SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Jerko Bućan

Z A V R Š N I R A D

Izrada 3D videoigre pomoću Unreal Enginea 5

Split, rujan 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Objektno programiranje

Z A V R Š N I R A D

Kandidat: Jerko Bućan

Naslov rada: Izrada 3D videoigre pomoću Unreal Enginea 5

Mentor: Ljiljana Despalatović, viši predavač

Split, rujan 2023.

SADRŽAJ

SAŽETAK

Završni rad pod nazivom izrada 3D videoigre u Unreal Engineu 5 obuhvaća cjelokupno istraživanje, razvoj i implementaciju 3D puzzle igre korištenjem Unreal Enginea 5. U radu se detaljno opisuju ciljevi i svrha igre, osnovni pregled strukture rada te ključni aspekti istraživanja i razvojnog procesa.

Rad se temelji na primjeni objektno orijentiranog programiranja i vizualnog programiranja kroz nacrtni sustav, kao i korištenju jezika C++ za razvoj kompleksnih komponenti igre. Unreal Engine 5 je odabran kao glavni alat za razvoj igre zbog svoje moćne i napredne funkcionalnosti, podrške za različite platforme te mogućnosti integracije s raznim alatima.

U radu se detaljno opisuje kreiranje projekta, razvoj okoline i karaktera, implementacija igračkih mehanika i interakcija, te dodavanje novih komponenti i zagonetki na višim razinama igre.

Kroz sadržaj ovog završnog rada steći će se dublji uvid u proces razvoja 3D igre unutar Unreal Enginea 5, istražiti će se ključni elementi objektno orijentiranog i vizualnog programiranja te primjene jezika C++ u stvaranju igara. Također, istaknuta je važnost Unreal Enginea 5 kao močnog i fleksibilnog alata za izradu visokokvalitetnih 3D igara.

Ključne riječi: 3D puzzle igra, C++, Unreal Engine 5, vizualno skriptiranje.

SUMMARY

Creating a 3D video game using Unreal Engine 5

The final paper entitled making a 3D video game in Unreal Engine 5 includes the entire research, development and implementation of a 3D puzzle game using Unreal Engine 5. The paper describes in detail the goals and purpose of the game, a basic overview of the work structure and key aspects of the research and development process.

The work is based on the application of object-oriented programming and visual programming through the blueprints, as well as the use of the C++ for the development of complex game components. Unreal Engine 5 was chosen as the main tool for game development due to its powerful and advanced functionality, cross-platform support and the ability to integrate with various tools.

The paper describes in detail the creation of the project, the development of the environment and characters, the implementation of game mechanics and interactions, and the addition of new components and puzzles at higher levels of the game.

Through the content of this final paper, a deeper insight into the process of 3D game development within Unreal Engine 5 will be gained, the key elements of object-oriented and visual programming and the application of the C++ language in game creation will be explored. Also, the importance of Unreal Engine 5 as a powerful and flexible tool for creating high-quality 3D games was highlighted.

Keywords: 3D puzzle game, C++, Unreal Engine 5, visual scripting.

1. UVOD

Razvoj video igara predstavlja intrigantno polje koje se neprestano širi i evoluira zahvaljujući tehnološkom napretku i rastućem interesu igrača diljem svijeta. Unutar ovog kreativnog okruženja, Unreal Engine 5 se izdvaja kao jedan od najnaprednijih i najmoćnijih alata za stvaranje 3D igara. Ovaj završni rad pruža uvid u proces razvoja igre koristeći Unreal Engine 5, istražujući specifičnosti, metode i alate koji su korišteni u tom procesu.

Cilj ovog rada bio je istražiti mogućnosti Unreal Enginea 5 i stvoriti 3D igru koja će pružiti jedinstveno i zadovoljavajuće iskustvo igračima. Ova igra je razvijena s naglaskom na raznovrsnost razina i mehanika, vizualne efekte i pažljivo osmišljen zvučni dizajn kako bi se stvorila imerzivna igraća okolina igrača. Istovremeno, kroz razvoj ove igre, nastojalo se razumjeti izazove s kojima se susreću razvojni timovi te pronaći optimalna rješenja za njihovo prevladavanje.

Sljedeće poglavlje se usredotočilo na specifikaciju igre, opisivanje detaljne vizije igre, njene glavne mehanike i ciljanu publiku. Zatim će slijediti poglavlje o metodama i alatima korištenima tijekom razvojnog procesa, gdje su raspravljani ključni koraci u stvaranju igre i kako su odabrani alati doprinijeli uspješnom ostvarenju ciljeva.

Dodaci, animacije i zvukovi su besplatno preuzete sa stranice `https://kenney.nl/`, koji su dodatno obrađeni radi potrebe projekta. Izrada projekta provedena je uz pomoć vodiča s `https://www.youtube.com` i službene dokumentacije Unreal Enginea, što je služilo kao relevantan izvor informacija i pomoći prilikom razvoja igre.

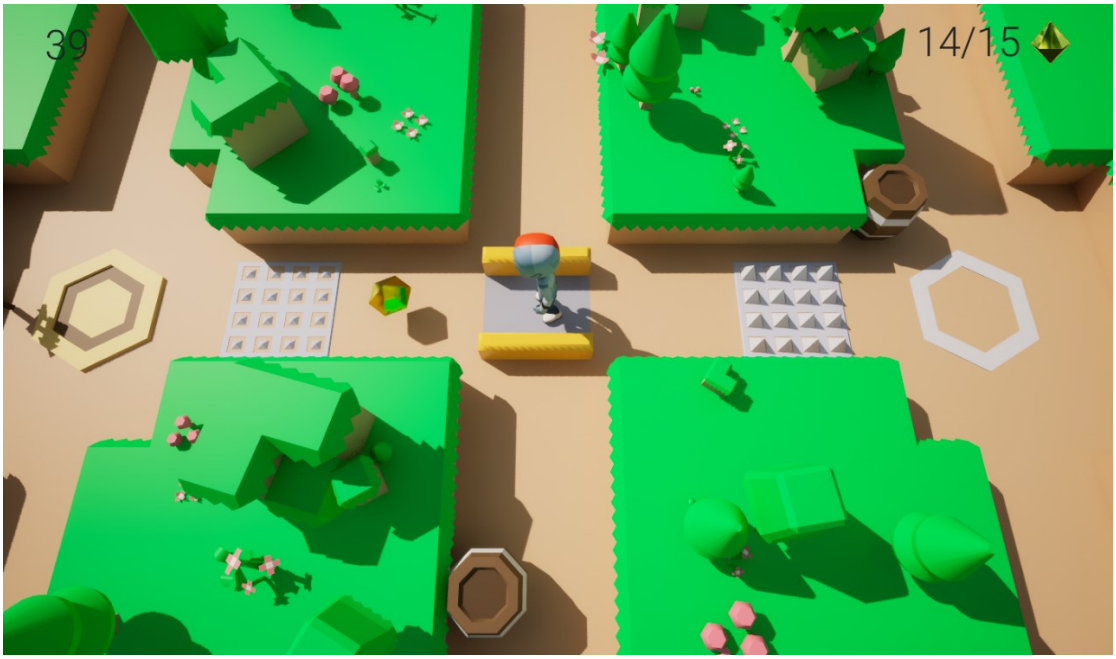
2. SPECIFIKACIJA IGRE

Žanr igre: 3D puzzle avantura

Ciljana platforma: PC

Ciljna publika: Igrači svih dobnih skupina koji uživaju u izazovnim mozgalicama i avanturama.

Opis igre: Puzzle igra koja stavlja igrača u šareni i čarobni svijet ispunjen zagonetkama (Slika 1). Igrač će preuzeti ulogu hrabrog istraživača koji se suočava s nizom izazova dok putuje kroz različite razine, svaka teža od prethodne. Svaka razina u igri predstavlja jedinstvenu zagonetku koju igrač mora riješiti kako bi napredovao na sljedeću razinu.



Slika 1: Prikaz videoigre

Glavne mehanike igre

1. **Sakupljanje predmeta:** Glavni cilj svake razine je sakupiti sve zlatne predmete kako bi se otključao put prema sljedećoj razini. Igrač mora pažljivo istraživati okolinu i koristiti logično razmišljanje.

2. **Puzzle mehanike:** Na svakoj drugoj razini uvode se nove mehaničke zagonetke koje igrač mora savladati kako bi riješio izazov i prikupio potrebne predmete. Ove mehanike mogu uključivati rotaciju platformi, upravljanje platformama putem botuna te korištenje lokota i ključeva kao elementa zagonetke.
3. **Istraživanje i avantura:** Igrač će se suočiti s raznolikim izazovima u svakoj razini, što će zahtijevati istraživanje okoline i rješavanje logičkih enigmi.

Napredovanje kroz igru: Igrač će započeti na početnoj razini koja će služiti kako bi se upoznao s osnovnim mehanikama igre. Nakon toga, igrač će prelaziti kroz različite razine, od kojih će svaka biti teža od prethodne. Svaka druga razina donosi nove mehaničke zagonetke kako bi se osvježilo iskustvo i izazvalo igrača na nov način.

Vizualni stil: Grafički stil igre bit će šaren i privlačan, s prekrasnim 3D okolinama i detaljno dizajniranim karakterom. Svaka razina imat će jedinstvenu temu.

Zvučni dizajn i glazba: Zvučni dizajn igre pažljivo je osmišljen kako bi stvorio dojmljivu atmosferu. Zvukovi okoline, zvukovi mehaničkih zagonetki i prateća glazba doprinijet će uranjanju igrača u čarobni svijet igre.

3. METODE I ALATI

3.1. C++

Jezik C++ je objektno orijentirani i višestruko primjenjiv programski jezik koji pruža programerima moćne alate za razvoj softvera na raznim platformama. Osim što je popularan izbor za razvoj računalnih igara, jezik C++ se koristi i u područjima poput operativnih sustava, aplikacija u stvarnom vremenu, mobilnih aplikacija, mikro kontrolera, web preglednika i još mnogo toga.

Jedna od ključnih prednosti jezika C++-a je mogućnost kombiniranja visokog i niskog nivoa programiranja. Na visokom nivou, programeri mogu koristiti objektno orijentiranu paradigmu za organizaciju kôda i olakšavanje ponovnog korištenja. S druge strane, niski nivo omogućuje pristup hardverskim resursima i upravljanje memorijom, čime se postiže veća kontrola nad performansama i resursima sistema.

Jezik C++ također omogućuje direktan rad s memorijom pomoću pokazivača, što može biti korisno za optimizaciju i pristupanje hardverskim resursima, ali zahtijeva odgovornost u upravljanju memorijom kako bi se izbjegle greške poput curenja memorije. Također, jezik C++ podržava operator overloading, omogućuje da se definiraju specifične operacije za korisnički definirane tipove podataka, što olakšava rad s kompleksnim objektima. Osim toga, jezik C++ nudi bogatu standardnu biblioteku, uključujući Standard Template Library (STL), koja sadrži algoritme, kontejnere i funkcije za olakšavanje rada s različitim strukturama podataka kao što su podaci, stringovi i vektori.

3.2. Unreal Engine 5

Unreal Engine, razvijen od strane tvrtke Epic Games pod vodstvom Tim Sweeney od 1995. godine, predstavlja jedan od najmoćnijih i najpopularnijih game enginea na tržištu. Ovaj iznimno moćan alat koristi se za stvaranje širokog spektra interaktivnih iskustava, uključujući video igre, virtualnu stvarnost (engl. *virtual reality „VR“*), proširenu stvarnost (engl. *augmented reality „AR“*), simulacije, obuku te mnoge druge aplikacije. Kroz godine, Unreal Engine je prošao brojne nadogradnje i evolucije, a najnovija verzija, Unreal Engine

5, najavljena je 2020. godine, donoseći revolucionarne tehnologije poput Nanite i Lumen koje omogućuju ultra-realističan vizualni doživljaj.

Unreal Engine pruža dvije glavne mogućnosti za programiranje igara, a to je vizualno programiranje putem nacrta (engl. *blueprints*) i programiranje u jeziku C++. Blueprints omogućuju brzo i jednostavno stvaranje igračkih mehanika bez potrebe za kodiranjem, dok programiranje u jeziku C++ pruža veću fleksibilnost i snagu za razvoj kompleksnih igara.

Unreal Engine je poznat po svojoj iznimnoj vizualnoj kvaliteti i podršci za foto realistične grafike. S tehnologijama poput fizički zasnovanog renderinga (engl. *Physically-Based Rendering „PBR"),* visokokvalitetnih materijala, globalnog osvjetljenja (Lumen) i naprednih efekata, Unreal Engine omogućuje stvaranje živopisnih i impresivnih svjetova.

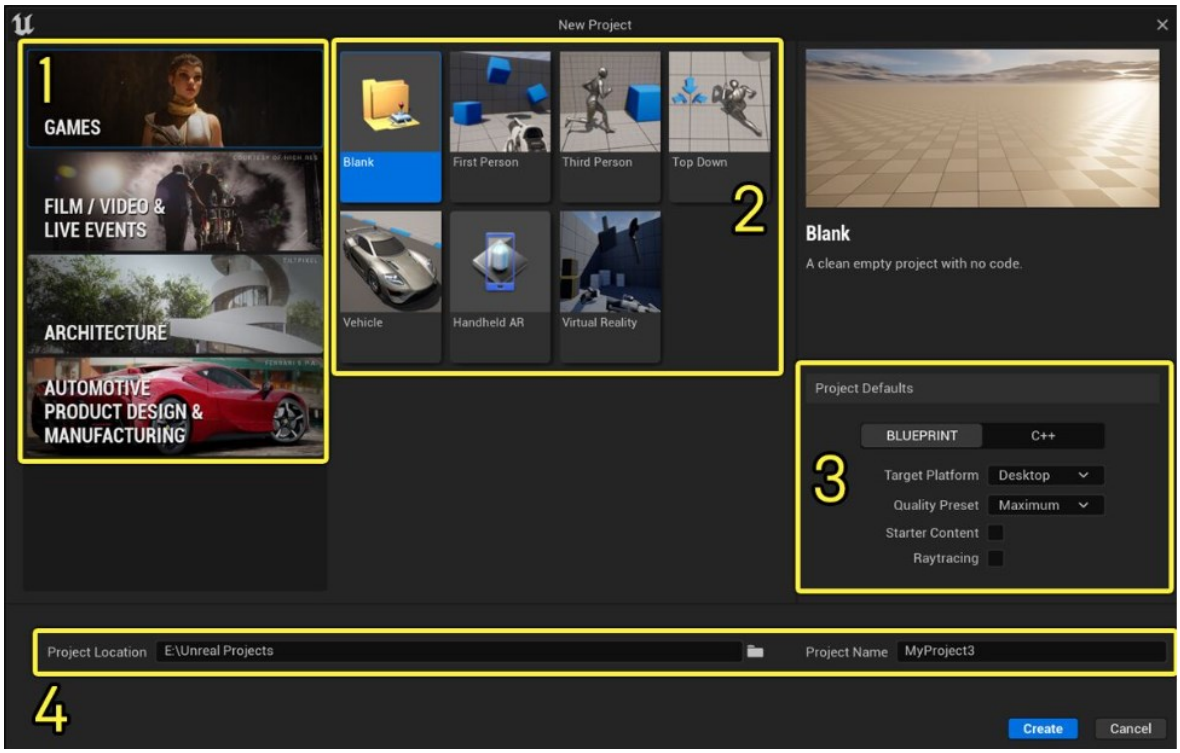
Podržava različite platforme, uključujući PC, konzole (PlayStation, Xbox), mobilne uređaje (iOS, Android), VR i AR uređaje. To omogućuje razvoj i distribuciju igara i aplikacija za široku publiku.

U ovom slučaju, koristio se UE5 u kombinaciji s programskim jezikom C++ za razvoj klasa i funkcionalnosti.

3.2.1. Kreiranje novog projekta

Kreiranje novog projekta u Unreal Engineu može biti pojednostavljen postupak, a ovdje je detaljan opis kako to može biti ostvareno. Ukoliko on već nije instaliran, moguće je preuzeti ga s službene web stranice Epic Games.

Kada se program pokrene, otvara se preglednik (engl. *Unreal Project Browser*) (Slika 2), može se odabrati željena verzija Unreal Enginea (ako ih je više instalirano) i predložak za novi projekt. Unreal Engine nudi različite kategorije i predloške za različite vrste projekata.



Slika 2: Unreal projektni preglednik

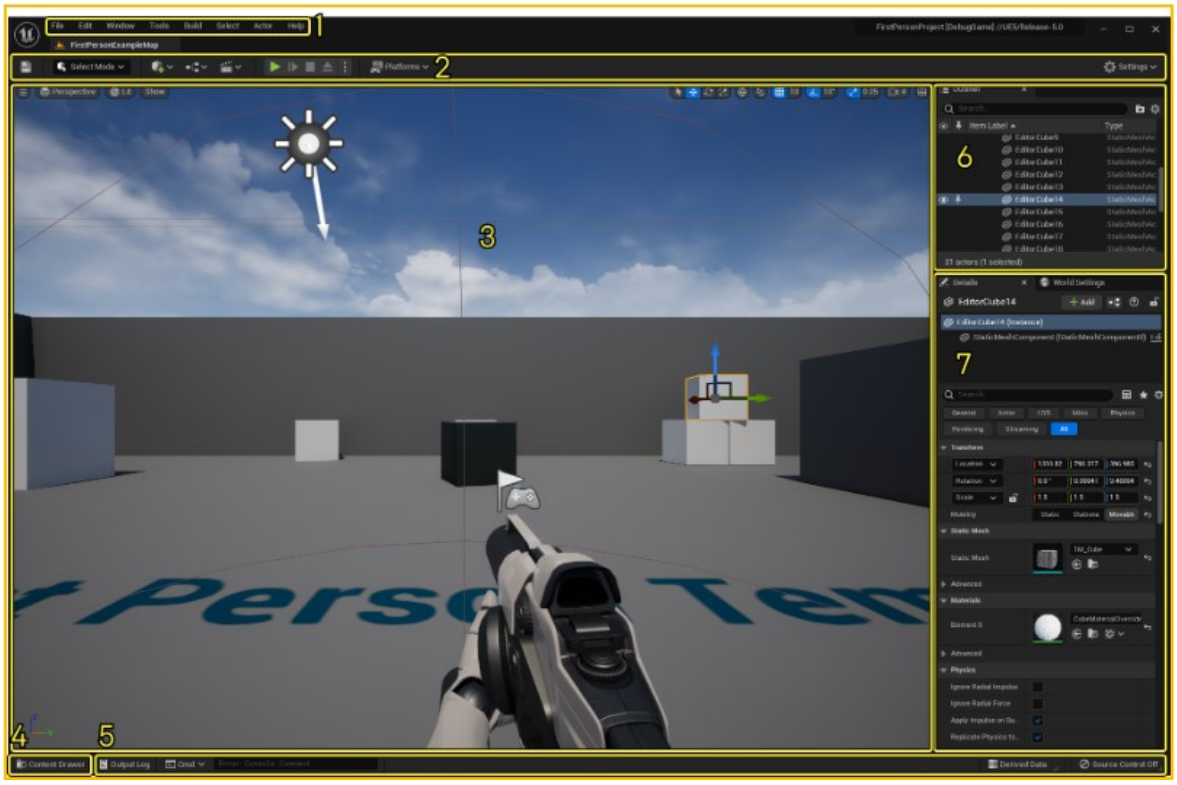
Na lijevoj strani označeno brojem 1 na slici 2 se odabire kategorija, u ovom slučaju igre (engl. *Games*), nakon toga prikažu se predlošci (Slika 2, broj 2) kao što su pogled iz prvog lica (engl. *First Person*), trećeg lica (engl. *Third Person*), prazan projekt (engl. *Blank*) i druge.

Ovdje odabiremo prazan projekt te se u zadanim postavkama projekta (engl. *Project Defaults*) (Slika 2, broj 3) omogućuje odabir ciljne platforme (engl. *Target Platform*) za igru ili aplikaciju (tj. hardver na kojem će se izvoditi, poput računala ili mobilnog uređaja), konfiguriranje postavki kvalitete (engl. *Quality Preset*) i praćenja zraka (engl. *Ray tracing*) te pruža mnoge druge opcije za prilagodbu.

Dolje na dnu ovog preglednika (Slika 2, broj 4) se odabire gdje spremiti projekt te sam naziv projekta. Nakon što je sve odabrano od opcija, klikom na botun stvaranje (engl. *Create*) kako započeti proces kreiranja projekta.

3.2.2. Upoznavanje sa sučeljem

Prije početka rada u Unreal Engineu, važno je biti upoznat s razvojnim sučeljem. Na glavnom ekranu (Slika 3) omogućeno je prilagođavanje sučelja prema vlastitim preferencama, što dopušta optimalno organiziranje radnih prozora i povećava učinkovitost razvojnog procesa. U tablici 1 opisano je detaljno objašnjenje slike 3.



Slika 3: Glavni zaslon

Tablica 1: Objašnjenje brojeva na slici 3

Broj	Naziv	Opis
1	Traka izbornika	Izbornike za pristup naredbama i funkcionalnostima specifičnim za uređivač.
2	Alatna traka	Sadrži prečace za neke od najčešćih alata.
3	Prikaz scene	Prikazuje sadržaj razine, poput kamera, rotora, statičkih mreža i tako dalje.
4	Pregled sadržaja	Svi elementi koji se mogu koristiti unutar projekta

5	Donja traka za alate	Sadrži prečace za naredbenu konzolu, izlazni dnevnik i funkcionalnost izvedenih podataka. Također prikazuje status kontrole izvora.
6	Hijerarhijski prikaz	Prikazuje hijerarhijski prikaz stabala svih sadržaja na vašoj razini.
7	Detaljni prikaz	Prikazuje različite postavke ovisno o tome što ste odabrali u prikazu scene razine.

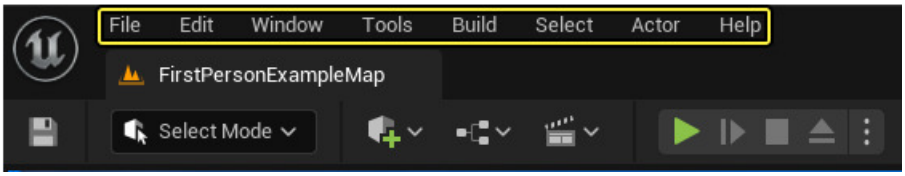
Prilagodba glavnog zaslona u Unreal Engine:

- Pomicanje i grupiranje prozora: Prozori se mogu pomicati i postavljati jedan uz drugi ili na različite strane zaslona kako bi se stvorio raspored koji najbolje odgovara. Također, slični prozori mogu se grupirati zajedno pomoću tabova kako bi se lakše prelazilo između različitih konfiguracija.
- Dodavanje novih prozora: Unreal Engine 5 dopušta dodavanje novih prozora prema potrebama korisnika. Na primjer, može se dodati prozor za Blueprint Editor ako se želi programirati igra kroz vizualno skriptiranje.
- Uklanjanje nepotrebnih prozora: Prozore koji nisu potrebni moguće je ukloniti kako bi se oslobodio viši prostor za prozore koji su važniji.
- Spremanje prilagođenih rasporeda: Kada se postigne željeni raspored prozora, taj raspored se može spremiti kao prilagođeni radni prostor kako bi se lako ponovno koristio u budućnosti.

Prilagodba glavnog zaslona u Unreal Engineu 5 omogućena je za stvaranje radnog okruženja prilagođenog radnim navikama i preferencijama, čime se olakšava razvojni proces i omogućava fokusiranje na kreiranje izvrsnih igara ili virtualnih iskustava.

3.2.3. Glavni prozori

Svaki urednik ima traku izbornika (Slika 4) koji se nalazi u gornjem desnom prozoru tog uređivača. Neki od izbornika, kao što su Datoteka, Prozor, i Pomoć, prisutni su u svim prozorima urednika, a ne samo u uređivaču razine. Ostali su specifični za urednika.



Slika 4: Traka izbornika

Alatna traka (Slika 5) sadrži prečace za neke od najčešće korištenih alata i naredbi u ovom uređivaču. Podijeljen je na sljedeća područja:

1. Botun za spremanje razine koja je trenutno otvorena,
2. Sadrži prečace za brzo prebacivanje između različitih načina uređivanja sadržaja unutar razine.
3. Sadrži prečace za dodavanje i otvaranje uobičajenih vrsta sadržaja unutar uređivača razine (nacrti, kino sekvenca,…).
4. Sadrži botune za prečac (igranje, preskakanje, zaustavljanje i izbacivanje) za pokretanje vaše igre u uredniku.
5. Sadrži niz opcija za konfiguriranje, pripremu i raspoređivanje projekta na različite platforme, poput radne površine, mobilne ili konzole.
6. Sadrži različite postavke za uređivača, pregled razine i ponašanje u igri.

Slika 5: Alatna traka

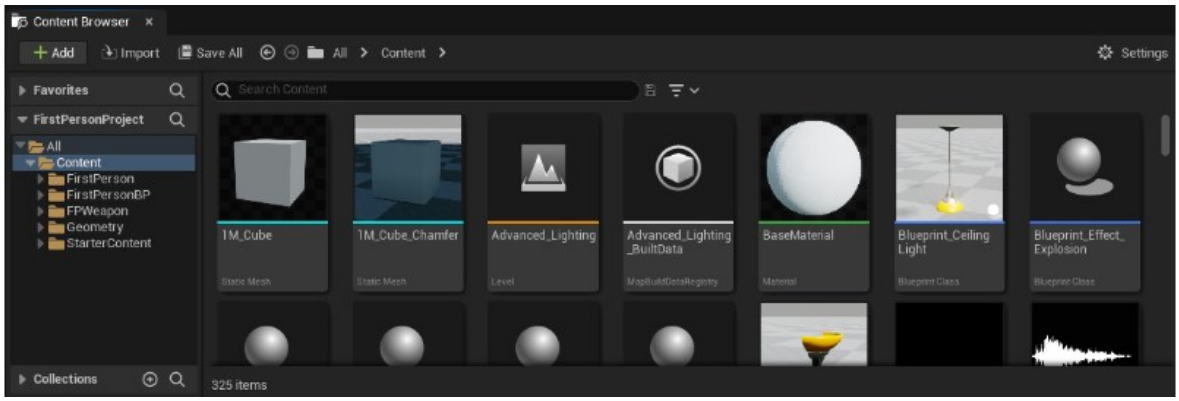
Donja traka za alate (Slika 6) sadrži prečace do naredbene konzole, izlaz dnevnik i funkcionalnost izvedenih podataka. Također prikazuje status kontrole izvora. Podijeljen je na sljedeća područja:

1. Izlazni dnevnik za uklanjanje pogrešaka koji ispisuje korisne informacije dok se aplikacija pokreće.
2. Naredbena konzola se ponaša kao bilo koje drugo sučelje naredbenog retka: unesite naredbe konzole kako biste pokrenuli određeno ponašanje urednika.
3. Izvedeni podaci koji pružaju funkcionalnost podataka.

4. Prikazuje status kontrole izvora ako je vaš projekt povezan s kontrolom izvora (na primjer, GitHub ili Perforce).

Slika 6: Donja traka za alate

Pregled sadržaja (Slika 7) je prozor za istraživanje datoteka koji prikazuje sve elemente koji se mogu koristiti unutar projekta, uključujući modele, skripte, nacrti, audio datoteke, slike i druge resurse.



Slika 7: Prozor pregleda sadržaja

Modeli su 3D objekti koji čine sastavni dio igre. Unutar Unreal Enginea, modeli se često koriste u FBX formatu datoteka, i mogu predstavljati likove, objekte, okolinu i sve ostalo što vidimo u igri.

Unreal Engine podržava različite formate audio datoteka kao što su WAV, MP3 i OGG. Ove audio datoteke koriste se za zvukove u igri, uključujući glazbu, efekte zvuka i dijaloge likova. Slike u različitim formatima (BMP, TIF, JPG, PNG itd.) koriste se za teksture površina i grafiku u igri.

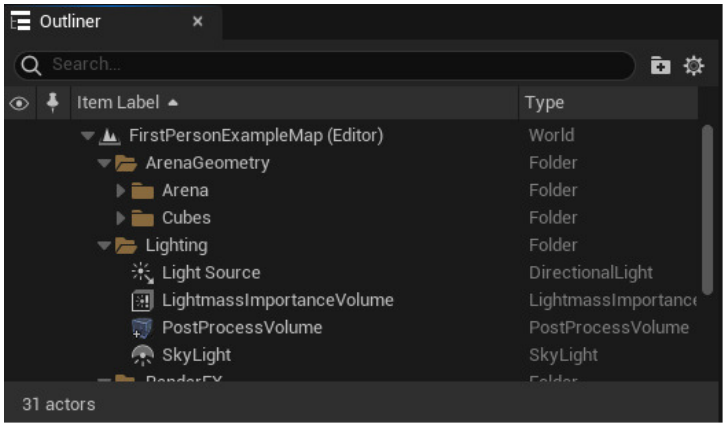
Animatori i dizajneri mogu stvarati kompleksne animacije za likove ili objekte unutar igre pomoću različitih animacijskih alata unutar Unreal Enginea. Materijali definiraju izgled površina i tekstura u igri. Koristeći grafički sučelje, moguće je stvoriti bogate i detaljne

materijale koji će oplemeniti izgled igre. Također ima moćan sustav čestica koji se koristi za stvaranje impresivnih efekata, poput vatre, dima, eksplozija i magije. Nudi razne vrste svjetala i specijalnih efekata koji poboljšavaju atmosferu i vizualni dojam igre.

Nacrti su vizualni sustav skriptiranja unutar Unreal Enginea koji omogućuje izradu interaktivnosti i logike u igri bez potrebe za programiranjem. Pomoću nacrti klasa moguće je definirati nove tipove objekata i likova s unaprijed definiranim ponašanjem i karakteristikama.

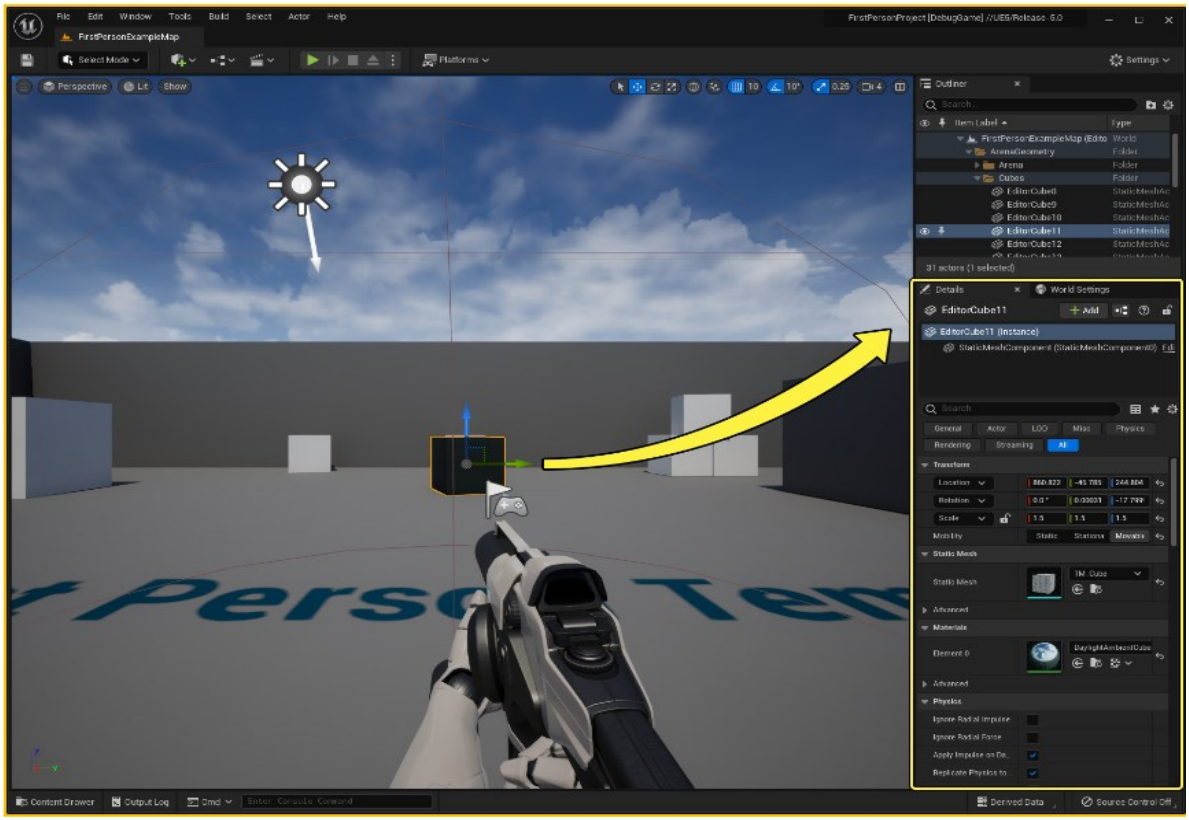
Unreal Engine omogućuje izradu kompleksnih razina i svjetova pomoću intuitivnog uređivača nivoa. Isto tako, na Epic Gamesovoj trgovini (Unreal Engine Marketplace) može se pronaći razne dodatke, poput gotovih modela, tekstura, materijala, kompletnih igara, alata i skripti koje olakšavaju razvoj igre.

Hijerarhijski prikaz (Slika 8) prikazuje sav sadržaj na razini korisnika. Prema zadanom, smješten je u gornjem desnom kutu prozora uređivača. Koristi se za brzo skrivanje ili otkrivanje objekata klikom na pripadajuće botune, omogućava pristup glumačkom izborniku desnim klikom na glumca, što tada omogućava izvođenje dodatnih, specifičnih radnji s glumcem, kao i stvaranje, premještanje i brisanje sadržaja na mapi.



Slika 8: Hijerarhijski prikaz sadržaja na trenutnoj razini

Kada je glumac na sceni odabran, detaljni prikaz (Slika 9) će prikazati postavke i svojstva koja utječu na tog odabranog glumca. Prema zadanom rasporedu, smješten je na desnoj strani prozora uređivača, ispod hijerarhijskog prikaza.



Slika 9: Detaljni prikaz odabranog glumca na sceni razine

3.3. Blender

Blender je imao ključnu ulogu kao osnovni alat za 3D modeliranje i animaciju unutar igre. Kroz Blender su uređivani već postojeći 3D objekti, zajedno s animacijama za likove. Modeliranje i animacije stvorene u Blenderu doprinijele su kreativnosti i postizanju realizma u dizajnu same igre.

3.4. JetBrains Rider

JetBrains Rider je integrirano razvojno okruženje (IDE) specijalizirano za razvoj aplikacija u programskom jeziku C# i .NET platformi. Razvijen od strane JetBrainsa, poznatog po svojim visokokvalitetnim alatima za programiranje, Rider nudi napredne značajke i intuitivan radni okvir za programere koji rade na .NET projektima.

Rider podržava programiranje u jeziku C++, što je jedan od ključnih jezika korištenih za razvoj igara u Unreal Engineu. To omogućuje programerima da koriste sve

mogućnosti jezika C++ i pruža im moćan alat za rad s Unreal Engine kôdom. Sadrži sve potrebne alate za razvoj, uključujući tekstualni uređivač s naglašavanjem sintakse, alate za debugiranje, alat za upravljanje paketima, testiranje i još mnogo toga, što olakšava produktivnost i učinkovitost. Također, koristi snažne algoritme analize kôda kako bi pružio inteligentne sugestije, ispravke i automatsko dovršavanje kôda.

Rider pruža napredne alate za debugiranje u stvarnom vremenu i profiliranje performansi, što omogućuje identifikaciju potencijalnih uskih grla i optimizaciju rada aplikacije. Osim toga, nudi raznovrsne mogućnosti kao što su refaktoriranje, izdvajanje metoda, uklanjanje dupliciranog koda i mnoge druge, što olakšava održavanje i poboljšavanje kvalitete kôda. Rider je dostupan na različitim operativnim sustavima, uključujući Windows, macOS i Linux, omogućavajući programerima da koriste isto integrirano razvojno okruženje na različitim platformama.

4. IMPLEMENTACIJA IGRE

Trenutačna faza razvoja usmjerena je na opis igre koja obuhvaća osam nivoa. U ulozi ženskog lika, igrač prikuplja zlatne predmete kako bi napredovao na sljedeću razinu. Na svakoj drugoj završenoj razini uvodi se nova mehanika zagonetki koja dodatno komplicira prolazak kroz nivo. Igrač se suočava s različitim preprekama koje zahtijevaju logičko razmišljanje kako bi ih prevladao, otključao ili rotirao objekte. Cilj igre je pružiti igračima uzbudljivo iskustvo putem rješavanja zagonetki i suočavanja s izazovima.

Razvoj igre je podijeljen u nekoliko faza:

1. Klasa za lika,
2. Klasa za GameMode i Widgete,
3. Puzzle Mehanike

4.1. Skripta za glavnog lika

U ovom radu, razvoj funkcionalnosti lika za 3D puzzle igru izvršen je putem jezika C++ klase `AGameProjectCharacter` i njene pripadajuće nacrt klase `BP_Character` gdje nasljeđuje osnovnu klasu `ACharacter` iz Unreal Enginea, što mu omogućuje korištenje standardnih mogućnosti za kontrolu lika u igri.

Unutar konstruktor metode, klase `AGameProjectCharacter` izvršena je inicijalizacija ključnih podatkovnih članova lika, status različitih ključeva, vrijeme i maksimalnu brzinu kretanja. Također, implementirana je funkcionalnost kamere koja prati glavnog lika tijekom igranja. Za ovu svrhu koriste se klase `UCameraComponent`, koja predstavlja vidno polje kamere i omogućuje podešavanje postavki kamere. Dio konstruktora je prikazan u ispisu 1.

```
PlayerCamera =
CreateDefaultSubobject<UCameraComponent>("PlayerCamera");
PlayerCamera->SetupAttachment(GetCapsuleComponent());


MoveSpeed = 1.0f;
bHasKeyRed = false;
bHasKeyBlue = false;
bHasKeyGrey = false;
bTimesUp = false;
```

Ispis 1: Dio konstruktora

Pored toga, ključne metode `MoveForward` i `MoveRight` (Ispis 2) implementirane su za omogućavanje kontrole lika tijekom igre. Te metode koriste funkciju `AddMovementInput` kako bi omogućile glatko i precizno kretanje lika prema naprijed i bočno. Funkcija `GetActorForwardVector` vraća vektor koji predstavlja smjer naprijed za glavnog lika, odnosno vektor koji pokazuje u kojem smjeru gleda lik.

Funkcija `GetInputAxisValue("MoveForward")` dobavlja vrijednost unosa za os `MoveForward`. Os `MoveForward` predstavlja naprijed/nazad kretanje, a vrijednost se dobiva iz korisničkog unosa. Ta vrijednost će odrediti koliko brzo će se objekt kretati prema naprijed.

Izračun konačnog smjera kretanja objekta postignuto je tako što je pomnožen vektor smjera naprijed s vrijednošću unosa za naprijed/nazad kretanje i dodan je vektor smjera udesno pomnožen s vrijednošću unosa za kretanje udesno. Na ovaj način se dobiva konačan vektor koji predstavlja ukupan smjer kretanja objekta, kombinirajući pomake u oba smjera.

Metoda `GetSafeNormal` normalizira vektor, što znači da mu postavlja duljinu na jedan. To je korisno jer će tako svi vektori smjera imati isti utjecaj na kretanje, bez obzira na duljinu pojedinih vektora. Kada se pozove `AddMovementInput`, lik će se kretati prema naprijed ako je `Value` pozitivan ili unatrag ako je negativan.

Ulazna vrijednost `Value` je vrijednost koja određuje koliko brzo će lik ići prema naprijed ili unatrag. Ako je pozitivan, lik će se kretati prema naprijed s brzinom varijable `MoveSpeed` (koja je podatkovni član klase `AGameProjectCharacter` i koja označava maksimalnu brzinu kretanja lika). Ako je negativan, lik će se kretati unatrag s istom brzinom.

```
void AGameProjectCharacter::MoveForward(float Value)
{
      FVector MovementDirection = (GetActorForwardVector() *
      Value + GetActorRightVector() *
      GetInputAxisValue("MoveRight")).GetSafeNormal();
      AddMovementInput(MovementDirection, MoveSpeed);
}
void AGameProjectCharacter::MoveRight(float Value)
{
      FVector MovementDirection = (GetActorForwardVector() *
      GetInputAxisValue("MoveForward") + GetActorRightVector() *
      Value).GetSafeNormal();
      AddMovementInput(MovementDirection, MoveSpeed);
}
```

Ispis 2: Metode kretanja lika

Metoda nazvana `SetupPlayerInputComponent` (Ispis 3) koja je također dio klase `AGameProjectCharacter`. Ova metoda se koristi za postavljanje ulaznih kontrola (engl. *input*) koje će omogućiti igraču upravljanje glavnim likom (engl. *character*) tijekom igranja igre.

Unutar ove metode, koristi se komponenta unosa igrača `PlayerInputComponent` koji je objekt tipa `UInputComponent` i predstavlja komponentu koja omogućuje prikupljanje i obradu ulaznih događaja od igrača.

```
void
AGameProjectCharacter::SetupPlayerInputComponent(UInputComponent
* PlayerInputComponent)

{

      Super::SetupPlayerInputComponent(PlayerInputComponent);

      PlayerInputComponent->BindAxis("MoveForward", this,
&AGameProjectCharacter::MoveForward);

      PlayerInputComponent->BindAxis("MoveRight", this,
&AGameProjectCharacter::MoveRight);

      PlayerInputComponent->BindAction("PauseGame",
IE_Pressed,this, &AGameProjectCharacter::CheckPause);

}
```

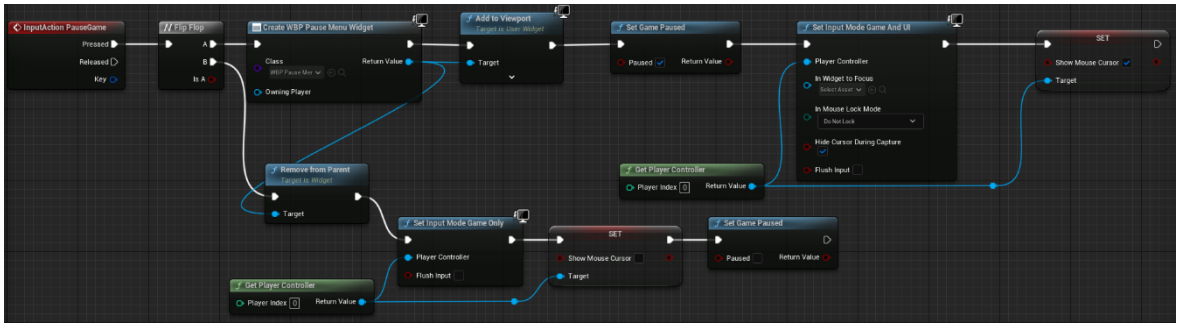
Ispis 3: Metoda za postavljanje ulaznih kontrola

Pomoću funkcija `BindAxis` i `BindAction` postavljaju se različite akcije i osi koje će reagirati na ulazne kontrole. Funkcija `BindAxis` linija povezuje ulaznu os `MoveForward` (koja je obično povezana s tipkama za kretanje naprijed i unatrag) s metodom `MoveForward` iz klase `AGameProjectCharacter`. Kada igrač pritisne odgovarajuće tipke za kretanje, metoda `MoveForward` će se aktivirati i glavni lik će se kretati naprijed ili unatrag ovisno o pritisnutim tipkama.

Isto tako i za ulaznu os `MoveRight` (koja je obično povezana s tipkama za kretanje lijevo i desno). Kada igrač pritisne odgovarajuće tipke za kretanje lijevo i desno, metoda `MoveRight` će se aktivirati i glavni lik će se kretati lijevo ili desno ovisno o pritisnutim tipkama.

U klasi također se nalazi funkcionalnost za upravljanje stanjem igre. Na primjer, metoda `CheckPause` omogućuje igraču pauziranje igre kako bi napravio kratku pauzu, nakon čega može nastaviti s igranjem. Funkcija `BindAction` povezuje ulaznu akciju `PauseGame` (koja je povezana s tipkom za pauziranje igre) s metodom `CheckPause`. Kada igrač pritisne tipku za pauziranje igre, koja je u ovom slučaju tipka „P“ metoda `CheckPause` će se aktivirati i igra će se pauzirati ili nastaviti, ovisno o trenutnom stanju igre. To je definirano kroz nacrt (Slika 10), u slučaju kada se tipka „P“ pritisne otvara se

pauzirani zaslon koji je već prethodno kreiran kao widget nacrt pod nazivom WBP_PauseMenu. Igra se pauzira te se prikaze kursor s kojim se vrši interakcija s botunima za nastavak igre, ponovno pokretanje ili povratak na glavni izbornik. Kroz ove veze (engl. *bindings*), igrač će moći upravljati likom u igri pomoću ulaznih kontrola i tako sudjelovati u igranju igre.



Slika 10: Nacrt pauziranje igre

Metoda `Tick` (Ispis 4) automatski se poziva za svaki okvir igre i koristi se za dinamičko ažuriranje logike lika ili igre tijekom samog igranja.

```
void AGameProjectCharacter::Tick(float DeltaTime)
{
      Super::Tick(DeltaTime);
      float ForwardValue = GetInputAxisValue("MoveForward");
      float RightValue = GetInputAxisValue("MoveRight");

      if (ForwardValue != 0.0f || RightValue != 0.0f)
      {
          if (ForwardValue > 0.0f && RightValue > 0.0f)
          {
             LastMovementDirection =
      EMovementDirection::ForwardRight;
          }
      ...
      }
```

```
    switch (LastMovementDirection)
    {
    case EMovementDirection::Forward:
        CharacterMesh->SetRelativeRotation(FRotator(0.0f,
    270.0f, 0.0f));
        break;
    ...
    }
}
```

Ispis 4: Dio `Tick` metode koji rotira lika

Metoda `Tick` je poziv metode iz nad klase (roditeljske klase), što osigurava da se i dalje izvršava osnovna logika metode `Tick` u roditeljskoj klasi. Dok naredba `switch` provjerava vrijednost varijable `LastMovementDirection` koja predstavlja smjer posljednjeg kretanja lika. Ovisno o vrijednosti varijable, postavit će se odgovarajuća rotacija lika, što će ga vizualno orijentirati prema naprijed, nazad, lijevo, desno ili dijagonalno.

Kroz klasu `AGameProjectCharacter`, implementirana je i metoda za detekciju smrti lika `Die` (Ispis 5).

```
void AGameProjectCharacter::Die()
{
    AGameProjectGameModeBase* GameMode =
    Cast<AGameProjectGameModeBase>(GetWorld()-
    >GetAuthGameMode());
    if (GameMode){GameMode->GameOver();}
}
```

Ispis 5: Metoda `Die`

U metodi `GameOver`, prvo se pokušava dohvatiti referenca na upravljač igrača (engl. *PlayerController*) koji kontrolira igrača pomoću funkcije `GetPlayerController`. Ako je pokazivač `PlayerController` valjan, dalje se nastavlja s izvođenjem kôda. Zatim se stvara widget za prikazivanje GameOver poruke pomoću funkcije `CreateWidget`, pri čemu se provjerava je li stvoreni widget valjan

pomoću `Cast<UGameProjectGameOverWidget>`. Ako je widget valjan, dodaje se na prikazani zaslon pomoću metode `AddToViewport`, te se postavlja pauza u igri i mijenja način rada unosa kako bi igrač mogao interaktirati samo s prikazanim widgetom (Ispis 6).

```
APlayerController* PlayerController =
UGameplayStatics::GetPlayerController(this, 0);
    if (PlayerController)
    {
       GameOverWidget =
Cast<UGameProjectGameOverWidget>(CreateWidget(GetWorld(),GameOve
rWidgetClass));
       if (GameOverWidget)
       {
          GameOverWidget->AddToViewport();
          PlayerController->SetPause(true);
          PlayerController->SetInputMode(FInputModeUIOnly());
          PlayerController->bShowMouseCursor = true;
       }
    }
```

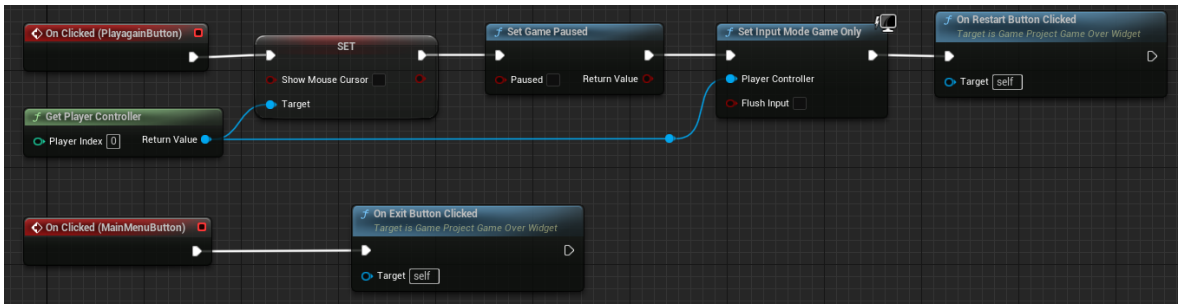
Ispis 6: Metoda `GameOver`

Widget sadrži dva botuna: izlaz (engl. *Exit*) i ponovno pokretanje (engl. *Restart)*, koji omogućuju igraču da napusti igru ili ponovno pokrene trenutnu razinu (Slika 11 i Ispis 7).

```
void UGameProjectGameOverWidget::OnExitButtonClicked()
{
    UGameplayStatics::OpenLevel(GetWorld(), FName("MenuLevel"),
true);
}

void UGameProjectGameOverWidget::OnRestartButtonClicked()
{
    FString LevelName =
UGameplayStatics::GetCurrentLevelName(GetWorld(), true);
    UGameplayStatics::OpenLevel(GetWorld(), FName(LevelName),
true);
}
```

Ispis 7: Metode u `UGameProjectGameOverWidget` klasi



Slika 11: Pozivanje metoda kroz nacrt

Metoda `OnExitButtonClicked` se poziva kada igrač klikne na botun „Exit“. U metodi se koristi funkcija `OpenLevel` kako bi se igra prebacila na razinu s imenom `MenuLevel`. Argument `true` u funkciji znači da se stvaraju nove instance razina, što znači da će se prethodna razina istog imena zatvoriti i otvoriti nova instanca razina `MenuLevel`. Ovo se obično koristi za povratak na glavni izbornik igre.

Klik na botun „Restart“ poziva se metoda `OnRestartButtonClicked`. Prvo se dohvaća ime trenutne razine pomoću funkcije `GetCurrentLevelName`. Zatim se koristi funkcija `OpenLevel` kako bi se igra ponovno pokrenula na trenutnoj razini, tako što se prebacuje na razinu s istim imenom kao trenutna razina.

Kombinacijom kôda iz klasa `AGameProjectCharacter` i `AGameProjectGameModeBase`, igra ima implementiranu logiku završetka igre kad lik igrača umre. Kad lik umre, prikazuje se „GameOver“ poruka na ekranu, igra se zaustavlja, i igraču se omogućuje interakcija s porukom. Ova funkcionalnost doprinosi boljem iskustvu igrača i omogućuje mu da zna kada je igra završena.

Animacija lika u Unreal Engineu omogućava da likovi u igri ožive i dobiju realističan izgled i pokrete. Ovaj proces započinje uvozom 3D modela lika u Unreal Engine, zajedno s odgovarajućim kosturom koji će omogućiti animiranje lika. Nakon što je 3D model uvezen, koristi se alat za izradu animacija, poput Blend Spacea, za kreiranje različitih pokreta i animacija lika.

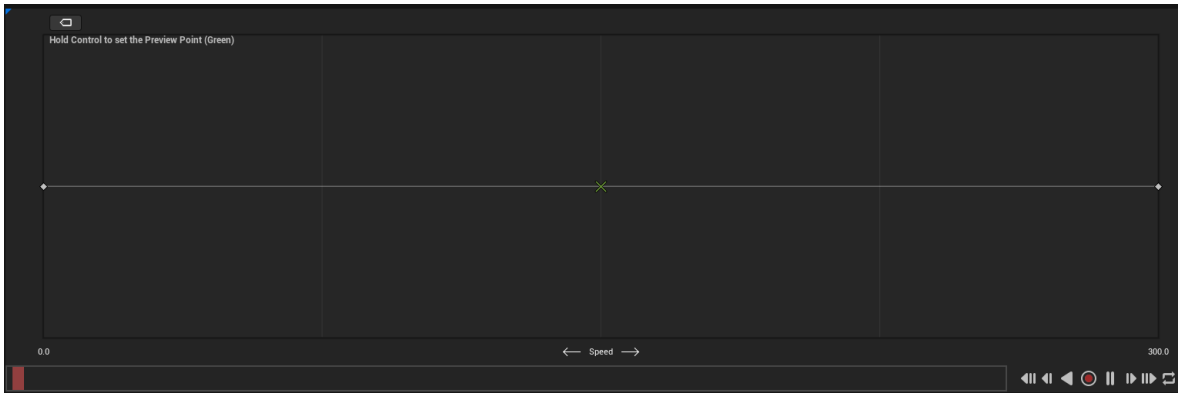
Blend Space je alat koji omogućava kombiniranje više animacija kako bi se stvorili glatki prijelazi između njih. Primjerice, može se koristiti za stvaranje trčanja, hodanja i

stajanja, a zatim se međusobno kombiniraju kako bi se omogućilo glatki prelazak iz jednog u drugi.

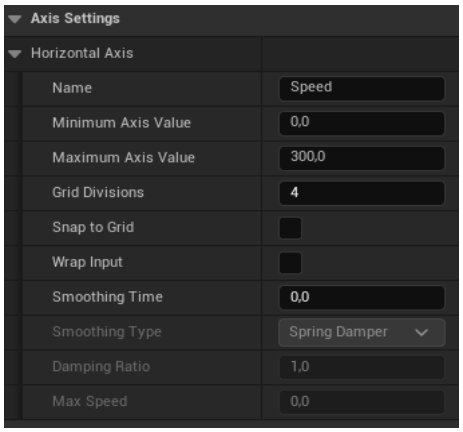
Kostur lika, također poznat kao rigging, je postupak postavljanja kostiju i zglobova unutar 3D modela kako bi se omogućilo animiranje lika. Svaka kost predstavlja dio lika (noga, ruka, glava itd.), dok su zglobovi mjesta gdje se kosti povezuju i omogućavaju pokrete.

Alat za izradu animacija u Unreal Enginu omogućava postavljanje ključnih okvira (engl. *keyframes*) koji definiraju položaj i rotaciju kostiju tijekom vremena. Ovi ključni okviri čine glatke prijelaze između različitih stanja i pokreta lika.

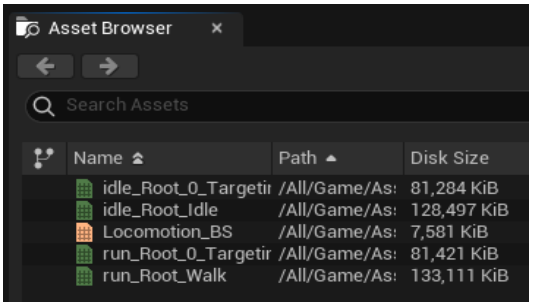
Na slici 12 može se vidjeti horizontalna os koordinatne mreže. U detaljima koordinatne mreže (Slika 13) horizontalna os je nazvana brzina (engl. *speed*) s maksimalnom vrijednosti osi (engl. *Maximum Axis Value*) 300, dok je minimalna 0. U donjem dijelu desne strane je popis svih animacija koje su uvedene u projekt (Slika 14). Za prikaz tih animacija ostvaruje se dvoklikom na pojedinu animaciju gdje se otvara novi prozor u kojem se prikazuje animacija.



Slika 12: Koordinatna mreža

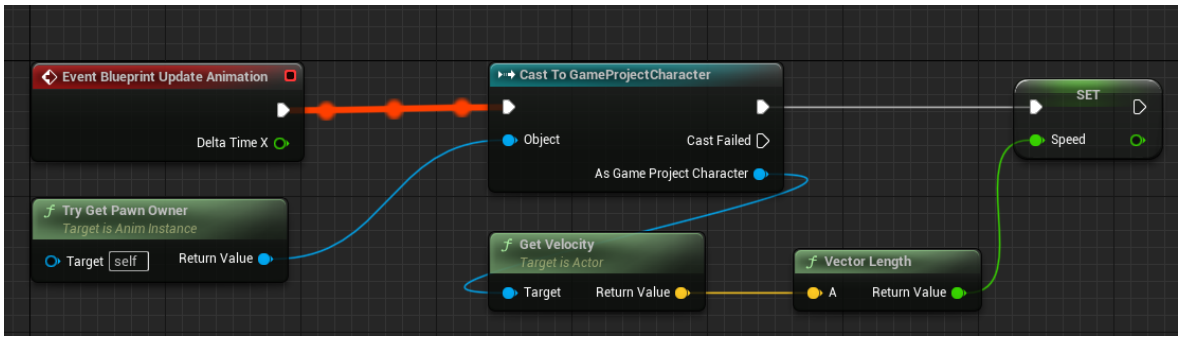


Slika 13: Detalji koordinatna mreže

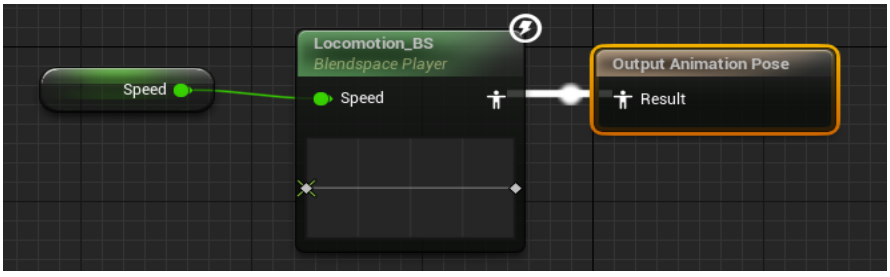


Slika 14: Lista animacija

Na horizontalnoj osi je minimalna vrijednost 0, što predstavlja mirovanje odnosno brzinu kretanja lika. U listi animacija se nađe ona koja je `Idle` i postavi se na tu vrijednost. Nakon toga na maksimalnu vrijednost, u ovom slučaju 300, se odabire vrsta animacije `Walk`. Lik sada ima animacije od šetanja do stajanja na mjestu. Sada se odabire nacrt (engl. *blueprint*) na gornjoj desnoj strani prozora gdje je potrebno kreirati događaj koji će ažurirati brzinu lika (Slika 15) te naknadno prenijeti stanje koje će ažurirati animaciju između stajanja i hodanja (Slika 16).



Slika 15: Nacrt brzine



Slika 16: Primjer stanja

4.2. Implementacija GameMode klase u igri

Postoji GameMode klasa, `AGameProjectGameModeBase`, koja je odgovorna za upravljanje logikom igre. Ova klasa ima niz metoda koje se koriste za različite funkcionalnosti igre, uključujući ažuriranje stanja igre, praćenje prikupljenih predmeta, rad s vremenom i prikazivanje odgovarajućih widgeta.

Metoda `BeginPlay` je dio početnog životnog ciklusa GameMode klase i poziva se kada igra započne (Ispis 8). Prvo se poziva metoda `Super::BeginPlay` kako bi se izvršile osnovne inicijalizacije iz nadređene klase. Zatim se dohvaća niz svih instanci objekata klase `AGameProjectItemBase` u igri pomoću funkcije `GetAllActorsOfClass`.

Broj prikupljenih predmeta `ItemsCollected` (podatkovni član) postavlja se na broj elemenata u nizu, što predstavlja ukupan broj predmeta u razini `ItemsInLevel` (podatkovni član). Također, u ovoj metodi se stvara i prikazuje widget klase `UGameProjectWidget`, koji služi za prikaz informacija o igri, uključujući prikupljene predmete. Ako je stvaranje widgeta uspješno, dodaje se na viewport i poziva se metoda `UpdateItemText` za ažuriranje broja prikupljenih predmeta u widgetu.

Nakon toga, stvara se instanca klase `AGameProjectTimer` naziva `TimerManager` i pokreće se mjerenje vremena pomoću njegove metode `StartTimer`. Na kraju, ažuriraju se svi botuni klase `AGameProjectARotatingButton` u razini kako bi se postavio njihov trenutni položaj na temelju stanja varijable `bIsBtnPressed` (podatkovni član klase `AGameProjectARotatingButton`)

```
TArray<AActor*> Items;
UGameplayStatics::GetAllActorsOfClass(GetWorld(),
AGameProjectItemBase::StaticClass(), Items);
ItemsInLevel = Items.Num();
if (GameWidgetClass){
    GameWidget =
Cast<UGameProjectWidget>(CreateWidget(GetWorld(),GameWidgetClass
));
    if (GameWidget){
       GameWidget->AddToViewport();
       UpdateItemText();
    }
}
TimerManager = GetWorld()->SpawnActor<AGameProjectTimer>();
if (TimerManager){TimerManager->StartTimer();}
TArray<AActor*> Buttons;
UGameplayStatics::GetAllActorsOfClass(GetWorld(),
AGameProjectARotatingButton::StaticClass(), Buttons);
for (AActor* button : Buttons){
    AGameProjectARotatingButton* RotatingButton =
Cast<AGameProjectARotatingButton>(button);
    if (RotatingButton){
       RotatingButton->UpdateButtonLocation(RotatingButton-
>bIsBtnPressed);
    }
}
```

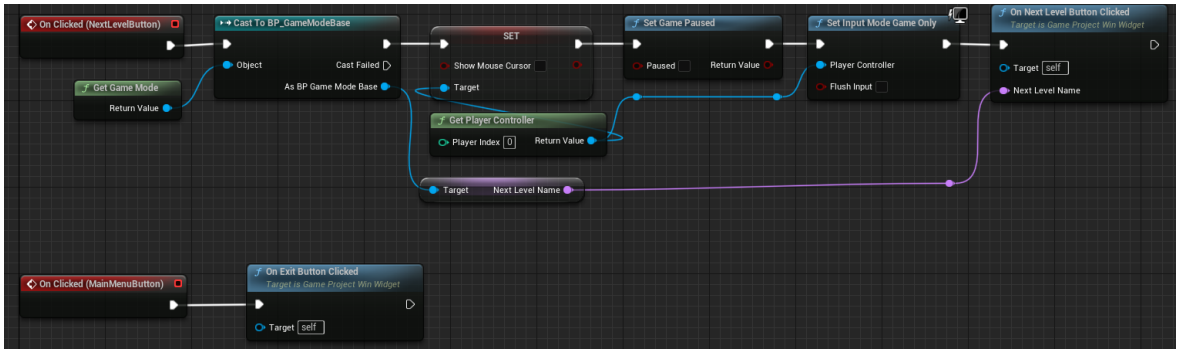
Ispis 8: Metoda `BeginPlay`

U GameMode klasi metoda `UpdateItemText` ažurira prikaz broja prikupljenih predmeta u widgetu `UGameProjectWidget`. Dok se metoda `ItemCollected` poziva kada igrač prikupi novi predmet. Povećava se broj prikupljenih predmeta `ItemsCollected` i poziva se `UpdateItemText` kako bi se ažurirao prikaz u widgetu.

Tu je i metoda `StopTimer` koja poziva istoimenu metodu u klasi `AGameProjectTimer` gdje zaustavlja mjerenje vremena koje je pokrenuto u metodi `BeginPlay` kada je to potrebno, a u metodi `UpdateTimer` se ažurira prikaz preostalog vremena u widgetu `UGameProjectWidget`. Uglavnom, klasa `AGameProjectTimer` se koristi za praćenje vremena u igri, odgovorna je za mjerenje vremena koje igrač ima na raspolaganju za završetak razine nakon čega umire.

Ima još jedan prikaz widgeta, kada igrač skupi sve zlatne predmete i stane na cilj poziva se metoda `ShowWinWidget` i prikaže se pobjednički widget sa dva botuna od koji je jedan za prelazak na sljedeću razinu, jedan za povratak na glavni izbornik. Prvo se dohvaća referenca na `PlayerController`, a zatim se stvara i prikazuje widget `UGameProjectWinWidget`, koji služi za prikazivanje poruke „Congratulation!“. Također, postavlja se pauza na igri i mijenja se način rada unosa kako bi igrač mogao interaktirati samo s prikazanim widgetom isto kao i kod metode `Die`.

U nacrtu se dohvati vrijednost varijable `NextLevelName` i šalje se metodi `OnNextLevelButtonClicked` u klasi `ShowWinWidget` te se ostale funkcije vraćaju na izvorno stanje kao na primjer kursor miša, pauziranje i način unosa (Slika 17).



Slika 17: Nacrt Widgeta

Na kraju ove dvije metode pozivaju funkciju `OpenLevel` gdje će otvoriti sljedeću razinu ili se vratiti na glavni izbornik ovisno o tome koji se botun odabrao (Ispis 9).

```
void UGameProjectWinWidget::OnNextLevelButtonClicked(FName
NextLevelName)
{
    UGameplayStatics::OpenLevel(GetWorld(), NextLevelName);
}

void UGameProjectWinWidget::OnExitButtonClicked()
{
    UGameplayStatics::OpenLevel(GetWorld(), FName("MenuLevel"),
true);
}
```

Ispis 9: Metode `UGameProjectWinWidget` klase

4.3. Implementacija različitih objekata i mehanika u igri

Ova sekcija se sastoji od različitih klasa koje nasljeđuju klasu `AActor` u Unreal Enginu. Svaka od tih klasa predstavlja igračke objekte ili entitete koji se pojavljuju unutar same igre. U nastavku je objašnjena svaka od tih klasa i kako su implementirane u igri kako bi se stvorile različite igračke mehanike i objekti.

4.3.1. AGameProjectItemBase

Klasa `AgameProjectItemBase` predstavlja temeljni objekt u igri koji igrač može prikupiti u ovom slučaju zlatni predmet. Objekt ima vizualni model (mesh) i može se rotirati oko svoje osi. Također, emitira zvuk kad ga igrač prikupi, te također koristi Niagara sustav za prikazivanje posebnih čestica (engl. *particles*).

U konstruktoru klase `AGameProjectItemBase` se postavljaju početne vrijednosti i konfiguracija komponenti za ovu klasu (Ispis 10).

```
PrimaryActorTick.bCanEverTick = true;

Mesh = CreateDefaultSubobject<UStaticMeshComponent>("Mesh");
RootComponent = Mesh;

CoinSoundComponent =
CreateDefaultSubobject<UAudioComponent>("CoinSoundComponent");
CoinSoundComponent->SetupAttachment(RootComponent);
CoinSoundComponent->SetSound(CoinSound);

NiagaraComponent =
CreateDefaultSubobject<UNiagaraComponent>("NiagaraComponent");
NiagaraComponent->SetupAttachment(RootComponent);
NiagaraComponent->Deactivate();

Mesh->OnComponentBeginOverlap.AddDynamic(this,
&AGameProjectItemBase::OverLapBegin)
```

Ispis 10: Konstruktor

Metoda `PrimaryActorTick.bCanEverTick` omogućuje osvježavanje objekta tijekom igre, što znači da će metoda `Tick` biti pozvana za svaki frame ako je potrebno. `CreateDefaultSubobject` stvara novu instancu klase `UStaticMeshComponent` koja će se koristiti kao vizualni prikaz objekta.

Kako bi se postavila korijenska komponenta (engl. *root componenet*) ovog objekta (trenutna instanca objekta `AGameProjectItemBase` koja se inicijalizira i konfigurira unutar konstruktora), mesh komponenta se postavlja kao `RootComponent`. Ova korijenska komponenta je centralna komponenta koja drži sve ostale komponente koje pripadaju ovom objektu.

Ova postavka omogućuje pravilno povezivanje svih ostalih komponenata, kao što su kolizijski okviri, zvukovi itd., na ovu mesh komponentu ili na druge komponente koje su već pričvršćene na nju.

Metoda `BeginPlay` je naslijeđena iz klase `AActor` i poziva se kada se objekt prvi put pojavi u igri. U ovom slučaju, metoda je prazna i nije potrebno izvršiti dodatne radnje kada objekt počinje igrati. Metoda `Tick` također je naslijeđena iz klase `AActor` i poziva se svaki frame kako bi se osvježio objekt tijekom igre.

U ovoj implementaciji, poziva se metoda `AddRotationToActor` (Ispis 11), koja dodaje rotaciju objektu na temelju vremenskog koraka `DeltaTime`. Rotacijska brzina je postavljena na 45 stupnjeva po sekundi, a zatim se rotacija primjenjuje na objekt pozivom metode `AddActorLocalRotation`.

```
void AGameProjectItemBase::AddRotationToActor(float DeltaTime){
    float RotationSpeed = 45.0f;
    FRotator RotationDelta(0.0f, RotationSpeed * DeltaTime,
0.0f);
    AddActorLocalRotation(RotationDelta);
}
```

Ispis 11: Metoda `Tick` i `AddRotationToActor`

Tu je još i metoda `OverLapBegin` koja se poziva kada se ovaj objekt preklapa s drugim objektom koji ima koliziju (Ispis 12). Metoda prima informacije o preklapanju i drugom objektu `OtherActor`. U ovoj implementaciji, prvo se provjerava je li `OtherActor` tipa `AGameProjectCharacter`. Ako je, reproducira zvuk na poziciji ovog objekta i ako postoji komponenta `NiagaraComponent` i nije već aktivan, aktivira se i postavlja se na istu poziciju kao mesh komponenta.

Poziva se metoda `ItemCollected` u GameMode klasi `AgameProjectGameModeBase` kako bi se obavijestio `GameMode` da je igrač prikupio ovaj objekt. Zatim se ovaj objekt uništava pozivom `Destroy` jer je prikupljen.

```
AGameProjectGameModeBase* GameMode =
Cast<AGameProjectGameModeBase>(GetWorld()->GetAuthGameMode());
if(Cast<AGameProjectCharacter>(OtherActor)){
    if (CoinSound){
       UGameplayStatics::PlaySoundAtLocation(this, CoinSound,
GetActorLocation());
    }
    if (NiagaraComponent && NiagaraComponent->IsValidLowLevel())
    {
       NiagaraComponent->SetWorldLocation(Mesh-
>GetRelativeLocation());
       if (!NiagaraComponent->IsActive()){
          NiagaraComponent->Activate();
       }
    }
    GameMode->ItemCollected();
    Destroy();
}
```

Ispis 12: Kolizija

4.3.2. AGameProjectTrapBase

Ovo je klasa za zamke u igri. Zamke imaju vizualni model (mesh) i koliziju. Zamke mogu biti aktivne ili neaktivne koje su definirane sa bool varijablom `bTrapActive`. Kada igrač pređe preko zamke dok je neaktivna, zamka će reagirati (npr. podići se). Također ima zvuk koji se reproducira kada se zamka aktivira.

U konstruktoru (Ispis 13) se postavljaju početne vrijednosti i konfiguracija komponenti za ovu klasu. `TrapCollision` predstavlja kolizijski okvir zamke. Ovaj okvir koristi se za detekciju preklapanja s drugim objektima. `RootComponent` postavlja `TrapCollision` komponentu kao korijensku komponentu ovog objekta.

Komponenta `TrapMesh` klase `UStaticMeshComponent`, služi za vizualni prikaz zamke i postavlja komponentu kao pod komponentu `TrapCollision` komponente. To znači da će komponenta `TrapMesh` pratiti `TrapCollision` i kretati i rotirati se zajedno s njom. Komponenta `TrapSoundComponent` je pod objekt klase `UAudioComponent`, koji će se koristiti za reprodukciju zvuka zamke (ispis 19).

Varijabla `OriginalMeshLocation` sprema početnu lokaciju komponente `TrapMesh` kako bi se kasnije mogla vratiti na tu poziciju. Varijable `bTrapActive`, `bTrapTriggered`, i `TrapMeshOffsetZ` koriste se za praćenje stanja i pomicanje vizualnog modele zamke za određeni iznos `TrapMeshOffsetZ` kada se zamka aktivira.

```
TrapCollision =
CreateDefaultSubobject<UBoxComponent>("TrapCollision");
RootComponent = TrapCollision;
TrapMesh =
CreateDefaultSubobject<UStaticMeshComponent>("TrapMesh");
TrapMesh->SetupAttachment(TrapCollision);

TrapSoundComponent =
CreateDefaultSubobject<UAudioComponent>("ButtonSoundComponent");
TrapSoundComponent->SetupAttachment(RootComponent);
TrapSoundComponent->SetSound(TrapSound);

OriginalMeshLocation = TrapMesh->GetRelativeLocation();
bTrapActive = false;
bTrapTriggered = false;
TrapMeshOffsetZ = 30.0f;
```

Ispis 13: Dio `Trap` konstruktora

Kada dođe do kolizije poziva se metoda `OnTrapOverlapBegin` (Ispis 14). Ako je varijabla `bTrapActive` postavljen na `false`, tada se provjerava je li preklapanje s igračem. U tom slučaju, postavlja se varijabla `bTrapActive` na `true`, što označava da je

zamka aktivirana. Kad je varijabla `bTrapActive` već `true`, tada metoda provjerava ponovno preklapanje s igračem. Ako je varijabla `bTrapTriggered` također `true`, to znači da je zamka već aktivirana i igrač je ostao u zamci, pa se poziva metoda `Die` na igraču, što znači da igrač umire. Zatim se varijabla `bTrapTriggered` postavlja na `false` kako bi se osiguralo da se metoda `Die` ne poziva više puta dok je igrač u zamci.

```
AGameProjectCharacter* PlayerCharacter =
Cast<AGameProjectCharacter>(OtherActor);
if (!bTrapActive){
    if (PlayerCharacter){
        bTrapActive = true;
    }
}
else{
    if (PlayerCharacter && bTrapTriggered){
        PlayerCharacter->Die();
        bTrapTriggered = false;
    }
}
```

Ispis 14: Metoda `OnTrapOverlapBegain`

Nakon, kada se prestaje preklapati s ovom zamkom poziva se metoda `OnTrapOverlapEnd` (Ispis 15). Ako je varijabla `bTrapActive` `true`, to znači da je zamka već aktivirana i provjerava se je li preklapajući objekt igrač. Tada se reproducira zvuk zamke, a komponenta `TrapMesh` se postavlja na početnu lokaciju `TrapMeshOffsetZ`, čime se postiže efekt da se zamka podiže nakon što je igrač prošao preko nje. Također, varijabla `bTrapTriggered` se postavlja na `true` kako bi se mogla obraditi situacija kada igrač ostane u zamci.

```
if (bTrapActive && OtherActor && OtherActor-
>IsA<AGameProjectCharacter>()){
    if (TrapSound){UGameplayStatics::PlaySoundAtLocation(this,
TrapSound, GetActorLocation());}
    TrapMesh->SetRelativeLocation(OriginalMeshLocation +
FVector(0,0,TrapMeshOffsetZ));
    bTrapTriggered = true;
}
```

Ispis 15: Metoda `OnTrapOverlapEnd`

4.3.3. AGameProjectARotatingPlatform

Klasa predstavlja rotirajuću platformu u igri. Platforma ima također vizualni model koji se može rotirati oko svoje osi s određenom brzinom i kutom rotacije. Također ima koliziju za detekciju sudara s drugim objektima. Platforma ima zvuk koji se reproducira dok se rotira. Također ima funkcije za obradu preklapanja i rotiranje platforme.

Osim što ima komponente `PlatformCollision`, `PlatformMesh`, `PlatformSoundComponent` definirano u konstruktori tu su još i varijable `OriginalMeshRotation`, `RotationAngle`, `RotationSpeed`, `TargetRotation` i `bIsRotating` koriste se za praćenje stanja rotacije platforme i upravljanje rotacijom (Ispis 16).

```
OriginalMeshRotation = PlatformMesh->GetRelativeRotation();
RotationAngle = 90.0f;
RotationSpeed = 1.0f;
TargetRotation = FRotator(0, RotationAngle, 0);
bIsRotating = false;
```

Ispis 16: Dio konstruktora platforme

Bool varijabla `bIsRotating` se postavlja na `true` u metodi `Onoverlap Begin` ako se preklapa s igračem, što znači da će se platforma kasnije rotirati kaka prestane preklapanje jer se tada poziva metoda `OnOverlapEnd`. Tada se reproducira zvuk platforme, platforma se rotira na sljedeću stranu i postavlja varijabla `bIsRotating` na `false` kako bi se osiguralo da se rotacija ne odvija neprekidno (Ispis 17).

```
AGameProjectCharacter* PlayerCharacter =
Cast<AGameProjectCharacter>(OtherActor);
if (bIsRotating && PlayerCharacter){
    if (PlatformSound){UGameplayStatics::PlaySoundAtLocation(this,
PlatformSound, GetActorLocation());}
    PlatformMesh->SetRelativeRotation(OriginalMeshRotation +
TargetRotation);
    RotationAngle += 90.0f;
    TargetRotation = FRotator(0, RotationAngle, 0);
    bIsRotating = false;}
```

Ispis 17: Metoda `OnOverlapEnd`

4.3.4. AgameProjectARotatingButton

Klasa `AgameProjectARotatingButton` predstavlja botun u igri. Ovaj botun ima dvije glavne funkcije: aktiviranje rotirajućih platformi kada igrač preklopi s njim, te prilagodbu svog vizualnog stanja kad se pritisne.

Kao i do sad u ovom konstruktoru su komponente `ButtonCollision`, `ButtonMesh`, `ButtonSoundComponent` te varijable `OriginalMeshLocation`, `bIsBtnPressed`, `bIsBtnActive` i `ButtonMeshOffsetZ` slične kao i kod klase `AGameProjectARotatingPlatform` koriste se za praćenje stanja botuna i upravljanje njegovim vizualnim stanjem.

Metoda `OnOverlapBegin` (Ispis 18) se poziva kada se objekt preklapa s drugim objektom, u ovom slučaju, s igračem. Ako je botun aktivan i ako botun nije pritisnut, tada se botun označava kao aktivan, a zatim se aktiviraju sve rotirajuće platforme `RotatingPlatform->RotatePlatform` (Ispis 19) koje su prethodno registrirane i povezane s ovim botunom. Također, postavlja se tajmer koji će nakon 0.5 sekundi pozvati metodu `ResetButtonActivity`, koja će ponovno postaviti varijablu `bIsBtnActive` na `false` kako bi se spriječilo ponavljanje aktivacije platformi pri ponovnom preklapanju s igračem.

```
AGameProjectCharacter* PlayerCharacter =
Cast<AGameProjectCharacter>(OtherActor);
if (bIsRotating && PlayerCharacter){
    if (PlatformSound){
        UGameplayStatics::PlaySoundAtLocation(this,
PlatformSound, GetActorLocation());
    }
    PlatformMesh->SetRelativeRotation(OriginalMeshRotation +
TargetRotation);
    RotationAngle += 90.0f;
    TargetRotation = FRotator(0, RotationAngle, 0);
    bIsRotating = false;
}
```

Ispis 18: Metoda `OnOverlapBegin`

```
void AGameProjectARotatingPlatform::RotatePlatform()
{
    PlatformMesh->SetRelativeRotation(OriginalMeshRotation +
TargetRotation);
    RotationAngle += 90.0f;
    TargetRotation = FRotator(0, RotationAngle, 0);
}
```

Ispis 19: Metoda `RotatePlatform` u `AGameProjectARotatingPlatform` klasi

Za promjenu stanja botuna korist se metoda `UpdateButtonState` koja ažurira stanje botuna, tj. mijenja njegovo vizualno stanje i postavlja varijablu `bIsBtnPressed` na `true` ili `false` ovisno o tome je li botun pritisnut ili ne (Ispis 20).

```
void AGameProjectARotatingButton::UpdateButtonLocation(bool
bIsPressed){
    if (bIsPressed){
        ButtonMesh->SetRelativeLocation(OriginalMeshLocation);
    }
    else{
        ButtonMesh->SetRelativeLocation(OriginalMeshLocation +
FVector(0, 0, ButtonMeshOffsetZ));
    }
}
void AGameProjectARotatingButton::UpdateButtonState(bool
bIsPressed){
    if (!bIsPressed){
        UpdateButtonLocation(!bIsPressed);
        bIsBtnPressed = true;
    }
    else{
        UpdateButtonLocation(!bIsPressed);
        bIsBtnPressed = false;
    }
}
```

Ispis 20: Metoda `UpdateButtonState`

Dok metoda `CheckButtonState` (Ispis 21) provjerava stanje svih botuna u igri. Ako pronađe neki drugi botun, poziva metodu `UpdateButtonState` na tom botunu kako bi osiguralo da svi botuni imaju ispravno ažurirano stanje.

```
void AGameProjectARotatingButton::CheckButtonState(){
    TArray<AActor*> Buttons;
    UGameplayStatics::GetAllActorsOfClass(GetWorld(),
StaticClass(), Buttons);
    for (AActor* button : Buttons){
        AGameProjectARotatingButton* RotatingButton =
Cast<AGameProjectARotatingButton>(button);
        if (RotatingButton){
            RotatingButton->UpdateButtonState(RotatingButton-
>bIsBtnPressed);
        }
    }
}
```

Ispis 21: Metoda `CheckButtonState`

4.3.5. AGameProjectKeyPickup i AGameProjectLockInteraction

Ovo je klasa za prikupljanje ključeva i klasa za interakciju s bravom u igri. Ključevi imaju različite tipove (crveni, plavi, sivi) koji se određuju enumeracijom `EKeyType`. Ključevi također imaju vizualni model i koliziju.

Kad igrač prikupi ključ, reproducira se zvuk. Također i brave imaju različite tipove (crvena, plava, siva) koji se određuju enumeracijom `ELockType`, te imaju vizualni model i koliziju. Kada igrač ima odgovarajući ključ, može interaktirati s bravom, što će reproducirati zvuk interakcije i omogućiti daljnji prolazak kroz razinu.

Enumeracija `enum class EKeyType` definira enumeraciju `EKeyType` (Ispis 22) s tri moguća tipa ključa (RedKey, BlueKey i GreyKey). Ova enumeracija će se koristiti za označavanje tipa ključa koji ovaj objekt predstavlja.

```
UENUM(BlueprintType)
enum class EKeyType : uint8
{
    KeyType1 UMETA(DisplayName = "RedKey"),
    KeyType2 UMETA(DisplayName = "BlueKey"),
    KeyType3 UMETA(DisplayName = "GreyKey"),
};
```

Ispis 22: `EKeyType`

Glavna metoda `OnKeyOverlap` se poziva kada se objekt preklapa s drugim objektom, u ovom slučaju, s igračem. Ako je igrač preklopio s ključem, metoda reproducira zvuk prikupljenog ključa, postavlja odgovarajuću zastavicu za tip ključa (prema enumeraciji `EKeyType`) za igrača i uništava sam objekt ključa (Ispis 23).

```
AGameProjectCharacter* PlayerCharacter =
Cast<AGameProjectCharacter>(OtherActor);
if (PlayerCharacter){
    if
(KeyPickupSound){UGameplayStatics::PlaySoundAtLocation(this,
KeyPickupSound, GetActorLocation());}
    switch (KeyType){
    case EKeyType::KeyType1:
       PlayerCharacter->bHasKeyRed = true;
       break;
    case EKeyType::KeyType2:
       PlayerCharacter->bHasKeyBlue = true;
       break;
    case EKeyType::KeyType3:
       PlayerCharacter->bHasKeyGrey = true;
       break;
    }
    Destroy();
}
```

Ispis 23: Metoda `OnKeyOverlap`

Implementaciju metode `OnDoorOverlap` u klasi `AGameProjectLockInteraction` koristi naredbu `switch` kako bi se ovisno o tipu `LockType` izvršila odgovarajuća provjera. Na primjer ako igrač ima ključ tipa `RedKey` varijabla `bHasKeyRed` je `true`, tada će ključ biti potrošen (postavlja se na `false`), reproducirati će se zvuk otključavanja i objekt će biti uništen pozivom funkcije `Destroy`. Ako igrač nema odgovarajući ključ, ništa se ne događa (Ispis 24).

```
AGameProjectCharacter* PlayerCharacter =
Cast<AGameProjectCharacter>(OtherActor);
if (PlayerCharacter){
    switch (LockType){
    case ELockType::LockType1:
       if (PlayerCharacter->bHasKeyRed){
          PlayerCharacter->bHasKeyRed = false;
          if (LockInteractionSound){
             UGameplayStatics::PlaySoundAtLocation(this,
LockInteractionSound, GetActorLocation());
          }
          Destroy();
       }
       break;

     ...
    }
}
```

Ispis 24: Metoda `OnDoorOverlap`

Tu je i metoda `AddMovementsToActor` koja dodaje animaciju ključa tijekom igre, rotirajući ključ oko Y osi i oscilirajući po Z osi kako bi stvorio vizualni efekt tijekom igre (Ispis 25).

```
void AGameProjectKeyPickup::AddMovementsToActor(float DeltaTime)
{
    float RotationSpeedYaw = -45.0f;
    float OscillationSpeed = 2.0f;
    float OscillationAmplitude = 15.0f;

    FRotator RotationDeltaYaw(0.0f, RotationSpeedYaw *
DeltaTime, 0.0f);
    AddActorLocalRotation(RotationDeltaYaw);

    FVector NewLocation = GetActorLocation();
    float PitchOffset = FMath::Sin(GetGameTimeSinceCreation() *
OscillationSpeed) * OscillationAmplitude;
    NewLocation.Z = 60.0f + PitchOffset;
    SetActorLocation(NewLocation);
}
```

Ispis 25: Metoda `AddMovementsToActor`

5. ZAKLJUČAK

U ovom završnom radu predstavljeno je istraživanje i razvoj igre koristeći mogućnosti Unreal Enginea kao razvojno okruženje. Rad je pružio uvid u proces izrade igre kroz upotrebu programskog jezika C++ i vizualnog skriptiranja uz korištenje gotovih biblioteka iz pogonskog alata. Cilj je bio pokazati različite načine implementacije funkcionalnosti i prikazati prednosti i svrhu svakog pristupa.

Unutar ovog rada naglasak je bio na programiranju, dok je izgled likova, okoline i zvuka riješen korištenjem gotovih dodataka. Iako to može utjecati na grafičku optimizaciju igre, omogućilo je brži razvoj i testiranje bez potrebe za naprednim znanjem programiranja.

Izrada igre predstavlja zahtjevan zadatak, koji zahtijeva kreativnost u stvaranju priče, ugođaja i zanimljivih igračkih mehanika. Unreal Engine je pružio snažan alat za olakšavanje razvoja igre, omogućivši fokusiranje na implementaciju originalnih ideja bez gubljenja previše vremena na tehničke aspekte.

Kombinirajući znanje programskog jezika C++ i vizualnog skriptiranja, moguće je prilagoditi igru prema potrebama projekta, pazeći na optimizaciju i resurse. Istovremeno, vizualni skriptni jezik nudi jednostavniji pristup za one koji nemaju programersko iskustvo, što može biti korisno prilikom testiranja ili za brzo prototipiranje.

U konačnici, Unreal Engine se pokazao kao snažno razvojno okruženje koje može značajno olakšati proces izrade igara, bez obzira na iskustvo programiranja, nudeći širok spektar mogućnosti za implementaciju raznih ideja.

LITERATURA

[1] Unreal Engine 5.2 Documentation,
https://docs.unrealengine.com/5.2/en-US/ (posjećeno 13.07.2023.).