Izrada 2D platformera u programskom alatu Unreal Engine 4

Mahnet, Domagoj

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Domagoj Mahnet

IZRADA 2D PLATFORMERA U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Domagoj Mahnet

Matični broj: 46444/17–R

Studij: Informacijski sustavi

IZRADA 2D PLATFORMERA U PROGRAMSKOM ALATU UNREAL ENGINE 4

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mario Konecki

Varaždin, rujan 2020.

Domagoj Mahnet

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad prvo će pokazati sam koncept rada Unreal Enginea 4 te proći kroz povijest njegovog razvoja i objasniti sam rad njegovih najbitnijih dijelova. Nakon toga opisan je platformer žanr računalnih igara, njihov razvoj od samih početaka pa sve do danas, te su dani primjeri igara koji najbolje oslikavaju sam žanr.

Nakon toga rad prolazi kroz opis izrade jedne takve igre pomoću Unreal Enginea 4. Dio je objašnjen kroz C++ kod, a dio kroz vizualno skriptiranje Unreal Enginea. Igrač upravlja glavnim likom koji ima izgled jednog srednjovjekovnog viteza. Cilj igre je doći do kraja levela putem preskačući otrovne gljive, a na putu se nalaze i agresivni zombiji.

Ključne riječi: Unreal Engine, platformer, C++, 2D računalna igra, zombi

Sadržaj

1. Uvod

Razvoj računalnih igara postao je jedna velika te unosna grana računalne industrije. Danas je gotovo nemoguće pronaći čovjeka koji nikad nije odigrao barem jednu računalnu igru. Tome je uvelike doprinijela njihova široka dostupnost, bilo to na računalu ili na mobilnom uređaju. Računalne igre također su postale veoma važan dio današnje kulture.

Tema ovog završnog rada jest prvo opisati Unreal Engine i njegov razvoj od samih početaka pa sve do danas. Unreal Engine je danas jedan od najpoznatijih i najkorištenijih motora igara današnjice. Nakon toga slijedi opis platformer žanra, koji je bio na vrhuncu popularnosti krajem 20. stoljeća. Platformer igra karakteristična je po skakanju s platforme na platformu kako bi se došlo do kraja razine. Najpoznatiji predstavnici ovog žanra su Super Mario, Crash Bandicoot i Sonic the Hedgehog. Ovaj rad sastoji se i od praktičnog dijela, izrade jedne 2D platformer računalne igre. Detaljno će se objasniti izrada razina, likova i objekata same igre, te programska logika koja ih povezuje u jednu smislenu i skladnu cjelinu.

Na odabir teme te samu izradu ovog rada uvelike je utjecala moja velika ljubav prema videoigrama, te nostalgija za samim platformerima koje sam igrao u djetinjstvu. Kako nisam nikada izradio svoju vlastitu računalnu igru, izrada ovog rada pomogla mi je u ispunjavanju dugogodišnje želje, te u usavršavanju znanja na području dizajna i znanja C++ programskog jezika.

2. Metode i tehnike rada

Za izradu same igre korišten je Unreal Engine 4. Njegovim korištenjem izrađen je glavni lik, kamera, platforme na koje glavni lik može skočiti, otrovne gljive, zombiji koji pokušavaju ubiti glavnog lika te grafičko sučelje same igre. Kako Unreal Engine 4 ne omogućuje direktnu izmjenu samog koda računalne igre, bio je potreban i Visual Sudio Community 2019. Za potrebe izrade ove dokumentacije korišten je Microsoft Word 2019.

Službena dokumentacija Unreal Enginea bila je glavni dio istraživanja mogućnosti samog Unreal Enginea, kao i Unrealov službeni Youtube kanal. Također su korišteni i razni Youtube tutoriali, te Unreal Engine forumi.

Izrada same igre odvija se slijedno, počevši od jednostavnih elemenata poput likova i objekata prema izradi kompletnih razina. Nakon samog dizajniranja, slijedi povezivanje svih tih elemenata jednu cjelinu putem koda, kako bi ti elementi mogli imati međusobnu interakciju.

Temeljna znanja za izradu ovog znanja steknuta su istraživanjem službene dokumentacije, Youtube tutoriala te internetskih foruma. Svi grafički elementi preuzeti su sa https://www.gameart2d.com te su slobodni za korištenje, bez licencnih odredbi.

3. Razrada teme

3.1. Unreal Engine

1998. godine inženjeri Epic Gamesa razvili su prvu inačicu Unreal Engine-a za potrebe razvoja igre imena Unreal, i time omogućili samom korisniku mogućnost modificiranja (eng. modding) same igre. Korisnik je mogao, korištenjem UnrealScripta, izmijeniti sadržaj Unreal igre ili dodati neki svoj. Također je i omogućena sposobnost izmjene ponašanja NPC-a (non-player characters). Te NPC-e kontrolira umjetna inteligencija putem niza algoritama koji se nalaze u samom programskom kodu.

Do 2002. godine Unreal Engine je konstantno nadograđivan,a među najznačajnije nadogradnje svakako ubrajamo sustav čestica (eng. particle system), statičke mrežne alate (eng. static mesh tools), motor fizike (eng. physics engine) i Matinee alat. Sustav čestica omogućio je generiranje posebnih efekata, poput magle ili dima [1]. Razvoj statičkih mrežnih alata donio je sposobnost manipuliranja objektima. Statička mreža je zapravo geometrijski dio koji sadrži određen set poligona koji se mogu spremiti u video memoriju, time omogućujući efikasan prikaz putem grafičke kartice [2]. Razvojem motora fizike došlo je do naprednih interakcija između dva objekta, time realno prikazujući rezultat njihove interakcije (npr. kolizija dva automobila). Matinee je alat kojim grafički inženjeri mogu kreirati kratke, neinteraktivne videosadržaje unutar neke računalne igre.

2006. Epic Games izdaje Unreal Engine 3, koji je dominirao tržištem sve do 2014. godine. Unreal Engine 3, koristeći unaprijeđen grafički motor u kombinaciji s DirectX-om 9/10 donio je ogroman pomak u realističnom prikazu likova te objekata. Također je predstavljen i sustav za vizualno skriptiranje naziva Kismet, time olakšavajući dizajnerima igre lakše kreiranje logike same igre, time smanjujući potrebu za samim kodiranjem. Dodana je podrška za Xbox360 i Playstation 3, redizajniran je sustav za manipuliranje osvjetljenjem te je izrađen novi motor fizike. Porast u potražnji mobilnih igara doveo je do razvoja podrške Unreal Enginea za razne mobilne platforme. Sve ove nadogradnje i tehnološke mogućnosti dovele su Unreal Engine 3 na mjesto najpopularnijeg motora igre koji se i dan danas često koristi.

Unreal Engine 4 objavljen je 2014. godine, a najvećom promjenom smatra se zamjena Kismeta s Blueprint sustavom [1].

3.1.1. Level Editor

Level Editor predstavlja osnovnu funkcionalnost kreiranja u Unreal Editoru. Ovdje se kreiraju razine, koje mogu predstavljati bogat open world svijet, ili pak samo nekoliko objekata

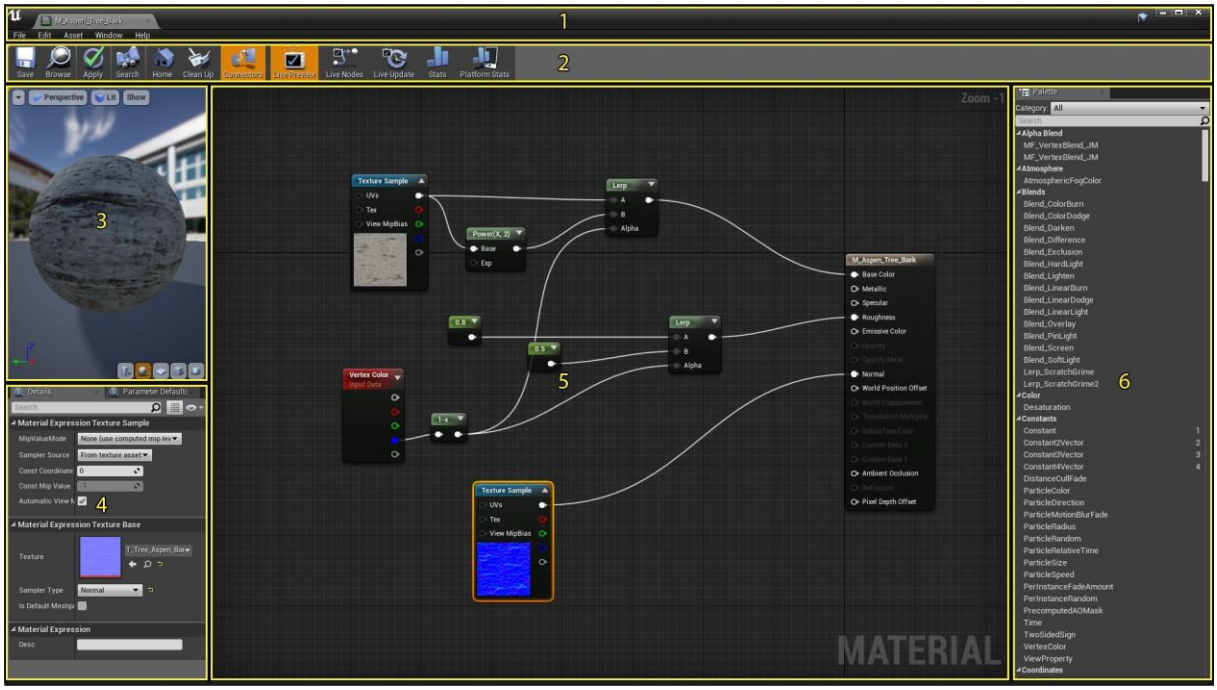
[3]. Razine su zapravo prostoru kojem se nalaze objekti i geometrija svijeta kojeg igra pokušava dočarati korisniku [4]. Svaki objekt koji se nalazi u tom svijetu, bio on svjetlo, dim ili lik, smatra se Actorom. Actori su svi oni objekti koji mogu biti postavljeni u neku razinu. Oni su zapravo generička klasa koja podržava translaciju, rotaciju i skaliranje. Mogu biti kreirani i uništeni putem samog igrinog koda [5].

Slika 1: Level Editor

1. Traka s karticama i menijem
2. Traka s alatima
3. Traka s modovima
4. Pretraživač sadržaja
5. Prozor
6. Traka s Actorima
7. Traka s detaljima [3]

3.1.2. Material Editor

Material Editor je grafičko sučelje bazirano na čvorovima koje korisniku omogućuje kreiranje shadera koje je moguće primijeniti na željenoj geometriji (npr. statičke ili kosturske mreže) s ciljem kreiranja željenih materijala.

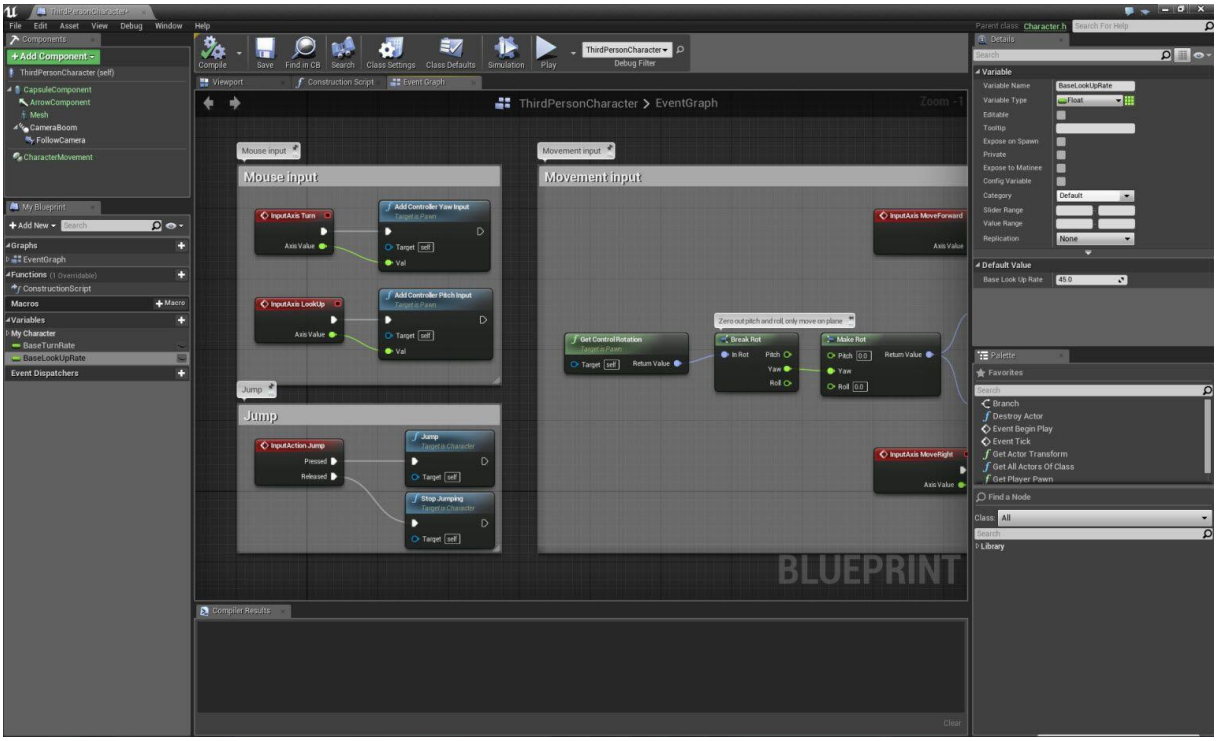


Slika 2: Material Editor

1. Meni
2. Traka s alatima
3. Prozor za prikaz objekta
4. Detalji
5. Prozor za prikaz u obliku grafa
6. Paleta [6]

3.1.3. Blueprint Editor

Blueprint sustav za vizualno skriptiranje koristi se za definiranje logike događaja unutar neke razine, kontroliranje interno skriptiranog ponašanja Actora, ili za kontroliranje kompleksnih animacija korištenih u sistemima za upravljanje igrivim likovima. Blueprint Editor omogućuje korisniku kreiranje i izmjenu složenih vizualnih skripata koje pogone različite aspekte igre u razvoju.



Slika 3: Blueprint Editor

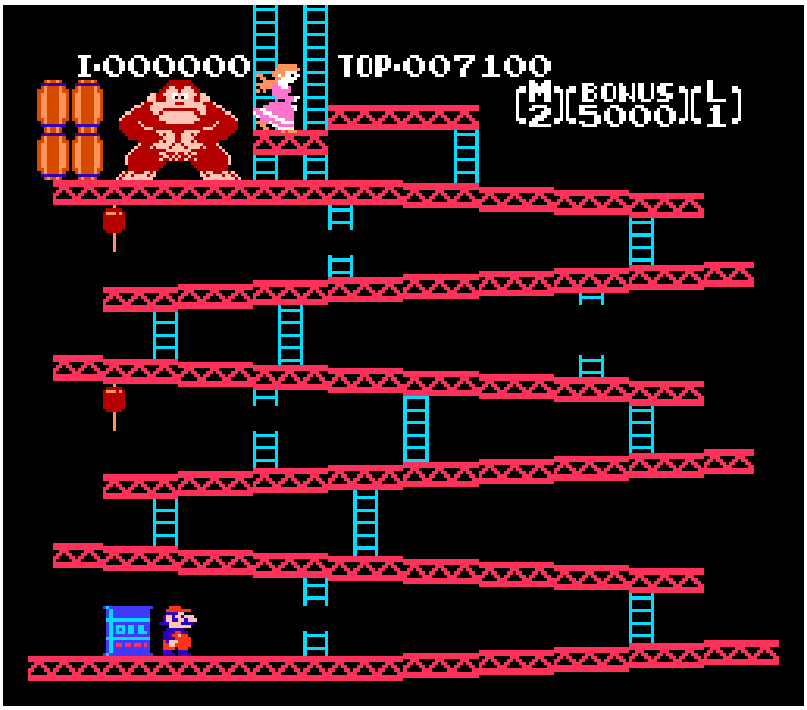
Blueprint Editor je zapravo grafički editor baziran na čvorovima. Koristi kontekstualno osjetljiv dizajn, time olakšavajući pristup funkcionalnostima objekata u specifičnom trenutku kada je ta funkcionalnost potrebna, ali u isto vrijeme nudi i fleksibilnost. Sadrži nekoliko panela i alata kako bi korisniku bilo olakšano kreiranje vlastitih varijabli i funkcija. Također sadrži i brojne alate za otklanjanje grešaka te analizu rada sustava kako bi korisnik mogao što brže otkloniti nedostatke te poboljšati tok podataka između raznih podsustava [7].

Postoji više vrsta Blueprint Editora, ovisno o tome gdje je Blueprint Editor potreban. Level Blueprint je specijalizirana vrsta Blueprinta koja predstavlja graf globalnih događaja unutar neke razine. Tako svaka razina ima automatski kreiran Blueprint koji se može modificirati. Blueprint Class omogućuje kreatorima jednostavno dodavanje funkcionalnosti uz

već postojeće klase koje definiraju logiku same igre. Data-Only Blueprint je klasa Blueprinta koja sadrži samo kod, varijable i komponente koje je naslijedila od roditeljske klase. Tada te klase mogu naslijeđena svojstva modificirati i izmijeniti, ali ne mogu dodavati nove elemente. Ovaj kod je također prikazan u grafičkom obliku pomoću čvorova [8].

3.2. 2D Platformer

Platformer igre su žanr videoigara i podžanr akcijskih igara. U tipičnim igrama ovog žanra igrač kontrolira svog lika u video igri te mu daje naredbe za kretanje. Igrač mora skakati između platformi te izbjegavati prepreke na putu. Platformere dijelimo na dva dijela: one s statičkim prikazom i one s pomičnim prikazom. Platformeri s statičkim prikazom svaki level prikažu u cjelini, dok se kod platformera s pomičnim prikazom okolina mijenja ovisno o kretanju lika kojim korisnik upravlja. Prve igre ovog žanra koristile su statičke prikaze, zbog tehničkih limita u to vrijeme.



Slika 4: Donkey Kong

Najpoznatije igre tog vremena su Donkey Kong i Burgertime. Specifičnost ovog žanra jest u tome što se igre sastoje od nekoliko razina, svaka teža od prijašnje. Također je moguće i postojanje takozvanih „boss“ razina, u kojim se lik bori protiv manjeg broja, ali zato znatno jačeg neprijatelja [9].

3.2.1. 1980-e

Donkey Kong se smatra prvom video igrom ovog žanra. Objavljena je 1981. godine, a cilj igre jest stići do najviše platforme putem izbjegavajući kolutajuće bačve preskačući preko njih. Rast popularnosti Donkey Konga doveo je do nastanka brojnih platformera fokusiranih na mehanike skoka. Donkey Kong se također smatra i prvom „hop and bop“ igrom, najčešćim podžanrom platformer igara. Hop and bop svoje ime dobiva u mehanici skoka na neprijatelja, time ga poražujući. Najpoznatija igra ovog podžanra jest Super Mario [10].

Samo 5 mjeseci nakon izlaska Donkey Konga objavljena je igra naziva Jump Bug. Ova igra prva je spojila mehanike platformi uz korištenje pucačkih mehanika. Jump Bug koristi horizontalni pomak prikaza te mogućnost pucanja u nadolazeće protivnike. Upravo ovaj horizontalni pomak omogućio je razvoj većih i kompleksnijih razina, pošto sada dizajneri više nisu bili ograničeni na fiksnu veličinu razine. Jump Bug pokrenuo je razvoj novog podžanra platformer igara, pod nazivom „run and gun“ [11].

Sredinom 1980-ih godina razvio se još jedan podžanr: puzzle platformer. Ovaj žanr koristi konvencionalne mehanike platformer igara koje spaja s elementima slagalica, time kreirajući igru čiji cilj nije poraziti neprijatelja ili doći do najviše razine, nego riješiti razne složene probleme ili slagalice. Najbolji primjer ovog žanra jest The Lost Vikings [10].

Slika 5: The Lost Vikings

3.2.2. 1990-e

U 1990-im platformer igre prelaze sa pseudo 3D na pravi 3D. Rendering ovih igara postao je veoma zahtjevan, stoga su za igranje ovih igara bila potrebna računala jaka za to doba. 1994. Exact je izbacio igru naziva Geograph Seal. To je bila potpuno 3D poligonalna shooter igra u prvom licu. Smatra se i prvom iskonski 3D action-shooter igrom sa „free roaming“ okolinom, ali je ostala nezamijećena na zapadu jer je izdana samo u Japanu. Sljedeće godine Exact izdaje Jumping Flash! za Sonyevu novu konzolu, Playstation. Prema Guiness World Recordsu Jumping Flash! smatra se prvom pravom platformer igrom u 3D-u [12].

1996. Nintendo izdaje Super Mario 64, koji je omogućio igračima puno veću slobodu kretanja 3D svijetom. Iz tog razloga Nintendo na svoj kontroler dodaje analognu palicu, koja postaje standard za skoro sve kontrolere. Super Mario 64 je revolutionizirao platformer žanr tako što sada više nije bio cilj doći do kraja levela, već je bilo potrebno ispuniti zadatke koji su bili potrebni za prolazak levela [13].

Razvoj 3D platformera postavio je novi zadatak pred developere tih igara. Naime, bila je potrebna inteligentna dinamička kamera koja bi slijedila glavnog igrača igre. To u 2D platformerima nije bio problem jer 2D platformeri imaju fiksni pogled zbog nedostatka treće dimenzije .

Slika 6: Super Mario 64

3.2.3. Moderni platformeri

Početkom trećeg tisućljeća platformer igre uvelike gube na popularnosti. 1998. godine njihov udio na tržištu iznosio je 15%, dok je 2002. taj udio spao na mizernih 2% [14]. 2002. godine Nintendo izbacuje svoju drugu 3D Super Mario igru, imena Super Mario Sunshine. Igra je bila kritizirana zbog kratke duljine, nemaštovito dizajniranih razina te sporosti samog gameplaya [15]. Ovaj žanr više nikada nije dosegao popularnost koju je nekad uživao [14].

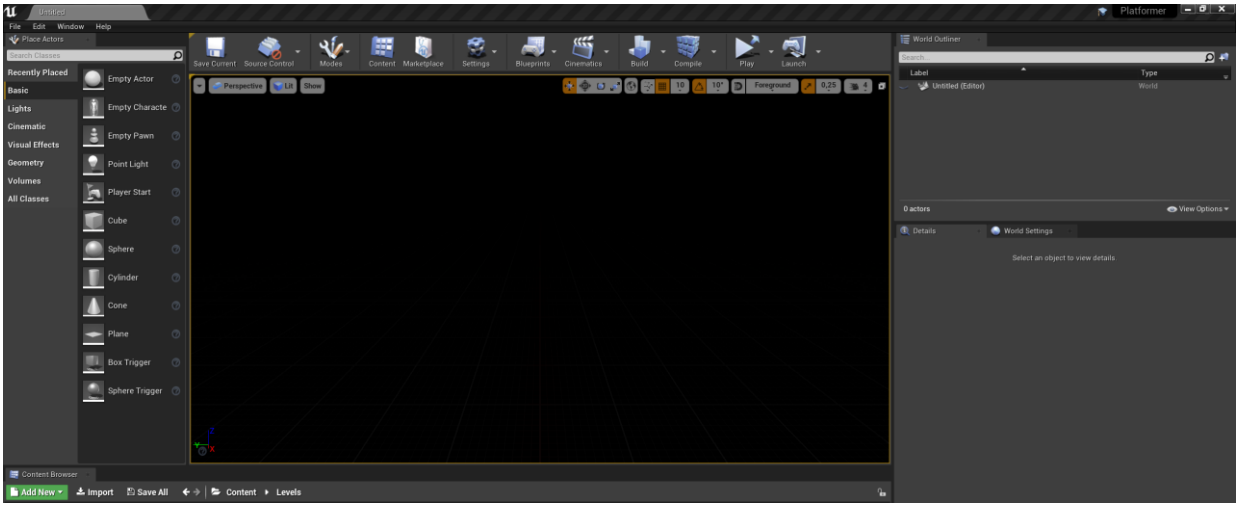
Najpoznatije platformer igre nastale u drugom desetljeću 21. stoljeća su New Super Mario Bros. 2, Ori and the Blind Forest i Sonic the Hedgehog 4.

Slika 7: Sonic the Hedgehog 4

3.3. Razvoj igre

3.3.1.Kreiranje razine

Nakon kreiranja projekta potrebno je kreirati prvi level. Njega se kreira klikom na File u izborniku te odabirom New Level i Empty Level. Pojaviti će se sljedeći prozor:

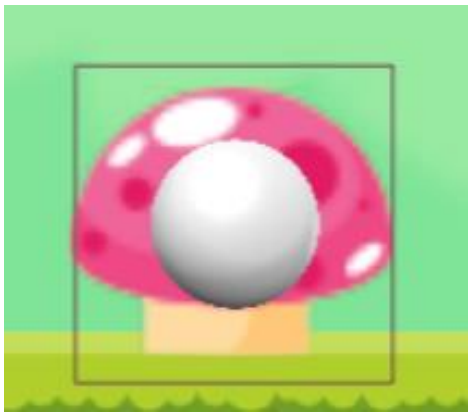


Slika 8:Level Editor

Zbog 2D te side – scroller svojstava igre y-os trodimenzionalnog sustava neće biti potrebna. Stoga se ortografski prikaz može postaviti na Left ili Right kako bi se prikazale samo x i y osi viewporta. Ovdje se mogu dodavati raznorazni objekti poput likova, platformi, bombi, ili drugih Actora.

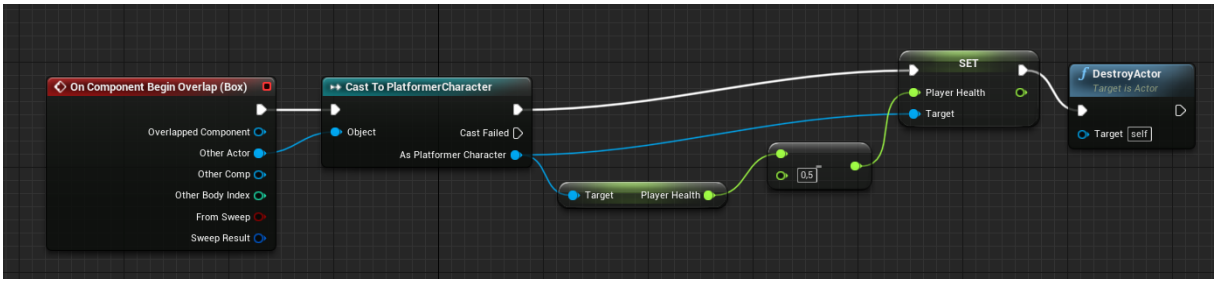
3.3.1.1. Kreiranje elemenata levela

Desni klik u Content Browseru > Blueprint Class kako bi bio kreiran novi element. Pod svojstvima u Level editoru nalazi se Sprite. Njega je potrebno postaviti kako bi ovaj element imao svoj izgled. Također je potrebno dodati Collision Box u Blueprint Editoru pod Components.



Slika 9: Otrovne gljive

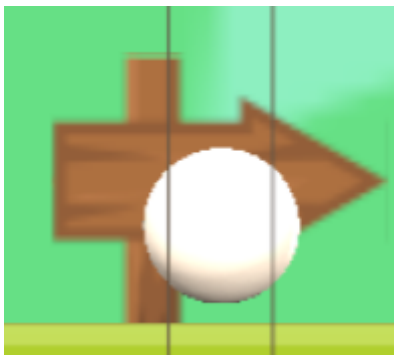
Recimo da želimo da ovaj element radi neki damage. U blueprint editoru potrebno je skriptirati jednostavan kod kojim će ovaj element napraviti damage te se nakon toga samouništiti.



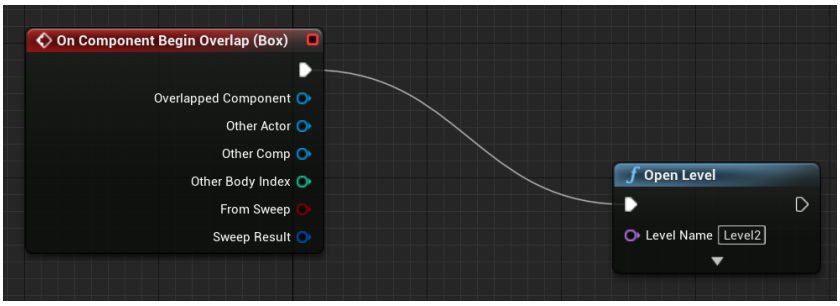
Slika 10: Logika otrovnih gljiva

Na isti način kreirani su ostali elementi levela. Za platforme je bitno napomenuti da je potrebno pod Details postaviti Collision Preset na BlockAllDynamic, kako bi glavni lik mogao stati na platformu.

Kreiran je i portal koji glavnog lika vodi prema sljedećem levelu.



Slika 11: Portal za prelazak na sljedeći level



Slika 12: Prelazak na sljedeći level

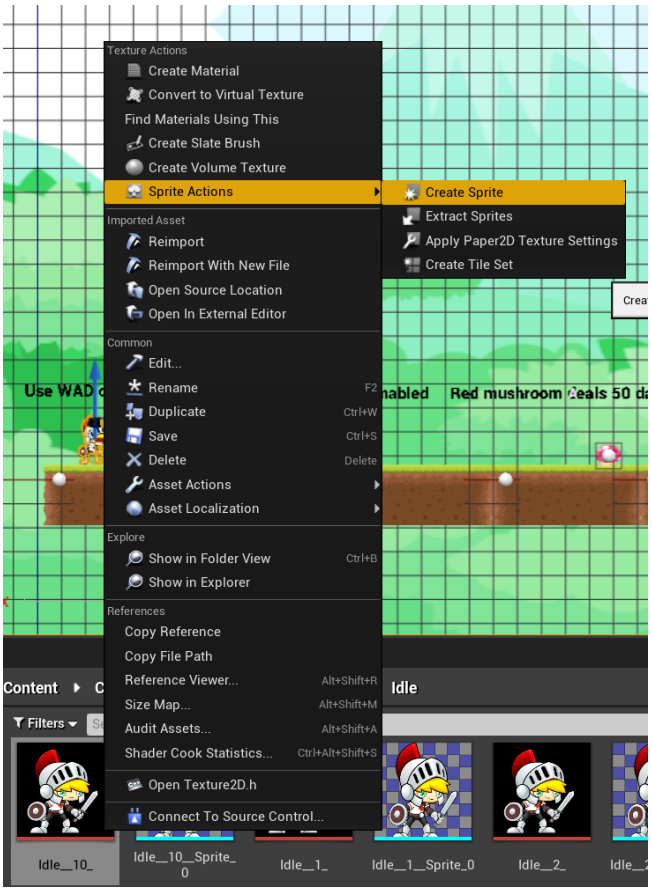
Slika 13: Level 1

3.3.2. Kreiranje glavnog lika

Ukoliko je prilikom kreiranja samog projekta odabrana opcija 2D Side – scroller, Unreal će sam generirati osnovnu funkcionalnost igre u obliku C++ koda. Tako Unreal automatski kreira i C++ klasu glavnog lika te header te klase.

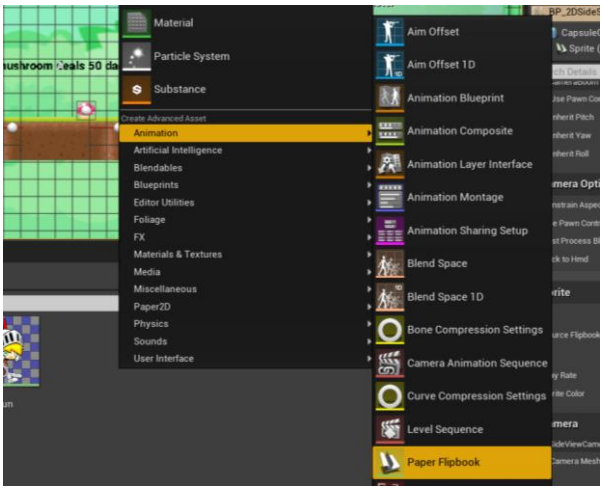
3.3.2.1. Animacije

Kako bi glavni lik imao izgled, potrebno mu je dodati Sprite svojstvo.



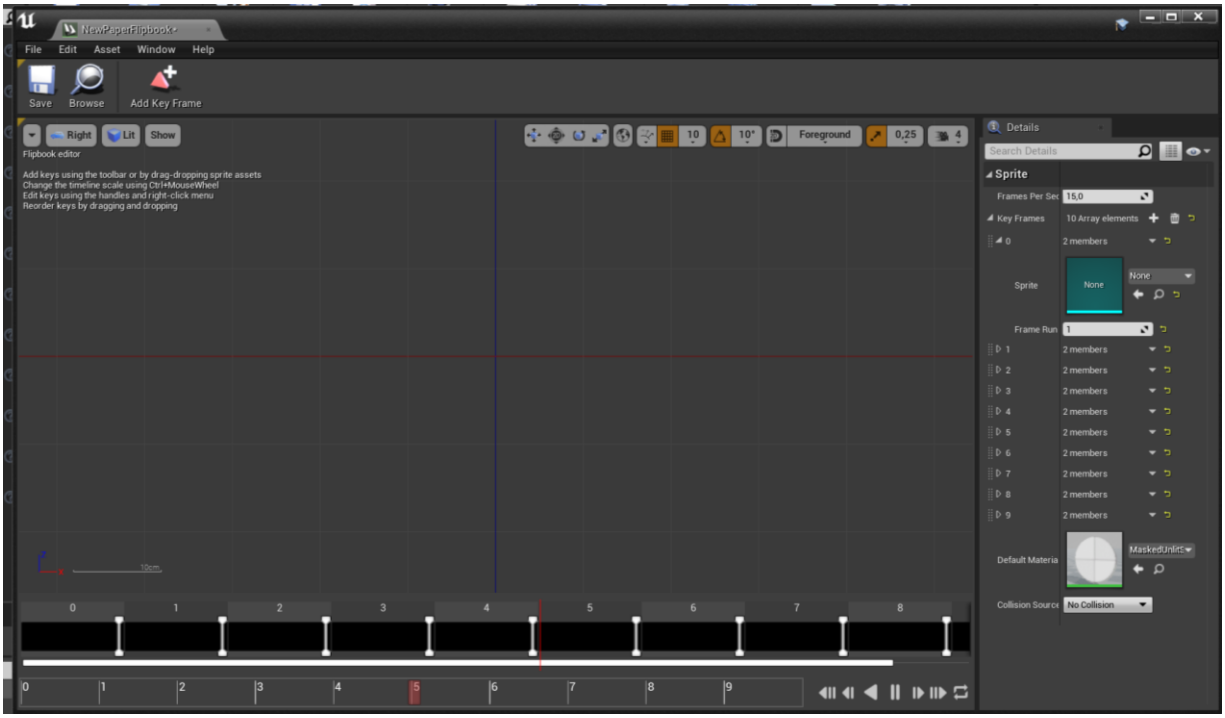
Slika 14: Kreiranje Spritea

Nakon importanja slika u projekt, što se postiže jednostavnom drag-and-drop akcijom, potrebno je iz tih slika izvući Sprite prema uputama na slici. Za potrebe animiranja lika poželjno je izvući barem deset Spritea kako bi animacija izgledala što uvjerljivije. Nakon toga potrebno je kreirati Flipbook. Desni klik u Content Browseru > Animation > Paper Flipbook.



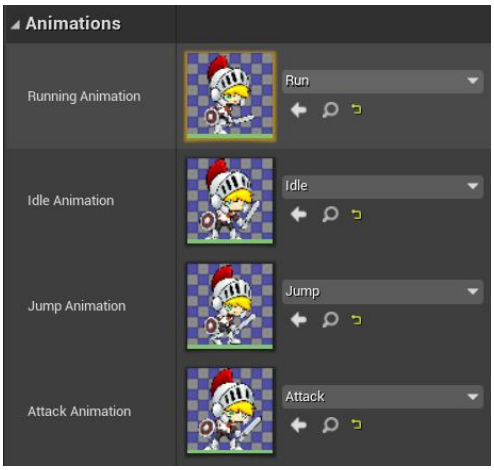
Slika 15: Kreiranje Flipbooka

Otvoriti će se Paper Flipbook Editor kao na slici:



Slika 16: Paper Flipbook Editor

Pod detaljima potrebno je dodati onoliko Key Frameova koliko jedna animacija sadrži slika. Zatim je svakom Key Frameu potrebno dodati pripadajuć Sprite. Ovisno o željenoj brzini izvođenja animacije, moguće je modificirati Frames Per Sec kako bi se ona ubrzala, odnosno smanjila. Klikom na Save Flipbook se sprema te ga je moguće implementirati. U Editoru glavnog lika pod Details sekcijom nalazi se kategorija Animations. Ovdje se animaciji dodjeljuje njezin Flipbook source.



Slika 17: Animacije

Važno je napomenuti kako se ovdje po defaultu nalaze samo Running Animation te Idle Animation. Kako bi bilo omogućeno dodjeljivanje drugih animacija, poput Jump ili Attack animacija, potrebno je u header klasi glavnog lika dodati nove animacije. To se omogućuje unosom sljedećeg koda:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
    class UPaperFlipbook* JumpAnimation;
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
    class UPaperFlipbook* AttackAnimation;
```

Sada se u editoru pod Details sekcijom u kategoriji Animations ne nalaze samo Idle i Running Animations, nego i Jump Animation te Attack Animation.

Zadnja važna stvar kod animacija jest određivanje u kojem trenutku će se koja animacija prikazivati. U klasi glavnog lika potrebno je kreirati funkciju koja će neprestano provjeravati stanja u kojem se glavni lik nalazi, te određivati prikladnu animaciju.

```
void APlatformerCharacter::UpdateAnimation()
{
    const FVector PlayerVelocity = GetVelocity();
    const float PlayerSpeedSqr = PlayerVelocity.SizeSquared();
    UPaperFlipbook* DesiredAnimation;
    DesiredAnimation = IdleAnimation;
    if (PlayerSpeedSqr > 0.0f) {
            DesiredAnimation = RunningAnimation;
    }
    if (GetCharacterMovement()->IsFalling())
    {
            DesiredAnimation = JumpAnimation;
    }
    if (isAttacking == true) {
            DesiredAnimation = AttackAnimation;
    }
    GetSprite()->SetFlipbook(DesiredAnimation);
}
```

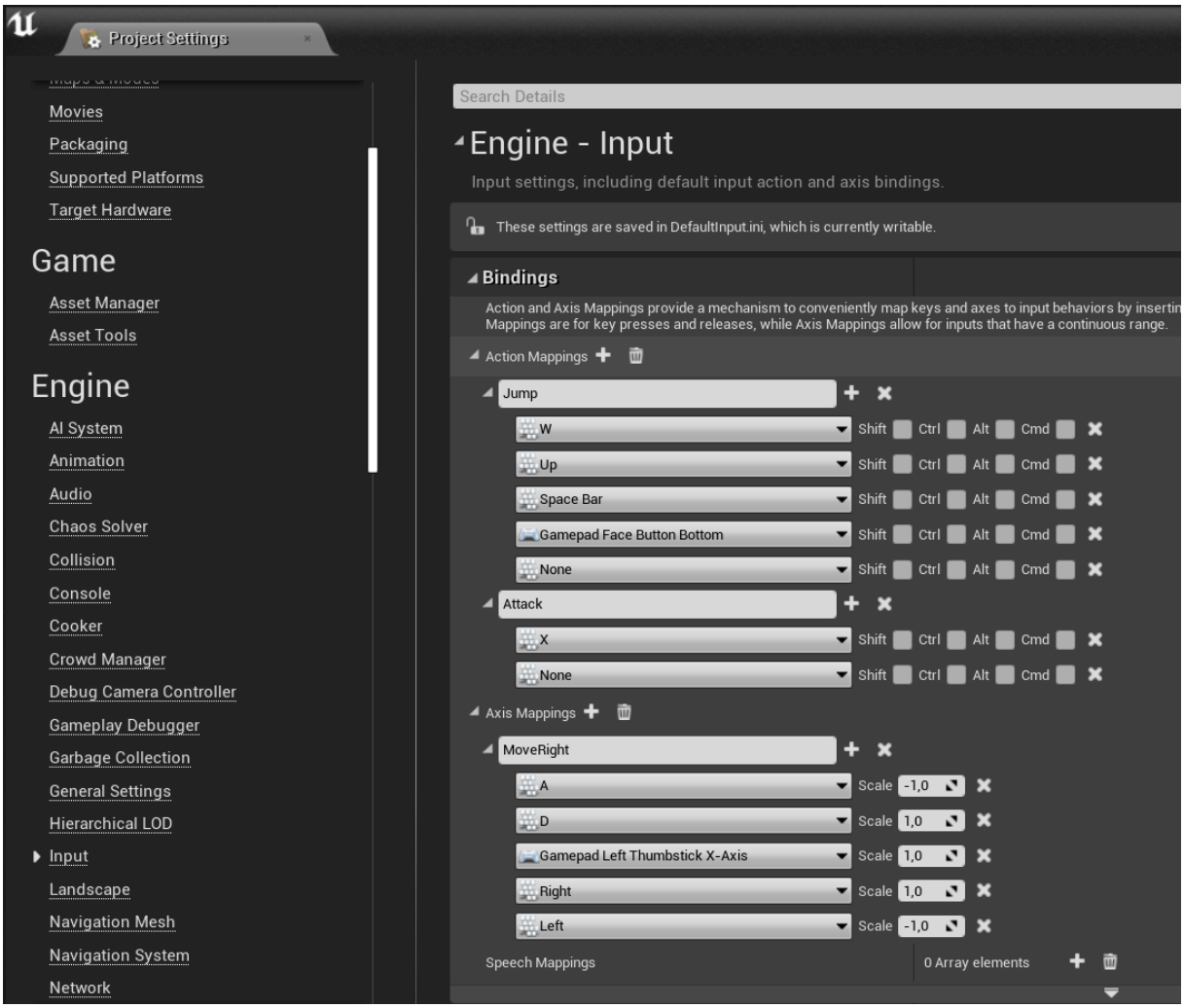
Ova funkcija prvo dohvaća brzinu kojom se glavni lik kreće. Ukoliko se glavni lik kreće brzinom većom od 0.0f, odnosno ne stoji na mjestu, željena animacija postavlja se na Running Animation. Sljedeće što ta funkcija provjerava jest nalazi li se lik u skoku. Ovdje funkcija željenu animaciju postavlja na Jump Animation neovisno o tome kreće li se glavni lik horizontalno ili ne, zbog toga što lik ne može samo skakati na mjestu, nego može i skakati i kretati se horizontalno istovremeno. Zadnje što funkcija provjerava jest napada li glavni lik trenutno protivnika. Ukoliko napada, željena animacija postavlja se na Attack Animation, neovisno o rezultatima prethodnih provjera, zbog toga što glavni lik može napasti dok se kreće horizontalno, dok skače, ili dok izvršava obje akcije istovremeno. Ukoliko sve ove provjere vrate negativan rezultat, željena animacija postavlja se na Idle Animation.

3.3.2.2. Kretanje

Za početak potrebno je postaviti input kontrole za kretanje glavnog lika. Te kontrole naređuju glavnom liku da se kreće, da skoči ili da napadne protivnika.

```
 void APlatformerCharacter::SetupPlayerInputComponent(class
UInputComponent* PlayerInputComponent)

{

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

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

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

     PlayerInputComponent->BindAction("Attack", IE_Pressed, this,
&APlatformerCharacter::Attack);

     PlayerInputComponent->BindTouch(IE_Pressed, this,
&APlatformerCharacter::TouchStarted);

     PlayerInputComponent->BindTouch(IE_Released, this,
&APlatformerCharacter::TouchStopped);

}
```

Sada će u editoru te kotrole biti prikazane te je moguće postaviti više željenih tipki na kontroleru koje bi pozivale funkcije prema gornjem kodu.



Slika 18: Postavke kontrola

Action Mapping input funkciju poziva samo jednom na key press event. U ovoj igri kao Action Mapping postavljene su akcije skoka i napada. Skok se može izvesti pritiskom na tipku W, strelicu Up, ili Spacebar, dok je napad moguć pritiskom na tipku X. Axis Mappings pozivaju funkcije sve dok ne prestanu biti pritisnute. Dakle glavni lik će se kretati desno sve dok igrač drži tipku D ili strelicu Right pritisnutom, odnosno lijevo ako igrač drži tipku A ili strelicu Left pritisnutom.

Sada te funkcije koje pozivaju input tipke moraju nešto sadržavati kako bi se glavni lik uistinu ponašao prema dobivenim naredbama.

```
void APlatformerCharacter::MoveRight(float Value)
{
    AddMovementInput(FVector(1.0f, 0.0f, 0.0f), Value);
}
```

Ovom funkcijom glavni lik se kreće horizontalno, po x-osi. Za sada je glavni lik uvijek okrenut prema desno, jer je njegov Sprite tako dizajniran. Stoga je nužna funkcija koja provjerava kreće li se glavni lik u lijevom ili desnom smjeru, te ovisno o rezultatu rotira glavnog lika.

```
 if (TravelDirection < 0.0f)
     {
            Controller->SetControlRotation(FRotator(0.0, 180.0f, 0.0f));
     }
else if (TravelDirection > 0.0f)
     {
            Controller->SetControlRotation(FRotator(0.0f, 0.0f, 0.0f));
     }
```

Funkcije `&ACharacter::Jump` i `&ACharacter::StopJumping` Unreal Engine automatski generira te se nalaze u klasi Character.cpp.

U klasi glavnog lika putem koda moguće je postaviti i modificirati željena svojstva kretanja glavnog lika.

```
GetCharacterMovement()->GravityScale = 2.0f;
```

Postavlja jačinu gravitacije koja utječe na glavnog lika prilikom padanja.

```
GetCharacterMovement()->AirControl = 0.80f;
```

Postavlja količinu moguće kretnje glavnog lika lijevo ili desno prilikom skoka.

```
GetCharacterMovement()->JumpZVelocity = 1000.f;
```

Postavlja brzinu skoka glavnog lika.

```
GetCharacterMovement()->GroundFriction = 3.0f;
```

Postavlja brzinu promjene smjera kretnje glavnog lika. Veća vrijednost omogućava brže promjene smjera.

```
GetCharacterMovement()->MaxWalkSpeed = 600.0f;
```

Postavlja maksimalnu brzinu kretanja glavnog lika.

```
GetCharacterMovement()->MaxFlySpeed = 600.0f;
```

Postavlja maksimalnu brzinu letenja glavnog lika.

```
GetCharacterMovement()->bConstrainToPlane = true;
     GetCharacterMovement()->SetPlaneConstraintNormal
(FVector(0.0f, -1.0f, 0.0f));
```

Limitira kretanje glavnog lika na XZ ravninu.

3.3.2.3. Kamera

Za početak je potrebno kreirati root (kapsulu kolizije) glavnog lika na koji će se povezati CameraBoom element.

```
GetCapsuleComponent()->SetCapsuleHalfHeight(96.0f);
GetCapsuleComponent()->SetCapsuleRadius(40.0f);
```

Postavke CameraBoom elementa su postavljene na sljedeća svojstva:

```
CameraBoom=CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"
));
CameraBoom->SetupAttachment(RootComponent);
CameraBoom->TargetArmLength = 500.0f;
CameraBoom->SocketOffset = FVector(0.0f, 0.0f, 75.0f);
CameraBoom->SetUsingAbsoluteRotation(true);
CameraBoom->bDoCollisionTest = false;
CameraBoom->SetRelativeRotation(FRotator(0.0f, -90.0f, 0.0f));
```

Zatim se kreira ortografska kamera u odnosu na glavnog lika na sljedeći način:

```
SideViewCameraComponent=CreateDefaultSubobject<UCameraComponent>(TEXT("S
ideViewCamera"));
SideViewCameraComponent->ProjectionMode =
ECameraProjectionMode::Orthographic;
SideViewCameraComponent->OrthoWidth = 2048.0f;
SideViewCameraComponent->SetupAttachment(CameraBoom,
USpringArmComponent::SocketName);
```

Zadnja stvar vezana za kameru jest onemogućiti automatsku rotaciju kamere, glavnog lika te komponenti kamere:

```
CameraBoom->SetUsingAbsoluteRotation(true);

SideViewCameraComponent->bUsePawnControlRotation = false;

SideViewCameraComponent->bAutoActivate = true;

GetCharacterMovement()->bOrientRotationToMovement = false;
```

Svi ovi dijelovi koda nalaze se u konstruktoru klase glavnog lika.

3.3.2.4. Health sistem

Gotovo svaka 2D platformer igra sadrži neki oblik Health sistema. Kako bi ova igra imala jedan takav sistem, potrebno je u header klase glavnog lika postaviti sljedeću liniju koda:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Health")
            float PlayerHealth;
```

Sada je moguće modificirati health i kroz editor i kroz kod. U ovom slučaju inicijalna vrijednost iznosi 1.00f, a postavljena je u kostruktoru klase glavnog lika.

```
PlayerHealth==1.00f;
```

3.3.2.5. Attack sistem

Ova igra sadrži combat sistem, stoga je potrebno glavnom liku omogućiti spostobnost napadanja protivnika. U header je potrebno dodati sljedeći kod:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Attacking")
            bool isAttacking;
void Attack();
```

Bool isAttacking je potreban kako bi igra mogla prepoznati pokušava li glavni lik napasti protivnika, te ovisno o vraćenom rezultatu mijenja animaciju glavnog lika, ili pak poziva funkciju Attack. Funkcija Attack taj bool postavlja na true ukoliko dođe do poziva te funkcije.

3.3.2.6. Death i respawn

Već je spomenuto kako glavni lik sadrži health sistem. Implementirana je funkcija koja neprestano provjerava ima li glavni lik manje ili jednako 0.0f healtha, te provjerava nalazi li se lik u ponoru, tj nalazi li se lik ispod najniže platforme na koju može skočiti.

```
if (PlayerHealth <= 0.0f) {
     _sleep(0.5);
     UGameplayStatics::OpenLevel(GetWorld(),
   FName(*GetCurrentMapName()));
     }
if (location.Z < -200) {
     _sleep(0.5);
     UGameplayStatics::OpenLevel(GetWorld(),
FName(*GetCurrentMapName()));
     }
```

Ukoliko bilo koja od ovih provjera vrati true, igra se pauzira na pola sekunde, te zatim poziva funkciju

```
UGameplayStatics::OpenLevel(GetWorld(),FName(*GetCurrentMapName()));
```

Ova funkcija dolazi već predefinirana te joj je samo potrebno proslijediti ime levela kojeg želimo otvoriti u parametru. Kako bi se postigla respawn mehanika, potrebno je pronaći naziv levela u kojem se glavni lik trenutno nalazi. To se postiže korištenjem sljedeće funkcije:

```
FString APlatformerCharacter::GetCurrentMapName()
{
     UWorld* MyWorld = GetWorld();
     FString CurrentMapName = MyWorld->GetMapName();
     return CurrentMapName;
}
```

Sve provjere za koje je nužno neprestano ih provjeravati pozivaju se unutar APlatformerCharacter::UpdateCharacter() funkcije, koja se pak neprestano poziva preko Tick funkcije:

```
void APlatformerCharacter::Tick(float DeltaSeconds)
{
     Super::Tick(DeltaSeconds);
     UpdateCharacter();
}
```

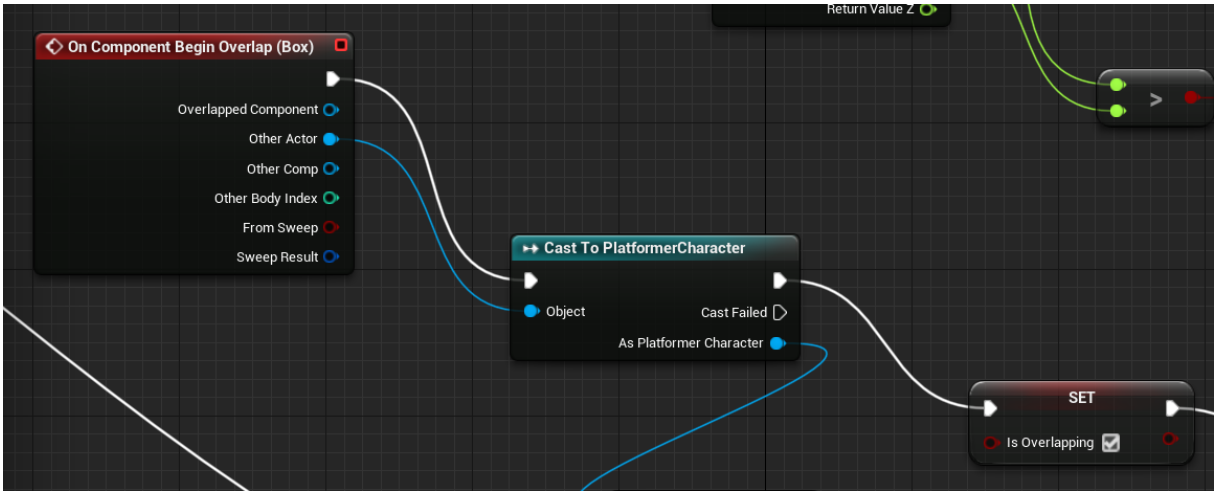
3.3.3. Zombiji

Za razvoj glavnog lika korišten je pristup preko C++ koda. Kako Unreal Engine 4 ima mogućnost i vizualnog skriptiranja preko Blueprint sustava, upravo taj sustav je korišten za razvoj protivnika kako bi se pokazala i ta strana Unreal Enginea. Zombiji se kreiraju jednako kao i glavni lik, a na isti način se rade i animacije.

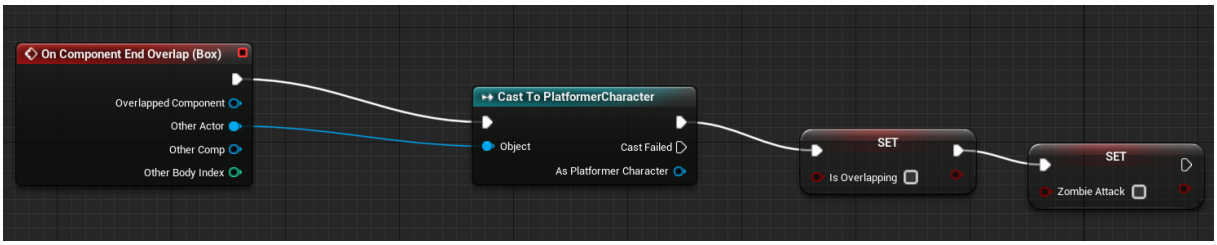
Slika 19: Zombi

3.3.3.1. Praćenje glavnog lika

Najprije je potrebno saznati nalazi li se zombi u koliziji s glavnim likom, točnije preklapaju li se njihovi Box Collision elementi. Ukoliko da varijabla IsOverlapping se postavlja na true, a u suprotnom na false.

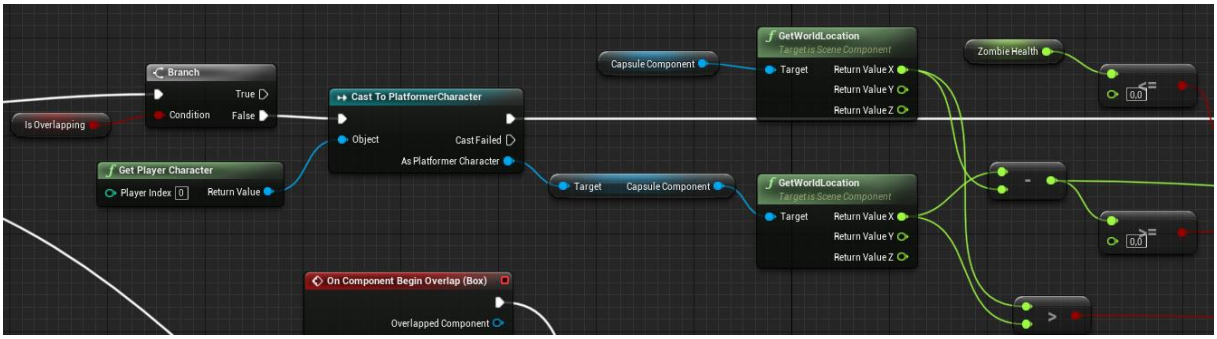


Slika 20: Početak preklapanja Collision Boxeva



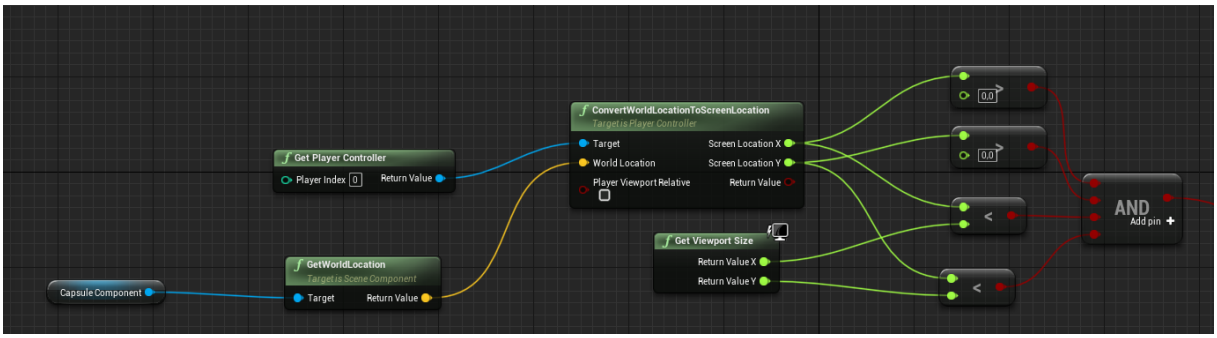
Slika 21: Kraj preklapanja Collision Boxeva

Nakon toga uspoređuju se lokacije Zombija i glavnog lika i to se izvršava samo u slučaju ako je varijabla IsOverlapping false.

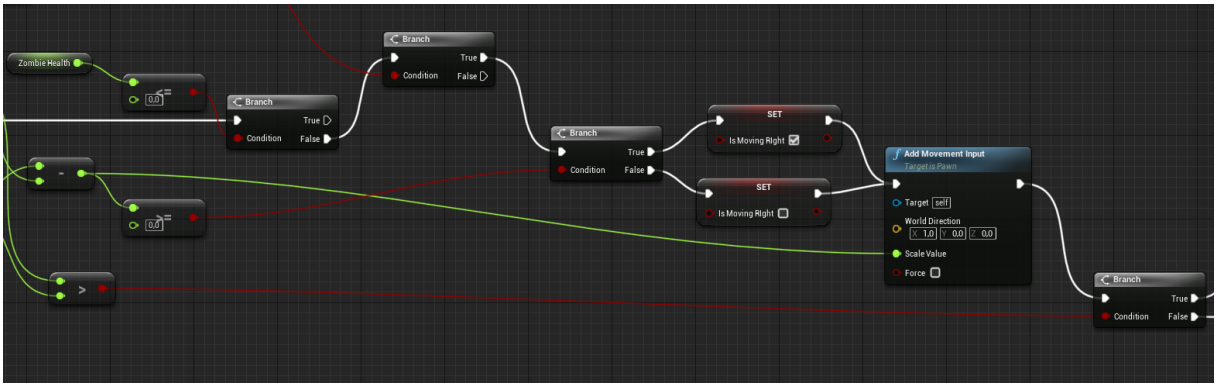


Slika 22: Usporedba lokacija glavnog lika i zombija

Zatim se provjerava nalazi li se zombi u viewportu igre, to jest može li ga igrač trenutno vidjeti na ekranu. Ukoliko ne, zombi stoji na mjestu, a ukoliko da, i usporedba lokacija glavnog lika i zombija vraća da se oni ne nalaze jedan pored drugog, dodaje se MovementInput zombiju po x-osi i to u smjeru prema glavnom liku.

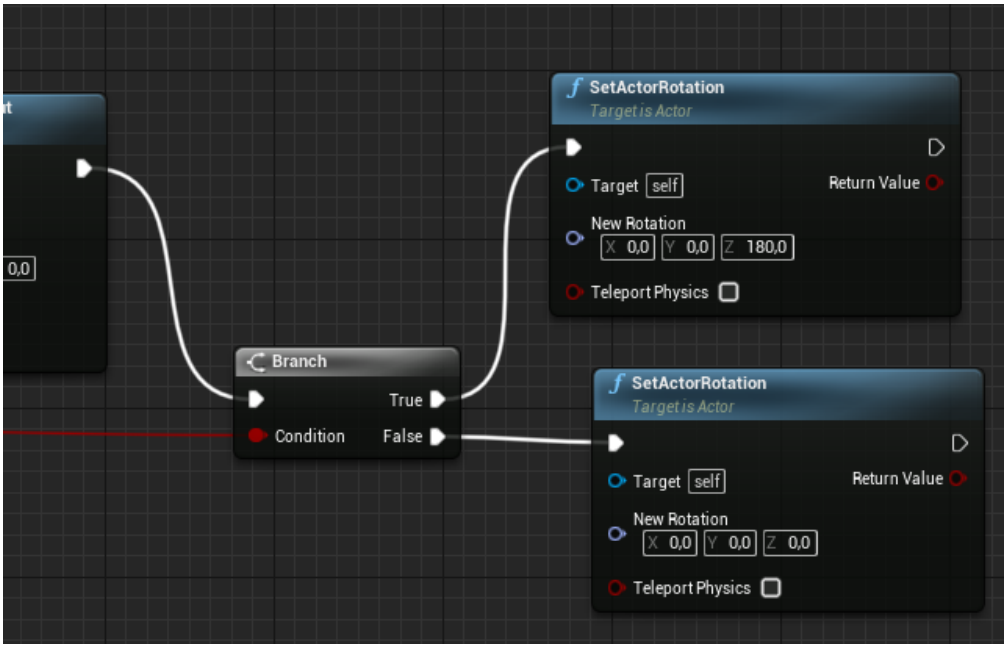


Slika 23: Provjera nalazi li se zombi u viewportu



Slika 24: Postavljanje zombijeve kretnje

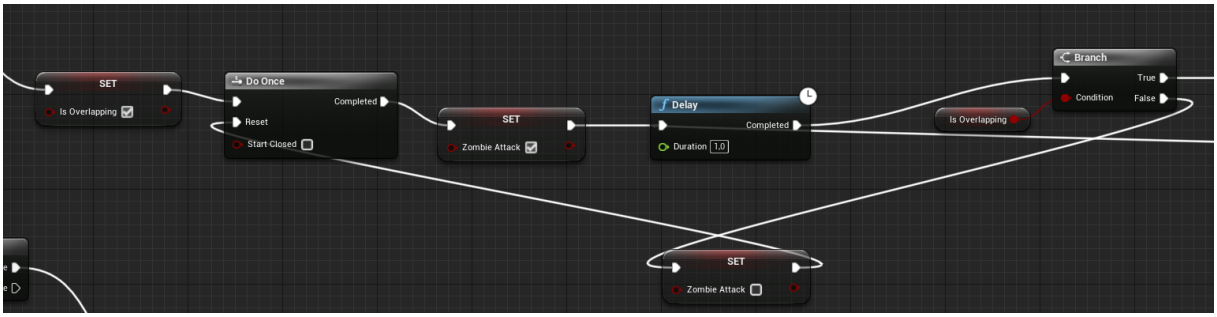
Kao i kod glavnog lika, potrebno je implementirati rotaciju zombija. Trenutno je zombi uvijek okrenut prema desno. Korištenjem provjere iz usporedbe lokacija zombija i glavnog lika uzima se njezina povratna vrijednost te se ovisno o njoj postavi rotacija lika na (0, 0, 180), ili na (0,0,0). Dakle, ako je vrijednost x lokacije zombija veća od vrijednosti x lokacije glavnog lika, to znači da se zombi nalazi nadesno u odnosu na glavnog lika. Kako je zombi po defaultu okrenut prema desno, potrebno je rotirati ga za 180 stupnjeva po z-osi, kako bi bio okrenut nalijevo. Kada glavni lik preskoči zombija, te zombi sada bude nalijevo od glavnog lika, provjera vraća false, te se u tom slučaju rotacija postavlja na defaultnu vrijednost, odnosno zombi je ponovno okrenut prema desno.



Slika 25: Postavljanje zombijeve rotacije

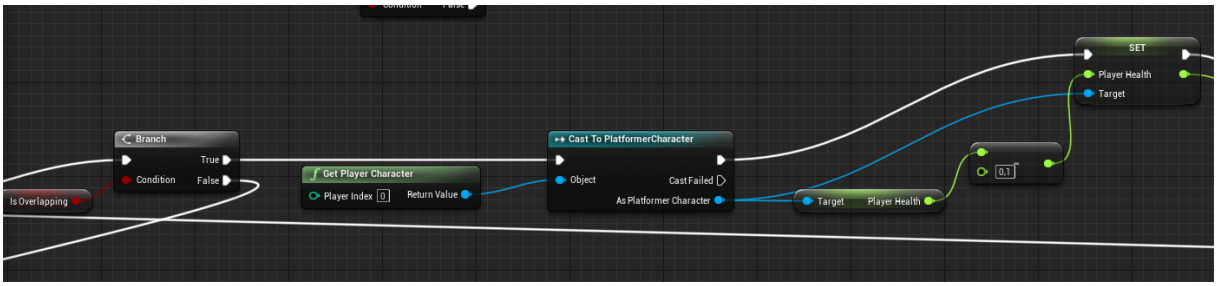
3.3.3.2. Napadanje glavnog lika

Za sad je implementirano da kada zombi bude vidljiv igraču na ekranu, počne pratiti glavnog lika. Međutim, kada dođe do njega, jednostavno stane na mjestu. Kako bi zombijev combat sistem bio implementira, potrebno je vratiti se na Component Overlap event, gdje se IsOverlapping postavlja na true. Dakle, kada se Collision Boxevi glavnog lika i zombija preklope, tada zombi mora početi napadati glavnog lika. Postavlja se varijabla ZombieAttack na true. Delay postavlja svojstvo zombija da može napasti jednom svaku sekundu. Ponovno se provjerava preklapaju li se njihovi Collision Boxevi. Ukoliko ne, ZombieAttack varijabla se postavlja na false.



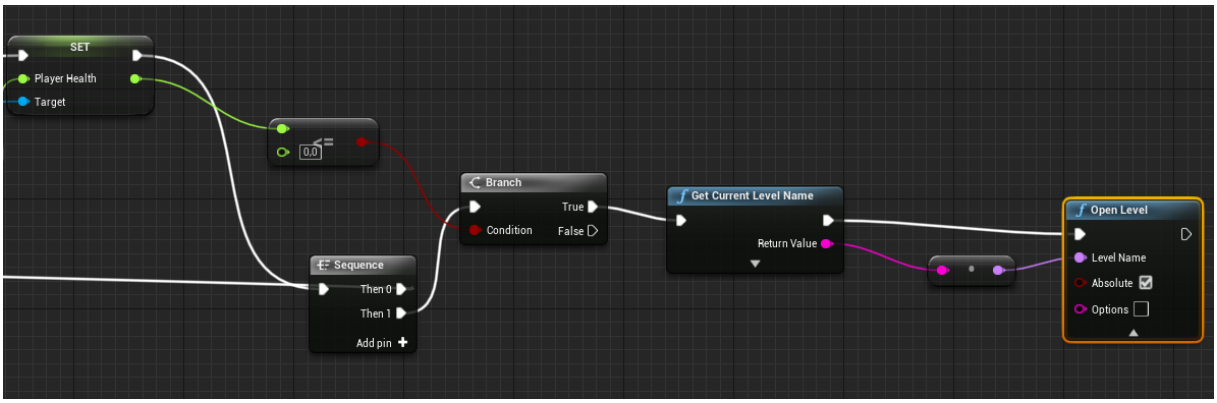
Slika 26: Zombijev napad

Ukoliko se Collision Boxevi i dalje preklapaju, odnosno zombi je uspješno napao glavnog lika, glavnom liku se skida 0.1f healtha.



Slika 27: Smanjenje healtha glavnog lika

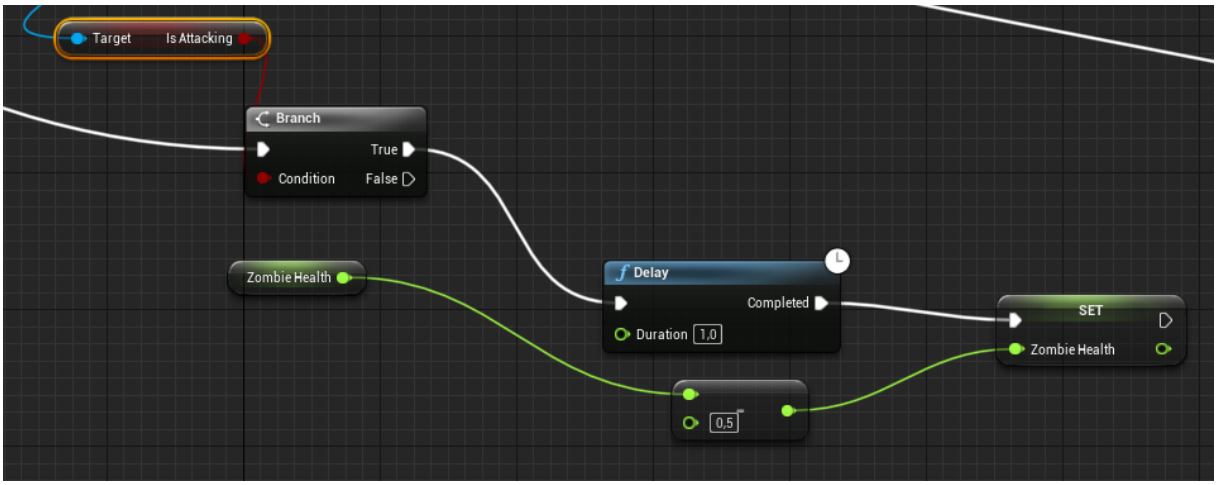
Ako je zombi uspješno ubio glavnog lika, tada se resetira level.



Slika 28: Reset levela

3.3.3.3. Napadanje zombija

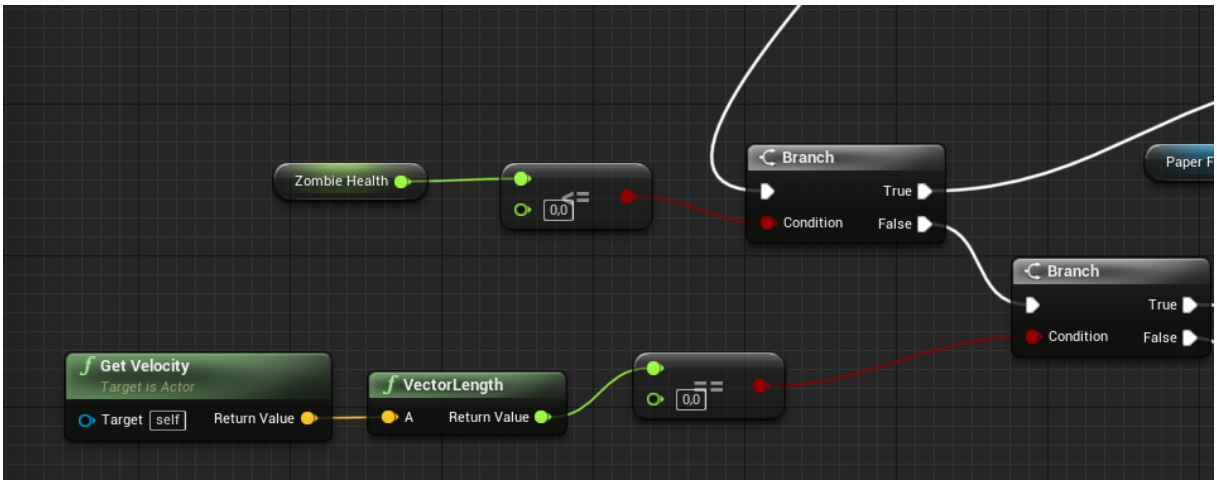
Već je spomenuta Attack funkcija glavnog lika. Ukoliko se Collision Boxevi zombija i glavnog lika preklapaju, te je IsAttacking glavnog lika postavljen na true, smanjuje se zombijev health za 0.5f. Također je postavljen delay kako bi igrač zombija mogao napasti jednom svake sekunde.



Slika 29: Smanjenje zombijevog healtha

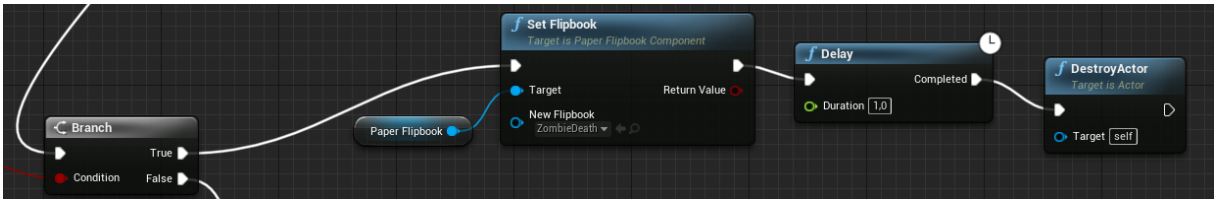
3.3.3.4. Animacije

Kao i kod glavnog lika, implementiran je algoritam koji odabire prihvatljivu animaciju za trenutno stanje zombija. Na početku se dohvaća health te brzina kretanja glavnog lika.



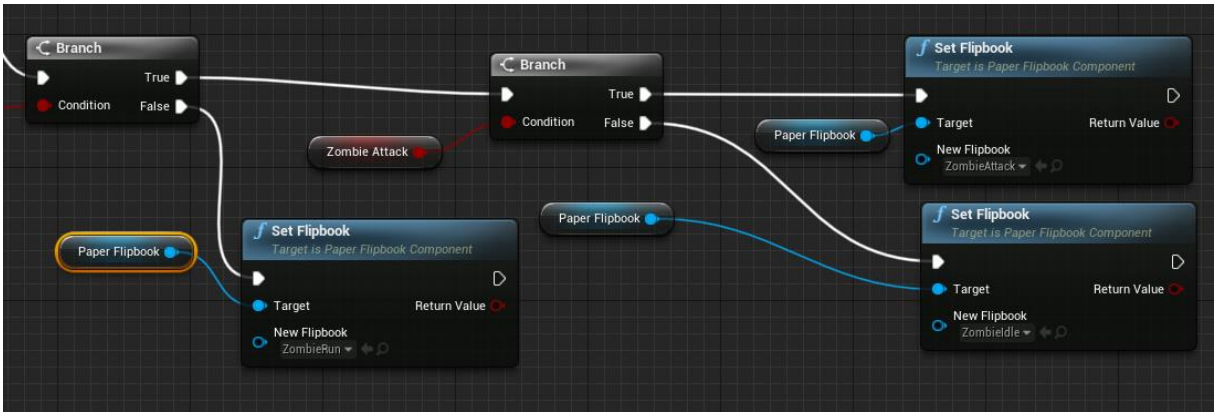
Slika 30: Provjera zombijevog healtha i brzine kretanja

Ukoliko je zombijev health manji ili jednak nuli, animacija se postavlja na ZombieDeath, koja se izvodi jednu sekundu te se nakon toga zombi uništi.



Slika 31: Postavljanje ZombieDeath animacije i samouništavanje

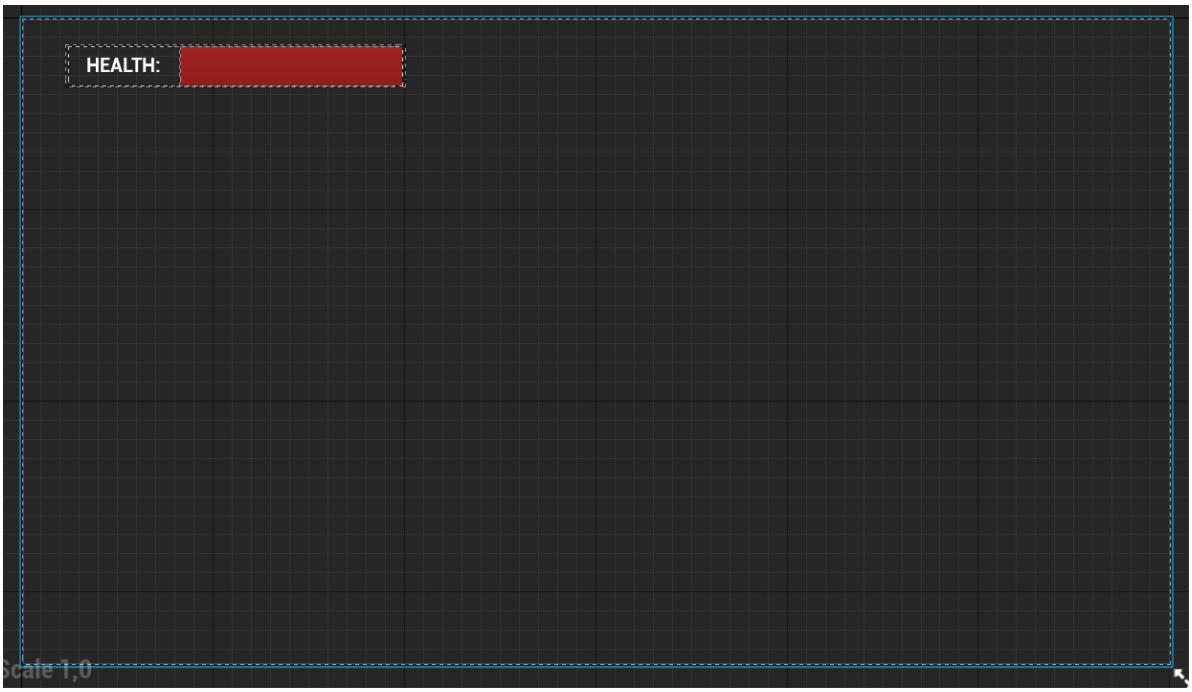
Ukoliko je zombijev health pak veći od nule, provjerava se kretnja zombija. Ukoliko se on kreće, animacija se postavlja na ZombieRun. U suprotnom, provjerava se napada li zombi glavnog lika. Ukoliko da, animacija se postavlja na ZombieAttack, a ukoliko ne, postavlja se na ZombieIdle.



Slika 32: Postavljanje animacije ovisno o zombijevom stanju

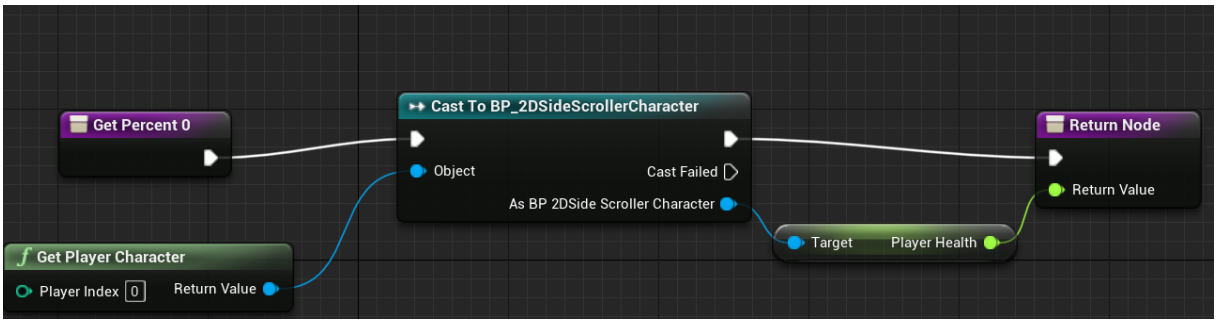
3.3.4. Grafičko sučelje

Grafičko sučelje koje prikazuje health glavnog lika kreira se slično kao i ostali elementi. Desni klik u Content Browseru > User Interface > Widget Blueprint. Kreirano je prozirno grafičko sučelje koje prikazuje trenutni health glavnog lika.



Slika 33: Prikaz healtha glavnog lika

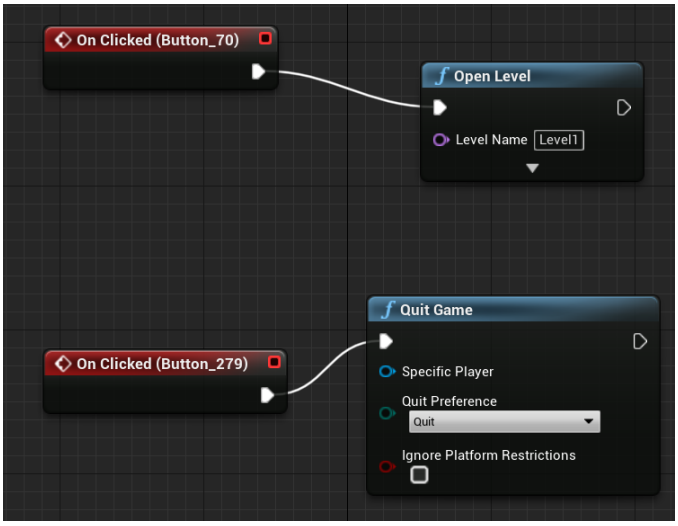
Kako bi se ažurirao te ispravno prikazivao health, potrebno je dodati sljedeći kod u Blueprint editor:



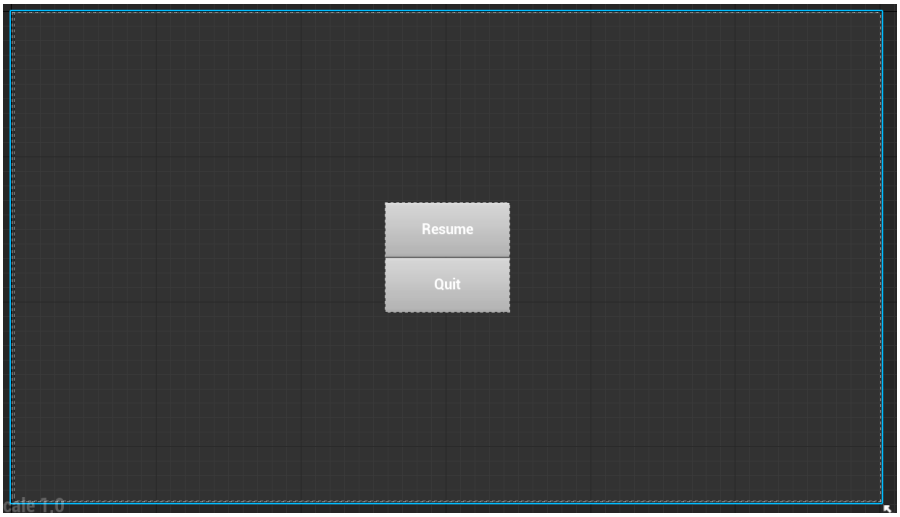
Slika 34: Povezivanje healtha sa sučeljem

Na isti način kreirani su i početni meni te meni pauze.

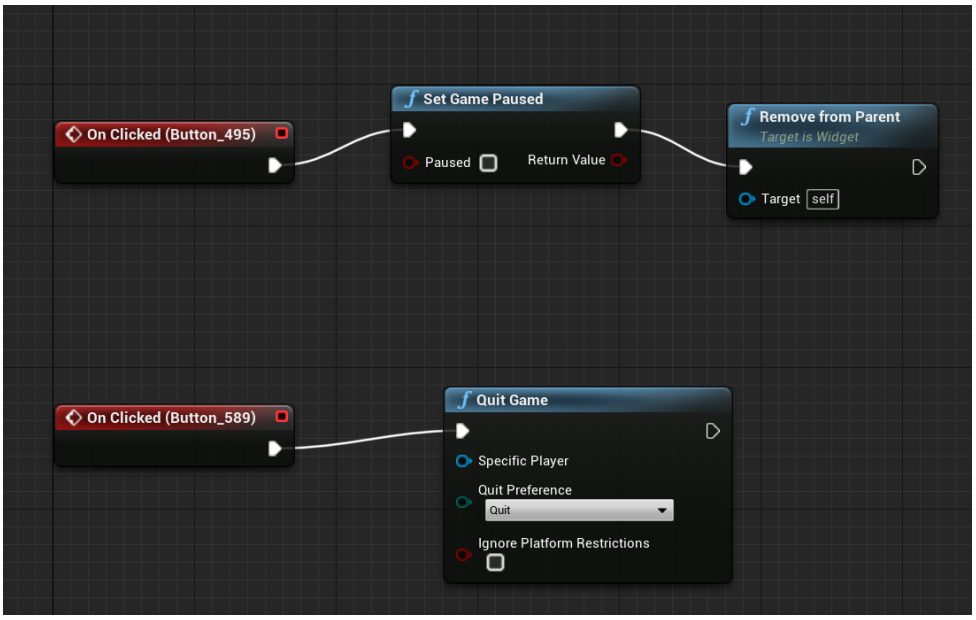
Slika 35: Početni meni



Slika 36: Logika početnog menija



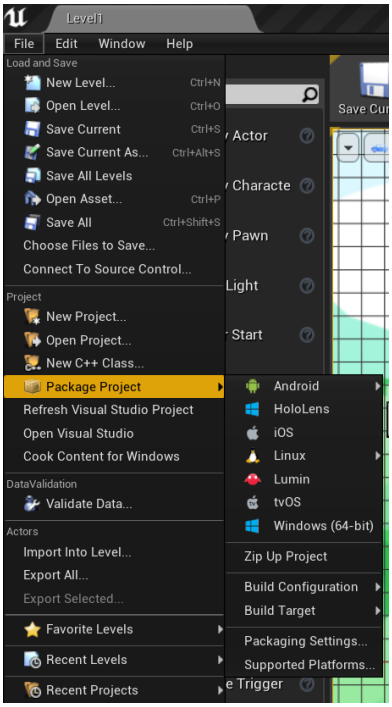
Slika 37: Meni pauze



Slika 38: Logika menija pauze

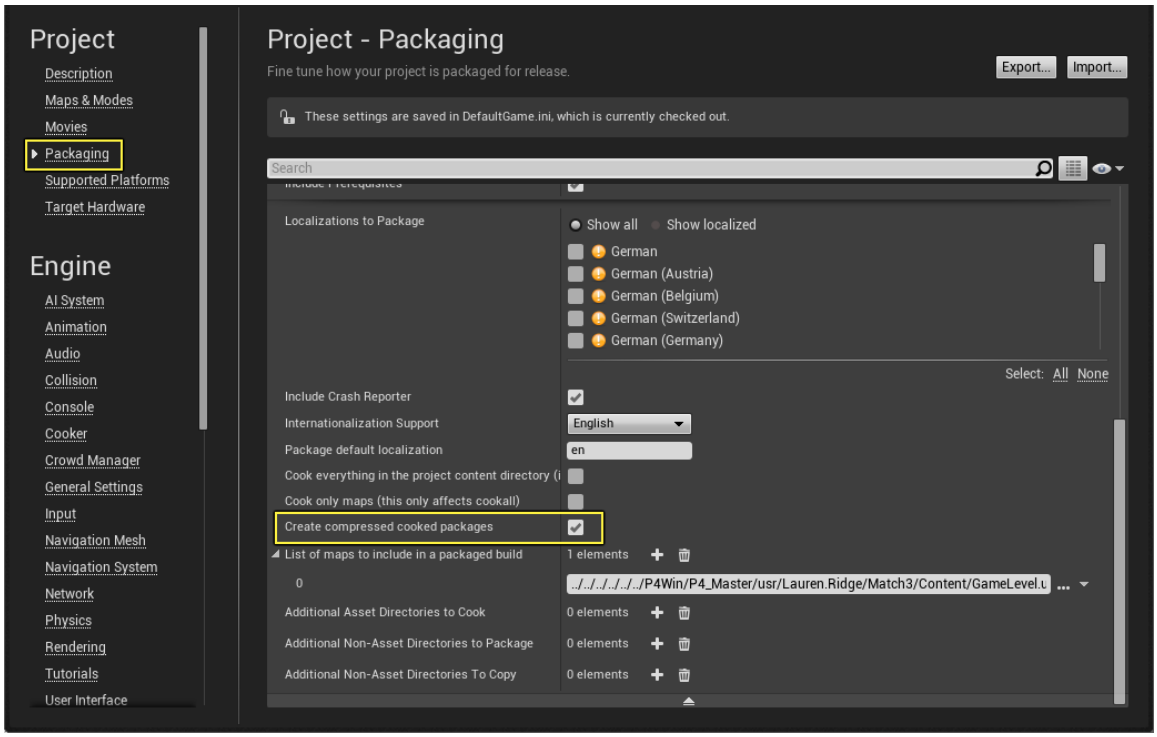
3.3.5. Kreiranje .exe datoteke

Nakon završetka dizajniranja i izrade igre, potrebno ju je pripremiti za masovnu upotrebu. Odabirom File > Package Project ponude se razne platforme za koje je moguće razviti finalni produkt. Za potrebe ovog završnog rada odabrana je Windows(64-bit) opcija.



Slika 39: Pakiranje projekta

Međutim, po inicijalnim postavkama Packaginga ovaj projekt zauzima 700MB. To je moguće značajno smanjiti. Odabirom Edit > Project Settings pod kategorijom Packaging nalazi se opcija Create compressed cooked packages. Ova opcija je značajno smanjila veličinu igrinog foldera, koja sada iznosi 120MB..



Slika 40: Kompresiranje datoteka

4. Zaključak

Ovaj rad u potpunosti je dočarao izradu jedne 2D platformer računalne igre. Detaljno je opisan, od samih početaka izrade igre pa sve do njenog završnog pripremanja za korištenje. Rad je poslužio svojoj svrsi, uvelike mi je poboljšao znanje C++ programskog jezika te me uveo u svijet videoindustrije. Naravno da ovdje ima puno prostora za napredak, ali bez početka nema ni napretka.

Unreal Engine 4 te Visual Studio pokazali su se moćnim, ali relativno jednostavnim alatima za razvoj jedne platformer računalne igre. Iako mi je ovo bio prvi dodir s Urealom, njegova intuitivnost olakšavala mi je snalaženje u njemu, a dokumentacija je sadržavala gotovo sve odgovore na moja pitanja.

Izrada ovog završnog rada omogućila mi je bolje razumijevanje rada računalnih igara, te sam sada sposoban samostalno izraditi računalnu igru slične jednostavnosti, a ako to znanje budem konstantno usavršavao, možda ću jednoga dana raditi na području razvoja računalnih igara. Izrazito sam zadovoljan dobivenim rezultatom.

Popis literature

[1] J. Lee, Learning Unreal Engine Game Development, Birmingham: Packt Publishing, 2016.

[2] Epic Games, »StaticMeshes,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Content/Types/StaticMeshes/index.html. [Pristupano 30. 6. 2020.].

[3] Epic Games, »LevelEditor,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/UI/LevelEditor/index.html. [Pristupano 30. 6. 2020.].

[4] Epic Games, »Levels,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Levels/index.html. [Pristupano 30. 6. 2020.].

[5] Epic Games, »Actors,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Actors/index.html. [Pristupano 30. 6. 2020.].

[6] Epic Games, »MaterialEditor,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/Editor/Interface/index.html. [Pristupano 30. 6. 2020.].

[7] Epic Games, »BlueprintEditor,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Blueprints/Editor/index.html. [Pristupano 30. 6. 2020.].

[8] Epic Games, »Blueprints,« [Na internetu]. Dostupno: https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/index.html. [Pristupano 30. 6. 2020.].

[9] M. Klappenbach, »What is a platform game?,« 27. 11. 2019.. [Na internetu]. Dostupno: https://www.lifewire.com/what-is-a-platform-game-812371. [Pristupano 30. 6. 2020.].

[10] racketBOY, »Platforming Games,« [Na internetu]. Dostupno: http://www.racketboy.com/retro/platforming-games-101-all-you-need-to-know. [Pristupano 30. 6. 2020.].

[11] Retro-Sanctuary, »Run and Gun Games,« [Na internetu]. Dostupno: http://retro-sanctuary.com/Run%20and%20gun%20games.html. [Pristupano 30. 6. 2020.].

[12] T. Fahs, »the-nextlevel,« 26. 11. 2006. [Na internetu]. Dostupno: https://web.archive.org/web/20160129224512/http://www.the-nextlevel.com/review/retro/geograph-seal-x68000/. [Pristupano 30. 8. 2020].

[13] »Wikipedia: Super Mario 64,« [Na internetu]. Dostupno: https://en.wikipedia.org/wiki/Super_Mario_64. [Pristupano 30. 8. 2020].

[14] D. Boutros, »GamaSutra,« 4. 8. 2006. [Na internetu]. Dostupno: https://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination_of_.php. [Pristupano 30. 8. 2020.].

[15] S. Maiorana, »The Jaded Gamer: Super Mario Sunshine,« 25. 4. 2004. [Na internetu]. Dostupno: https://web.archive.org/web/20051125062547/http://www.thejadedgamer.net/review_gcn_supermariosunshine.shtml. [Pristupano 30. 8. 2020.].

Popis slika

Izvori slika

- Slika 1: https://docs.unrealengine.com/Images/Engine/UI/LevelEditor/DefaultInterface_Windows.webp
- Slika 2: https://docs.unrealengine.com/Images/Engine/Rendering/Materials/Editor/Interface/MaterialEditor_Windows.webp
- Slika 3: https://docs.unrealengine.com/Images/Engine/Blueprints/Editor/blueprint_editor_example.webp
- Slika 4: https://en.wikipedia.org/wiki/Donkey_Kong#/media/File:Donkey_Kong_NES_Screenshot.png
- Slika 5: https://upload.wikimedia.org/wikipedia/en/thumb/6/61/TheLostVikings.png/200px-TheLostVikings.png
- Slika 6: https://venturebeat.com/wp-content/uploads/2016/06/supermario64.png?fit=800%2C560&strip=all
- Slika 7: https://www.microsoft.com/en-in/p/sonic-the-hedgehog-4-episode-i/bqhctfq14dfk?activetab=pivot:overviewtab