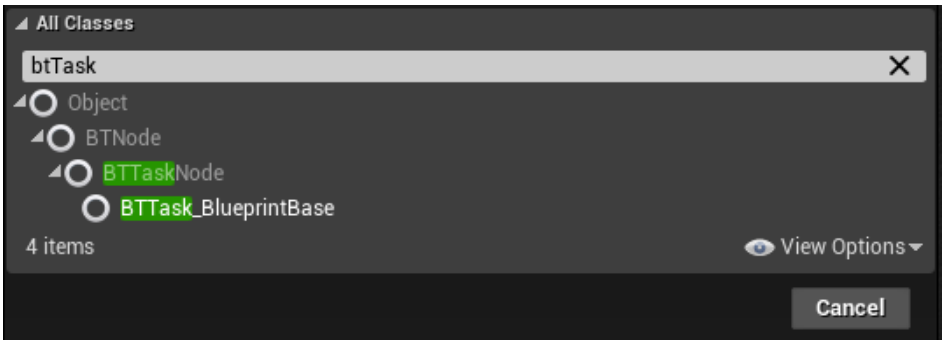


Slika 35: Prikaz stabla ponašanja

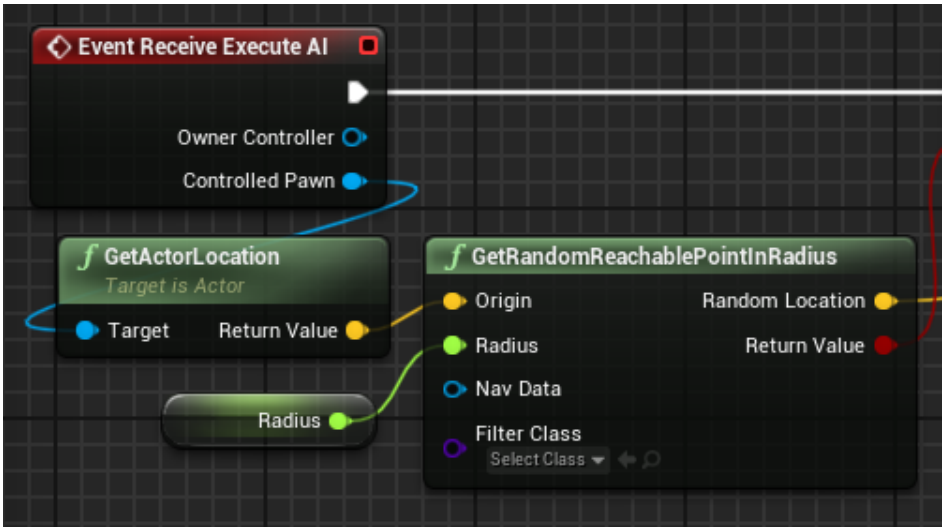
Prvo je napravljeno dijete `Roaming` koje je slijed (engl. *sequence)*. Unreal Engine opisuje slijed kao čvor koji pokreće svoju djecu s lijeva na desno. Prestaje pokretati svoju

djecu čim se jedno ne uspije pokrenuti i tada je slijed neuspješan. Ako su sva djeca uspješno pokrenuta tada je slijed uspješan. Djeca ovog slijeda su zadatci (engl. *tasks*). Iz slike 35 su vidljiva tri zadatka. Zadatci `MoveTo` i `Wait` su dio Unreal Enginea. Zadatak `MoveTo` šalje lika do određene destinacije koja je u ovom slučaju postavljena kao ključ ploče `MoveDestination`, a `Wait` služi da lik napravi zaustavljanje određenog trajanja.

Zadatak `MoveRandom` je implementiran na način da se desnim klikom na pretraživač sadržaja odabere izrada *blueprint* klase. Tu se zatim odabere `BTTask_BlueprintBase` kao na slici 36 i implementacija se odradi pomoću vizualnog skriptnog jezika. Zadatak je nazvan `MoveRandom`. U zadatku su napravljene varijable `TargetPoint` i `Radius`. Prvo se dobavi lokacija lika, koji će u ovom slučaju biti `Boss_BP`. Zatim se pomoću gotove funkcije koju nudi Unreal pokuša pronaći točka unutar radijusa kao na slici 37.



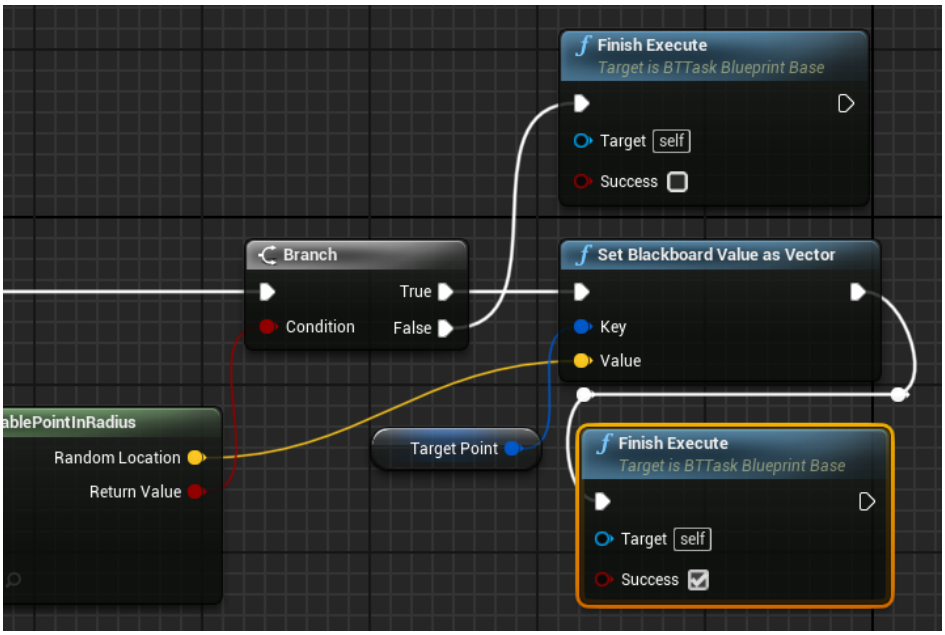
Slika 36: Kreiranje zadatka



Slika 37: Traženje dostupne točke

Funkcija kao povratnu vrijednost vraća istinu ili laž. Njena povratna vrijednost provjerava se u grananju i ako je vrijednost istina, vrijednost točke postavlja se u varijablu preko reference `TargetPoint` i zaustavlja se izvršavanje zadatka s uspješnim rezultatom kao na

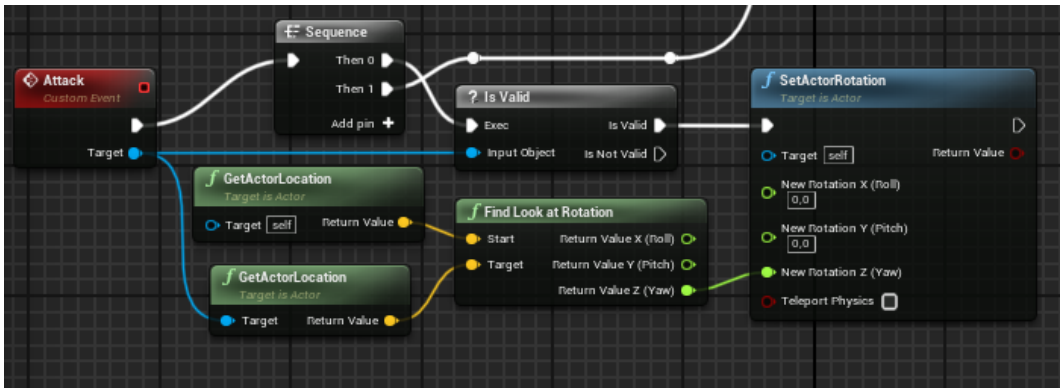
slici 38. Ako je povratna vrijednost laž zadatak se završava s neuspješnim rezultatom.



Slika 38: Drugi dio traženja dostupne točke

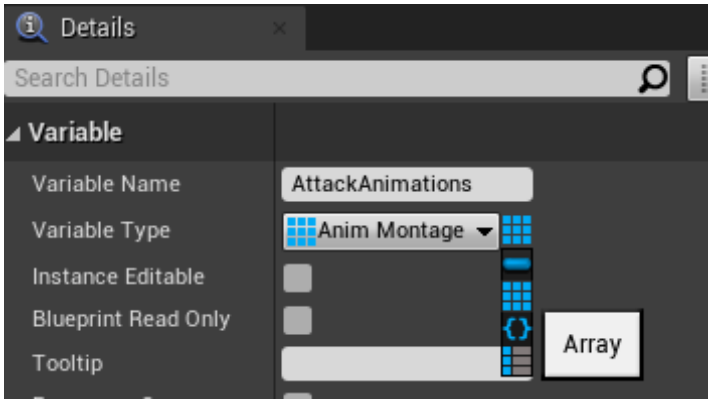
Kada je novi zadatak dodan u stablo ponašanja preko detalja se unese radijus, a kao `TargetPoint` se stavi referenca na `MoveDestianiton` iz ploče. Da bi se neprijatelj mogao kretati u nivo se mora ubaciti `Nav Mesh Bounds Volume` koji računa navigacijske putove kroz područja nivoa.

Za drugi dio stabla ponašanja napravi se novi čvor selektor i doda mu se dekorater, što je u biti provjera uvjeta za izvršavanje tog dijela stabla. Kao uvjet se postavi provjera je li postavljena vrijednost `Target` na ploči. Zatim je dodan zadatak `Move To` samo što ovaj put neprijatelj ide do `Target`. Drugi zadatak je trebalo implementirati, a zove se `RegularAttack`. Tom selektoru je dodan dekorater koji će provjeravati je li neprijatelj dovoljno blizu svoje mete. `BTDecorator_BluepritnBase` je uzeta kao baza za izradu dekoratera. U njemu se dohvati referenca na neprijatelja i s ploče se uzme referenca na glavnog lika. Iz njihovih pozicija se izračuna udaljenost i ako je varijabla `RegularAttackRange` iz `Boss_BP` veća ili jednaka od izračunate udaljenosti vraća se istina, u suprotnom laž. Ako dekorater vrati istinu odrađuje se zadatak `RegularAttack`. Unutar `Boss_BP` je napravljen događaj `Attack` koji će biti pozvan u zadatku `RegularAttack`. Za događaj `Attack` je postavljena sekvenca. Prvi dio sekvence usmjeri neprijatelja prema meti koju udara što je prikazano na slici 39.



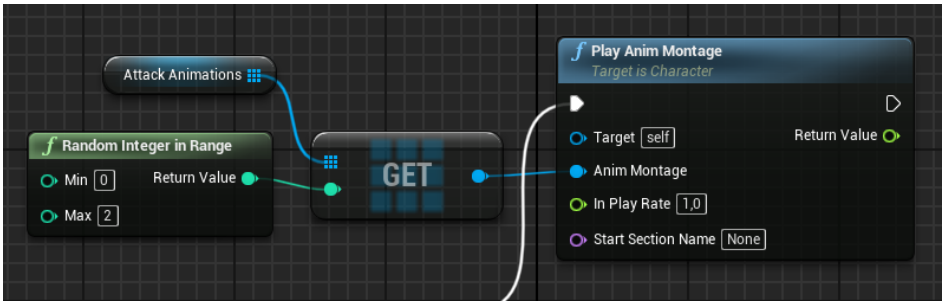
Slika 39: Usmjeravanje neprijatelja prema liku kojeg napada

Za drugi dio sekvence napravljena je varijabla `AttackAnimations` koja je u detaljima postavljena kao niz `Anim Montage` što je prikazano na slici 40.



Slika 40: Postavljanje varijable da bude niz

U niz su ubačene animacije za napad neprijatelja i slučajnim odabirom se pokreće jedna od animacija iz tog niza. Drugi dio sekvence je prikazan na slici 41.



Slika 41: Pokretanje jedne od animacija iz niza

Zadatak `RegularAttack` služi da se pozove događaj `Attack` i odgovori stablu ponašanja je li zadatak odrađen uspješno. U njemu je također dodana vremenska odgoda tako da neprijatelj ne napada previše brzo.

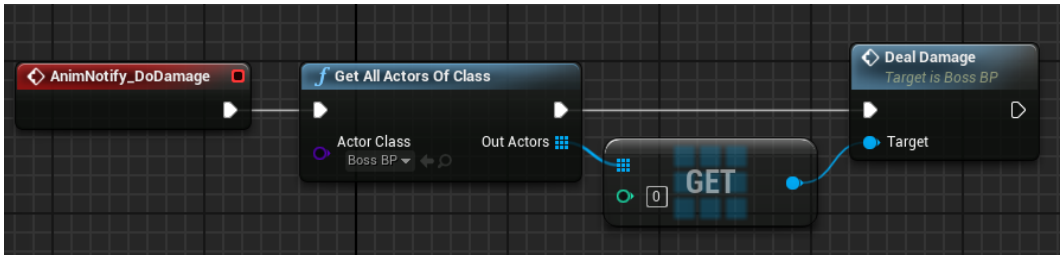
Za skidanje vrijednosti `Health` glavnom liku, napravljen je događaj `DealDamage`. Prvo se pozove funkcija `SphereOverlapActors`. Da bi ona provjerila preklapanje s ku-

glom dodana je varijabla `SphereRadius` i nova komponenta strelica nazvana `DamageArrow`. Kada se funkciji pošalju ti parametri ona na kraju strelice stvori kuglu tog radijusa.

Slika 42: Poziv događaja u određenom trenutku animacije

Funkciji je poslan niz *enuma* koji predstavljaju vrste objekata s kojima će se tražiti preklapanje i niz objekata s kojima će se ignorirati preklapanje. Kad funkcija pronađe preklapanja ulazi se u petlju i provjerava ima li neki od objekata oznaku `Character`. Ako ima, pokuša ga se pretvoriti u referencu na `MainCharacter_BP`. Pomoću te reference pozove se metoda `DamageMe` implementirana u C++ klasi `AMainCharacter`. Funkcija prima broj koji se oduzme od `Health`. Ovaj događaj može biti pozvan na raznim mjestima. S obzirom da postoje različite animacije napada onda se `DealDamage` može

postaviti da se zove u trenutku kada tijekom animacije neprijatelj izgleda kao da je udario ispred sebe. Jedna od animacija u trenutku napada je prikazana na slici 42. Trenutak pozivanja imenovan je `DoDamage`. Da bi se napravile animacije za neprijatelja potreban je `Blend Space 1D` pošto neće biti implementirano šetanje u stranu. Tu je napravljen događaj `AnimNotify_DoDamage` koji će se pokrenuti kada u animaciji dođe do trenutka gdje je postavljen `DoDamage`. Događaj je prikazan na slici 43 i sve što radi je nabavlja referencu na neprijatelja te iz njega zove događaj `DealDamage`.



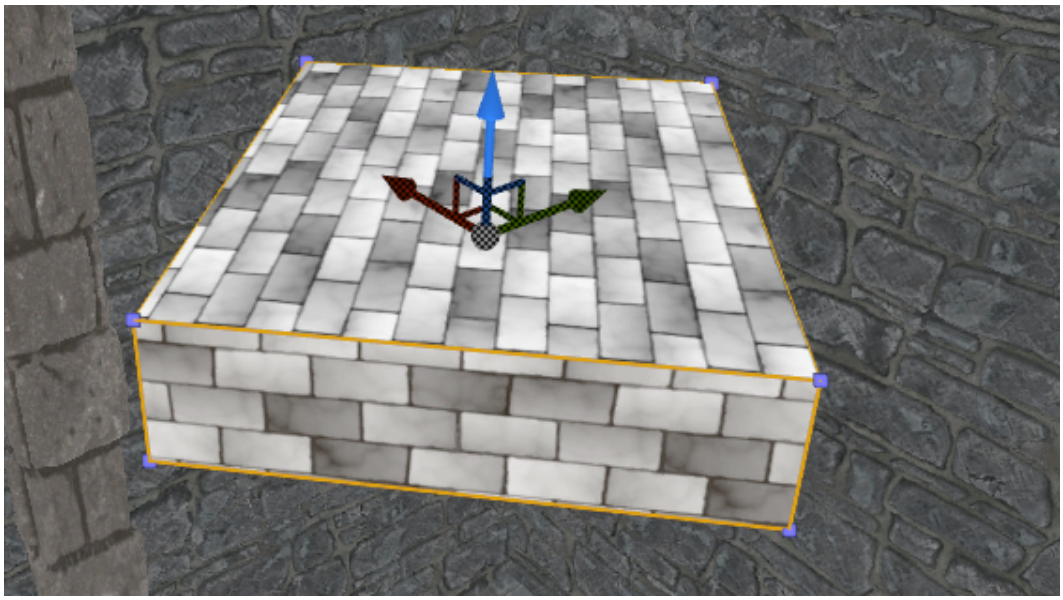
Slika 43: Pozivanje događaja u animaciji

Animacije i konačni automat stanja su napravljeni slično kao za glavnog lika, ali su jednostavniji. Još jedna sličnost je funkcija `Burn`, samo što je ovdje ona napravljena pomoću vizualnog skriptnog jezika.

4.6. Nivo

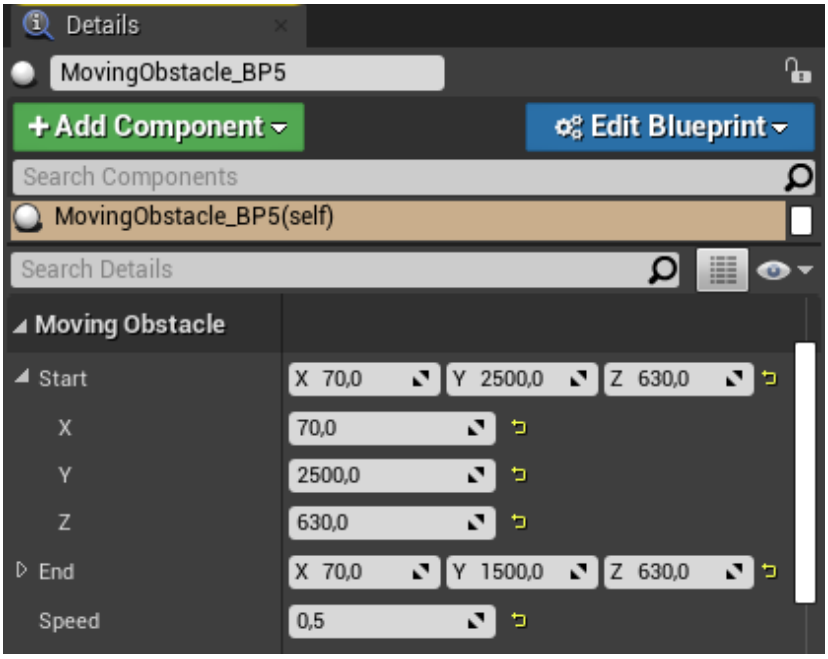
Nivoi predstavljaju mjesto odvijanja radnje. Ovisno od žanra i dostupne tehnologije mogu se razlikovati u veličini, prikazu, mogu biti dvodimenzionalni ili trodimenzionalni. S razvojem video igara i tehnologije postaju sve raznolikiji, veći, ljepši, s puno detalja i novim mogućnostima prilikom igranja. U Unreal Engineu svaki nivo ima svoju *blueprint* klasu u kojoj se definiraju posebna ponašanja unutar nivoa ili uključiti neku glazbu koja će svirati kroz taj nivo.

Nivoi su izrađeni u prozoru za prikaz. U pretraživaču sadržaja pronađe se neki objekt koji treba postaviti u nivo i jednostavno ga se povlačenjem mišom stavi u prozor za prikaz. Klikom na objekt u prozoru za prikaz se pojave strelice za pomicanje objekta kroz prostor kao na slici 44. Osim ove opcije postoji mogućnost rotiranja objekta ili mijenjanja proporcija. Postavljanje koordinata objekta unutar nivoa, njegova rotacija i veličina se mogu mijenjati i u njegovim detaljima.



Slika 44: Pomicanje objekta unutar nivoa

U detaljima se mijenjaju i druga svojstva poput materijala ili član varijabli ako su postavljene da su vidljive i promjenjive unutar C++ klase ili *blueprint* klase. Takav primjer je na slici 45 gdje su vidljive član varijable `Start`, `End` i `Speed` klase `MovingObstacle`.



Slika 45: Primjer mijenjanja vrijednosti varijabli kod postavljanja u nivo

5. Zaključak

Cilj ovog rada je bio prikazati razvoj igre koristeći mogućnosti Unreal Engine razvojnog okruženja. Prikazan je način izrade pišući kôd u C++ programskom jeziku uz korištenje gotovih biblioteka iz pogonskog alata. Osim pisanja kôda prikazan je način implementiranja raznih funkcionalnosti korištenjem vizualnog skriptnog jezika. Oba načina imaju svoje prednosti i svoju svrhu. Poznavanjem mogućnosti koje programski jezik C++ nudi može se paziti na optimizaciju i korištenje resursa. Prednost vizualnog skriptnog jezika je što ima toliko gotovih implementiranih funkcionalnosti da ljudi rade igre i prodaju ih bez programerskog znanja. Također može poslužiti prilikom testiranja jer je brže i lakše, a tester ne treba biti vrhunski poznavatelj napisanog kôda ni dobar programer.

S obzirom da je ipak glavna poanta bila programiranje, izgled likova, okoline te zvuk su uvezeni korištenjem Unreal Engine tržnice. Zbog toga igra nije baš grafički optimizirana, a osim toga za ovakvu kvalitetu dodataka bilo bi potrebno uložiti jako puno vremena. Svi dodatci su bili besplatni.

Izrada igre je zahtjevan posao, jer se treba razmišljati o priči, stvaranju ugođaja, mogućnostima koje će zaokupiti igrača i zabaviti ga. Što se tiče tehničke strane razvojno okruženje Unreal Engine tu mnogo pomaže jer razvoj ne treba krenuti od nule. Korištenjem postojećih biblioteka može se fokusirati na implementaciju vlastitih ideja.

Literatura

[1] L. zavod Miroslav Krleža, "Hrvatska enciklopedija, mrežno izdanje," http://www.enciklopedija.hr/Natuknica.aspx?ID=26978 (posjećeno 4.9.2022.), 2021.

[2] "Službena stranica unreal engine," https://www.unrealengine.com (posjećeno 1.9.2022.).

[3] "Unreal engine 4.27 documentation," https://docs.unrealengine.com/4.27/en-US/ (posjećeno 1.9.2022.).